

DEPENDABILITY ASSESSMENT OF WIRELESS SENSOR
NETWORKS WITH FORMAL METHODS

By
Alessandro Testa

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
“FEDERICO II” UNIVERSITY OF NAPLES
VIA CLAUDIO 21, 80125 – NAPOLI, ITALY
APRIL 2013

© Copyright by Alessandro Testa, 2013

“A mio padre che ha avuto sempre fiducia in me”

Table of Contents

Table of Contents	iii
List of Tables	vi
List of Figures	vii
List of Listings	ix
Acknowledgements	x
Introduction	1
1 Dependability Issues of Wireless Sensor Networks	7
1.1 Wireless Sensor Networks	7
1.2 Dependability Threats in Wireless Sensor Networks	9
1.3 Critical WSN-based Applications	13
1.3.1 WSN Dependability Metrics	15
1.3.2 Environmental Monitoring	16
1.3.3 Security Monitoring	17
1.3.4 Ambient Intelligence	18
1.3.5 Health Monitoring Systems	21
1.4 FMEA of a WSN-based Monitoring System	25
1.5 Open issues	32
2 Related Research	34
2.1 Experimental approaches	34
2.2 Simulative approaches	36
2.2.1 Simulators	38
2.3 Analytical approaches	39
2.4 Formal approaches	41
2.4.1 Model Checking	41

2.4.2	Linear Temporal Logic (LTL)	43
2.4.3	Situation Calculus	44
2.4.4	Event Calculus	46
2.4.5	Formal approaches for WSN	48
2.5	Comparison of Related Work	51
2.6	Discussion	55
2.7	Workflow towards a Methodology for Dependable WSN-based Applications	56
2.7.1	Informal Modeling	57
2.7.2	General Correctness Specification Modeling	58
2.7.3	Structural Correctness Modeling	60
2.7.4	Verification/Testing	61
3	Static Verification	62
3.1	Rationale of the static verification technique	62
3.2	General Correctness Specification	64
3.2.1	Isolation event	65
3.2.2	Packet Loss event	69
3.2.3	Battery Exhaustion event	72
3.3	Structural Specification	74
3.4	Types of Analysis	76
3.4.1	What-if Analysis	76
3.4.2	Robustness Checking	77
3.5	Application Scenario	81
3.5.1	Structural Specification generated by Event Calculus	83
3.5.2	Outcome and Metrics computation	85
4	Dynamic Verification	89
4.1	Rationale of the dynamic verification technique	89
4.2	Runtime Verification	91
4.2.1	Reactive Event Calculus	93
4.3	Event Capture in a Dynamic Verification context	93
4.3.1	Intra-WSN	95
4.3.2	Gateway Services	95
4.3.3	Remote Center Services	96
4.3.4	External Applications	96
4.3.5	System Monitors	96
4.4	Application Scenario	99
5	The ADVISES Tool	102
5.1	Introduction	102
5.2	Workflow	103

5.3	ADVISES Tool GUI	105
5.3.1	ADVISES Tool for Static Verification	106
5.3.2	ADVISES Tool for Dynamic Verification	111
5.4	Metrics computation	115
6	Case Studies	118
6.1	Case study 1 (what-if analysis): A Wireless Body Sensor Network	118
6.1.1	Scenario	118
6.1.2	Results	119
6.2	Case study 2 (what-if analysis): A more complex WSN	122
6.2.1	Scenario	122
6.2.2	Results	123
6.3	Case study 3 (robustness checking): Is it worth to add a node?	125
6.3.1	Scenario	125
6.3.2	Results	126
6.4	Case study 4 (robustness checking): Checking robustness in harsh environments	127
6.4.1	Scenario	127
6.4.2	Results	128
6.5	Case study 5 (runtime verification): WSN robustness checking at runtime .	130
6.5.1	Scenario	130
6.5.2	Results	133
	Conclusions	138
	Bibliography	145

List of Tables

1.1	Failure Mode and Effect Analysis of a monitoring system	31
2.1	Fault Injection Tools	35
2.2	Approach Classification	55
3.1	Basic elements of the specification for the isolation event	67
3.2	Basic elements of the specification for the packet loss event	71
3.3	Basic elements of the specification for the battery exhaustion event	73
4.1	Basic events detected by the system monitors	97
5.1	Tool Inputs	105
6.1	Results of simple linear topology	126

List of Figures

1.1	Wireless Sensor Network	8
1.2	Wireless Body Sensor Network	22
1.3	A Mobile Health Monitoring Architecture	24
1.4	A WSN-based Monitoring Architecture	28
1.5	Components and subcomponents considered in the FMEA	29
2.1	Comparison of Related Work	54
2.2	The workflow for designing specifications for WSN-based applications (4 blocks - 14 tasks).	57
3.1	Workflow for designing specifications for WSN-based applications: General Correctness Specification Modeling	64
3.2	Isolation of a WSN subnet	66
3.3	Example of packet loss	69
3.4	Workflow for designing specifications for WSN-based applications: Structural Correctness Specification Modeling	74
3.5	Example of topology of a WSN	75
3.6	Workflow for designing specifications for WSN-based applications: Verification/Testing	81
3.7	WSN topology of application scenario	82
4.1	Running System with support of the Runtime Checker	92
4.2	WSN-based system with event monitors	94
4.3	Application scenario in dynamic context	100

5.1	Workflow	104
5.2	ADVISES GUI at design time	107
5.3	Channel Model and Initial Battery level frames	109
5.4	Example of an Initial Event Trace specified by the user	109
5.5	Example of a EC file generated by the ADVISES tool	110
5.6	Use of the ADVISES Tool at runtime	112
5.7	Flow chart of the ADVISES operating at runtime	113
5.8	ADVISES GUI at runtime	114
6.1	WBSN case study	119
6.2	Topology loading with ADVISES tool	120
6.3	Topology loading with ADVISES tool	120
6.4	Topology of a more complex WSN	122
6.5	WSN with line topology	126
6.6	WSN with extra node in line topology	126
6.7	WSN topology with 8 nodes	128
6.8	Results of the case study	129
6.9	WSN topology in dynamic scenario	131
6.10	Scenario with <i>Stop(5)</i> event	131
6.11	Scenario with <i>Stop(7)</i> event	132
6.12	Java-based emulator	133
6.13	ADVISES GUI in dynamic verification	134
6.14	ADVISES GUI receives first event from the WSN	135
6.15	2-level forecasting with first detected event	135
6.16	3-level forecasting with first detected event	135
6.17	ADVISES GUI receives second event from the WSN	136

List of Listing

2.1	Example of Event Calculus <i>narrative</i>	47
3.1	Correctness Specification for the Isolation event	67
3.2	Correctness Specifications for the packet loss event	69
3.3	Correctness Specification for the battery exhaustion event	73
3.4	Example of Neighbor predicate Specification	75
3.5	Example of initial event trace	77
3.6	Algorithm for the computation of event sequences with n failures	79
3.7	Structural specification written in Event Calculus language for WSN-based application scenario	84
3.8	Outcome produced by the DECReasoner for WSN-based application scenario	86
6.1	Initial event trace for a WBSN	120
6.2	Parameters for a WBSN	121
6.3	Outcome of the DECReasoner for a WBSN	121
6.4	Initial event trace for a more complex WSN	123
6.5	Parameters for a more complex WSN	124
6.6	Outcome of the DECReasoner for a more complex WSN	124

Acknowledgements

“Il raggiungimento di un primo traguardo così importante, rappresenta il punto di arrivo di un percorso maturato tre anni...”

(tratta dalla tesi laurea triennale - Marzo 2006)

“Questo ulteriore lavoro di tesi testimonia in realtà che nel 2006 non fu solo un punto di arrivo (momentaneo), bensí un nuovo punto di partenza in salita andando incontro a nuove soddisfazioni.”

(tratto dalla tesi laurea specialistica - Settembre 2008)

E già! Chi l'avrebbe mai detto che anche il gradino della laurea specialistica sarebbe diventato a sua volta un nuovo punto di partenza?!!

Concluso quest'altro step che chiude definitivamente il ciclo di studi accademico mi verrebbe da dire, citando una delle espressioni piú celebri del Carosello degli anni '60: *“E mo', e mo', e mo'? Moplen!”*

Ora, trascorsi questi tre anni di training, di nuove esperienze, di ricerca scientifica, di scrittura di articoli scientifici, l'obiettivo é investire sul lavoro svolto mettendo in pratica la maturitá, l'esperienza acquisita e (perché no?!) un pizzico di fortuna; come diceva il buon Cicerone: *“Audaces fortuna juvat”* (da *Tusculane*).

Fare i ringraziamenti non é mai una cosa semplice, non tanto per i contenuti ma piuttosto per l'attenzione a non dimenticare nessuno e garantire la par condicio. Quindi, dal momento che ho il desiderio di ringraziare chiunque mi sia stato accanto in questi tre anni, mi scuso a priori se qualche nome non mi verrà in mente ma ciò non vuol dire che non lo porti con me.

Premesso ciò, i miei primi ringraziamenti vanno a coloro che hanno reso possibile questo lavoro di tesi e cioè al mio tutor accademico Marcello e al mio tutor aziendale Antonio. Entrambi si sono distinti per la loro esperienza e capacità nel seguirmi, nel darmi suggerimenti, nel rendermi piú critico di fronte ai complessi problemi della ricerca. Non posso non ringraziare anche i *boss* delle relative parti: Domenico e il Prof. Russo, e Pino che mi hanno

dato la preziosa possibilità di svolgere l'attività di ricerca presso l'*Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)* del *Consiglio Nazionale delle Ricerche (CNR)* di Napoli.

Ringrazio la mia famiglia e soprattutto mia mamma e mio padre che mi sono continuamente vicini (chi fisicamente, chi spiritualmente). Anche se parte della famiglia sin dall'inizio (o forse ancora ora?) si è mostrata scettica riguardo la scelta fatta tre anni fa, come dico sempre, nella vita bisogna avere sempre fede e credere in ciò che si fa portando avanti sempre degli obiettivi prefissati! E anche nel dottorato bisogna avere sempre fede e crederci sino in fondo!

Un ringraziamento speciale va ad una persona speciale e dolcissima: Cristina. Gli anni della mia attività di ricerca trascorsi sono pari agli anni in cui mi sta accompagnando in un nuovo e stupendo viaggio fatto non più da una sola persona ma da due. Mi ha sempre spinto, spronato ed è riuscita a sostenermi anche nei momenti più difficili e durante il periodo di permanenza a Londra. Ti amo!

Ringrazio il collegio dei docenti della Scuola di Dottorato per la professionalità e l'attenzione costante che hanno mostrato proponendo nuovi ed interessanti corsi, seminari e scuole di dottorato; e quindi non posso non ringraziare il coordinatore, Prof. Franco Garofalo, per il suo ruolo da guida svolto con il massimo impegno attraverso i continui aggiornamenti e la preziosa attenzione ai vari impegni che ciascun dottorando ha dovuto adempiere.

Uno *special thanks* al Dr. Juan Carlos Augusto il quale mi ha dato la possibilità di trascorrere parte del dottorato presso la *School of Science and Technology* della *Middlesex University of London*; la sua fiducia, il suo interesse per l'attività di ricerca svolta e le sue lezioni-lampo d'inglese mi hanno dato ulteriore supporto. E qui colgo l'occasione per ricordare i ragazzi del lab, in cui sono stato ospitato, i quali hanno assistito ad un'evoluzione parziale del presente lavoro: Arni, Joshua, Krishna, Friedrich, Yetish, Ryan, Ali, Rand, Hanna, Payam, Mirco, Pietro, Franco e Leonardo. E ringrazio Mario P., un compagno delle elementari ritrovato (dopo ben 18 anni!) a Londra che mi ha aiutato ad affrontare psicologicamente una metropoli come Londra *by day and by night*.

Un grazie a tutta la famiglia "allargata": cognato, suoceri, zii, zie, cugini, cugine, ecc. In questi tre anni posso dire che si è estesa ed ognuno di voi con il suo gesto (diretto o indiretto) si è reso complice di questo traguardo.

Ringrazio tutta la banda MobiLab (Flavio, Lelio, Enzo) e CINI (Antonio P., Roberto N. e P., Anna, Francesca e Stefania) ma in particolare i miei compagni di squadra del dottorato: Domenico, Antonio e Fabio; con loro ho condiviso diversi momenti (corsi, seminari, scuole di dottorato, le presentazioni di fine anno e la fase finale di questo percorso).

Come accennavo, il dottorato è stato frutto di un lavoro svolto presso l'ICAR; dunque mi sembra doveroso porgere un ringraziamento anche a tutta la squadra ICAR e in particolare a Nello, Mario C., Luigi, Giovanna, Nico, Massimo, Mario S., Christian, tutti i ragazzi del

lab e tutti gli altri ricercatori, tecnici e componenti dell'amministrazione.

Sebbene l'impegno di ricerca mi ha portato via molto tempo, costringendomi a non frequentare con la stessa assiduità di una volta i miei vecchi e nuovi amici, voglio tuttavia menzionare alcuni di loro che porto sempre con me cercando di mantenere quel legame che ci unisce da una vita: Enzo, Paola, Fabrizio, Francesco L.S., Luca, Antonio Capi, Ilaria, Ale Fox e tutta la combriccola puteolana, Fiorella, Diego, Licia, Salvo, Sasi, Ale, Rosario, Valentina, Enzo, Monica, Mario, Nello, Tony, ed i miei compagni di studio e non solo, Barbara, Alfonso e Fabio.

Infine *last but not least*, come oramai mio solito in ogni ringraziamento, voglio fare uno speciale ringraziamento a Colui che ha reso possibile tutto ciò sin dall'inizio; certo, se sono qui è per l'amore dei miei genitori, ma è soprattutto perché l'ha voluto il Signore che mi ha donato tutto il suo Amore e mi custodisce giorno dopo giorno. E quindi, tutto sommato....se sono riuscito ad arrivare fin qui lo devo anche a Lui. Grazie!

Napoli, Italy
March 30, 2013

Alessandro

Introduction

Wireless Sensor Networks (WSNs) [30] are being more and more used into critical application scenarios where the level of trust on WSNs becomes an important factor, affecting the success of large-scale industrial WSN applications; the extensive use of this kind of networks stresses the need to verify their dependability properties not only at design time to prevent wrong design choices but also at runtime in order to make a WSN more robust against failures that may occur during its operation.

Examples are target tracking, environmental monitoring (e.g. detection of fires in forests [5]), structural monitoring of civil engineering structures [146], health monitoring (in medical scenarios) [67, 114] and patient monitoring [44] by means of Ambient Intelligence (AmI[1]) systems. Depending on application scenarios, different dependability requirements can be defined, such as, node lifetime, network resiliency, and coverage of the monitoring area. The work presented in [36] evidenced that also in a simple deployment, a single node can be responsible of the failure of a huge piece of the network. For instance, a node that is close to the sink (i.e., the gateway of the WSN that has the role to collect all of the measures detected by the sensors) is more likely to fail due to the great solicitations it is subjected to, and its failure would likely cause the isolation of a set of nodes from the sink. Moreover, the correct operation of a WSN can be affected by other several undesired events, such as the crash of a node due to the cheap hardware adopted [136], software errors [41], battery exhaustion, and the unreliability of the wireless medium. Failure occurrences are strongly exacerbated when nodes are deployed in harsh environment [117] and these, in turn, may

isolate whole portions of the network or cause packets to be lost during their traversal to the destination.

If not adequately considered, these events may cause severe failures with dangerous consequences, such as, a health monitoring system not able to report critical alerts about a patient status to a medical center, or a structural monitoring system unable to report a developing crack in a structure.

Therefore it is necessary to verify the WSNs at design time in order to increase the confidence about the robustness of the designed solution before putting it into operation. It is also necessary to monitor the system during operation in order to avoid unexpected results or dangerous effects. Hence, it is important to verify such networks at runtime to perform what in the literature is defined *continuous monitoring* [81].

Formal methods are widely adopted in the literature to verify the correctness of a system specification. They are a particular kind of mathematically based techniques for the specification, development and verification of software and hardware systems [70] and allow to define specifications using a language very close to humans.

However, their practical use for the verification of dependability properties of WSNs has received little attention, due to the distance between system engineers and formal methods experts and the need to re-adapt the formal specification to different design choices. Even if some development teams would invest on the definition of a detailed specification of WSN correctness properties, a design change (e.g., different network topology, number of produced packets) could require to rethink the formal specification, incurring in extra undesirable costs. Nevertheless, it could be very useful to have an unique set of formal specifications that are able to check the WSN behavior at design time and runtime.

To address these issues, the profuse effort in this dissertation deals with the definition of formal specifications used for the behavioral checking of WSN based systems in static and

runtime phases; a set of correctness specifications applied to a generic WSN has been defined using a formal language. Since the behavior of a WSN can be characterized in terms of an event flow (e.g. a node turns on, a packet is sent, a node stops due to failure, etc.) we decided to adopt the *Event Calculus* formalism [124] that allows to easily specify the system in terms of events. This formal language is *event-based* and *narrative-based*, i.e. it allows to observe the sequence of generated events, given a received event. The narrative is very useful to analyze the behavior of the network and to indirectly evaluate the metrics of interest, such as coverage, resiliency, and lifetime. It is considered the *DECReasoner* [104] as an Event Calculus reasoner to produce the event traces.

In particular, the main contributions of this dissertation are:

- the definition of the formal specification of WSN correctness as two logical sets: a *general correctness specification*, valid independently of the particular WSN under study, and a *structural specification* related to the properties of the target WSN (e.g., number of nodes, topology, channel quality, initial battery charge), designed to be generated automatically; in this way, it is no necessary rewrite all the specifications that are common for every WSN, applying a modular solution;
- the use of specific WSN dependability metrics, such as *connection resiliency*, *coverage*, *data delivery resiliency* and *lifetime*, as drivers to guide design choices;
- the realization of two types of verification techniques exploiting the same set of formal specifications: *static verification* and *dynamic verification*. The former analyzes a WSN during the design phase evaluating the robustness against a sequence of failures that can occur; instead the latter, by means of a runtime verification techniques, consists in observing the WSN behavior during its operation and detecting, with a set of monitors, the occurring of failures in order to provide information about critical nodes and to perform forecasting at runtime about next criticalities.

- the development of an automated verification tool, named *ADVISES* (**A**utomated **D** **V**erification of **w**S**n** with **E**vent calculu**S**), to simplify the adoption of the proposed approach.

It is conceived: i) to operate in double mode: static and dynamic; ii) to automatically generate the structural specifications given the properties of a target WSN (e.g. topology); iii) to perform the reasoning starting from the correctness and structural specifications; iv) to compute dependability metrics starting from the event trace produced by the reasoner; and v) to receive events in real-time from a WSN to start runtime verification and to evaluate current and future criticalities;

- the presentation of the usefulness of the approach in the context of case studies, to show how the proposed framework and tool can help system engineers to take decisions upon key design and runtime questions such as: “How many nodes are covered in the WSN if a given sequence of failures occurs?”, “How many failures the WSN is able to tolerate so that a minimum coverage level is guaranteed?”, “How the WSN behaves, in terms of delivered packets, if the channel quality changes (e.g., to consider environments with different levels of harshness)?”, “What are the current critical nodes that can compromise the robustness of the network?”.

The ADVISES tool has been designed in order to provide useful support for answering questions like the ones above.

In static verification, this tool is used to perform two different types of analysis (starting from the same specification):

1. *what-if analysis*, to verify how the WSN behaves in response to a given sequence of events of interest for the designer;
2. *robustness checking*, to verify the long term robustness of the WSN against random

sequences of undesired events, useful to identify corner cases and dependability bottlenecks.

In runtime verification, the tool operates as a server that is in waiting for new events coming from the WSN; it autonomously returns informational messages to the user. For example, if a node fails in a WSN while it is running, this failure is detected by a monitor which sends the event to the tool. The tool receives the event and, by means of a reasoning, calculates the dependability metrics both for the current status of the WSN (e.g. raising an alarm if a given criticality level is reached) and to assess the criticality of the network, in order to alert the user about possible future hazardous scenarios considering the new network conditions.

The rest of the dissertation is organized as follows.

Chapter 1 provides the needed background on WSNs and on the dependability threats of these networks, then it describes the critical WSN-based applications presenting a failure modes and effects analysis of a generic WSN-based monitoring system. Open issues are considered to motivate the contribution of the dissertation.

A description of the main approaches used to assess the dependability in WSNs is given in chapter 2. The chapter also proposes a comparison of the studied work under most important chosen parameters.

Chapter 3 is dedicated to the static verification study of WSNs. After giving the definitions of general correctness specifications for a WSN to detect some failures that can occur and the definition of structural specification of WSNs, the chapter presents the types of analysis used for the static verification: *what-if analysis* and *robustness checking*.

The dynamic verification technique is discussed in chapter 4. It first provides the needed background on the Runtime Verification, then it discusses the concept of monitor used by the tool to receive critical events. Finally the chapter shows how the tool can be used in a runtime verification context.

The chapter 5 outlines the ADVISES Tool which has been designed and implemented for the automated formal dependability verification of WSN at design time and runtime. The chapter describes the GUI of the tool, the settings, the parameters to choice and all of the offered facilities. Also, it discusses the workflow of the tool that summarizes the functionalities of the ADVISES tool.

In the chapter 6 some case studies are shown. The chapter describes and discusses of the results obtained by some considered scenarios; the objective is to validate the proposed methodology and the strong novelty of the ADVISES tool at design time and runtime.

The dissertation concludes with final remarks and the indication of the lessons learned.

He who loves practice without theory is like the sailor who boards ship without a rudder and compass and never knows where he may cast.

Leonardo da Vinci

Chapter 1

Dependability Issues of Wireless Sensor Networks

In last decade Wireless Sensor Networks (WSNs) have become one of the most discussed topic in several research areas, with increasing interest and strong impact on technological development. WSNs are being more and more used into critical application scenarios, such as environmental monitoring or health monitoring where the dependability still is an important issue. For this reason, dependability assessment is becoming more popular in WSN research activities. This Chapter briefly introduces the WSNs, the dependability threats for this kind of networks, critical WSN-based applications. Finally, open issues are discussed motivating the need of the approach presented in this thesis.

1.1 Wireless Sensor Networks

The main purpose of a Wireless Sensor Network (WSN) [6] as a whole is to serve as an interface to the real world, providing physical information such as temperature, light, radiation, and others, to a computer system.

A WSN is a distributed system consisting of a set of nodes capable of hosting sensors or actuators, perform processing and communicate with each other through multi-hop network protocols by means of that a message reaches its destination after having crossed a number of nodes.

These networks are formed by many elements, called “*sensor nodes*” able to sense physical features of their surroundings or to monitoring a set of items. WSN nodes transmit information on environment in order to have a global view of the monitored items which is made accessible to the external user through one or more gateway node(s), named base station or sink node(s) [96]. Usually a WSN consists of one base station and a high number of wireless sensors (nodes).

An example of WSN is shown in figure 1.1.

Sensor nodes are often considered as smart sensors because of their power, processing and memory capabilities [69, 51, 142, 10]. The small size of sensors (about the size of a coin) allows them to be easily embedded into materials [142] or deployed in a mobile scenarios such as remote health care, cars, or floats over water [50].

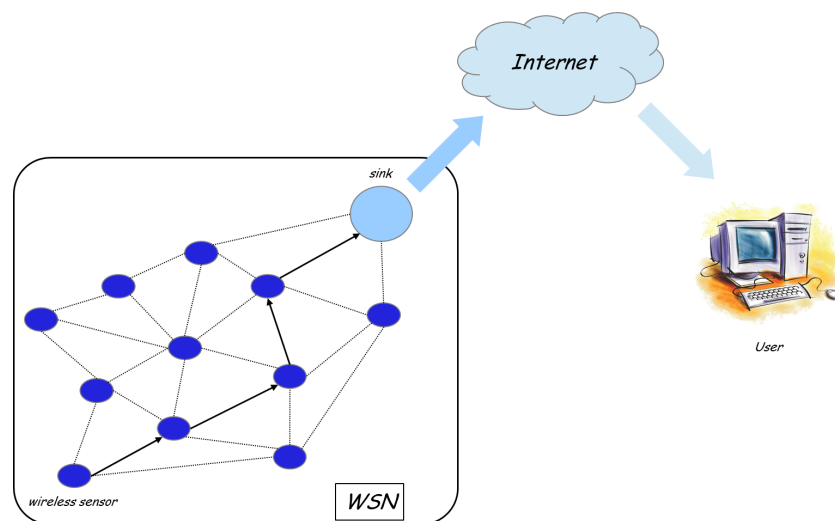


Figure 1.1: Wireless Sensor Network

1.2 Dependability Threats in Wireless Sensor Networks

Currently, the provision of appropriate means to enforce and/or to assess the dependability of WSN-based applications remains an unsolved issue.

The definitions of dependability, its attributes, threats and means, considered until today, have to be reviewed in the new context of WSNs since they introduce novel dependability threats, due to their highly evolvable and dynamic nature. Such new threats call for new means to be adopted to reach an adequate dependability and/or security level.

Simoncini in [129] introduced several drivers for WSN research over the next few years and he suggested some considerations for future work, with respect to the four main typologies of dependability means, i.e., *fault prevention*, *fault tolerance*, *fault removal*, and *fault forecasting* [12].

Fault prevention means are used to prevent the occurrence or introduction of faults. In the context of WSNs, this calls for a formal definition of novel accidental and malicious faults which may be introduced in a WSN-based system.

Fault removal techniques are instead used to reduce the number and severity of faults. To this aim, novel statistical testing and robustness testing techniques should be defined for such evolvable systems.

Fault tolerance means are adopted to avoid failures in the presence of residual faults, which escape the prevention and removal means. In the case of WSN-based systems, it is important to consider that the majority of faults are transient in nature, even if they may lead to catastrophic consequences. In order to define proper counter measures, such faults should

be carefully classified and characterized, based on their causes, triggers, and manifestation.

An analysis on such threats is performed in paragraph 1.4.

Fault forecasting embraces the set of techniques to estimate the present number, the future incidence, and the likely consequences of faults. In the context of WSNs, it could be useful to define specific fault-injection techniques to build dependability benchmarks to compare competing systems and architectures on an equitable basis. A dependability assessment of WSNs at design time could help to increase the confidence about the robustness of the designed solution before putting it into operation and to detect and remove possible critical node.

In addition, specific tools for the on-line monitoring of an WSN-based system need to be defined, in order to statistically characterize the system based on realistic data gathered from the field of operation. The collected data can then be used to improve successive generations of systems and to define proper fault tolerance strategies at runtime.

In [91] the *energy*, the *fault-tolerance* and the *mobility* of the WSNs represent the fundamental issues to be faced for developing the next-generation of smart environments. In particular, authors highlight the problem of the high spatial correlation among the nodes and link failures in these systems, due topological constraints and the common dependency from external events.

The recently proposed prototypes of WSN-based systems present severe technological limitations, which introduce several threats to the dependability of the overall system. Being the system usually composed by low-cost, battery-powered and wireless-enabled devices,

their correct functioning is affected by the capacity of batteries, the unreliability and variability of wireless communications, the mobility of users, the risk of physical damage, etc. In addition, the presence of hardware faults, due to defects and interference, and the presence of residual software bugs further complicate the picture.

These threats are in part discussed in [79], where authors discuss about the limited device lifetime and communication due to low electrical energy and the poor physical protection of mobile devices which can make them prone to physical damage, if deployed in a harsh environment. Therefore authors raise the issue of realizing a fault- and intrusion-tolerant collaborative data backup for WSN-based systems. They consider a scenario where it is needed to protect the backed-up data against denial-of-service attacks and availability problems due to failures.

In [8] technologies based on WSN are considered as one of the key research areas to favor the development of future healthcare industries. The authors illustrate the state of the art on WSN-based systems for healthcare with related benefits, issues and challenges. It is interesting to note that several challenges and open research problems for Ambient Intelligence (AmI) Systems are indicated, in particular for dependability problems. In this work it is reported an analysis of open issues divided by layers. The majority of dependability issues result to be concentrated in the physical layer, where the scarce transmission power and the small size of antennas particularly compromise the resilience of the communication. In particular, the reduced Signal-to-Noise Ratio (SNR) causes high bit error rates and reduces the capacity of the network to reliably cover the area of interest. Similar dependability

issues are discussed for the transport layer. The scarce computation power of sensor devices does not allow to implement complex, reliable transport protocols, with control flow and retransmission. On the other hand, the loss of even a single packet may result in a significant hazard. This is a major impairment to medical monitoring systems, dealing with life-critical data. In such systems, the resilient delivery of medical data could result vital for monitored patients.

Thus, the research is progressively recognizing the need of novel solutions to build dependable WSNs. These solutions mainly focus on two key issues: node failures and wireless network interference.

Regarding node failures, WSNs may suffer from intentional or unintentional node removal or unresponsive nodes. A combination of node redundancy and multi-sensor data fusion was one of the solutions proposed to face these issues [15, 52]. The introduction of redundant sensors avoids the loss of any data if a node becomes compromised or faulty. In addition, they can serve to facilitate multiple paths when the routing becomes an issue.

Interference has the potential to cause significant delays and data loss and is a major concern with all wireless devices. Motes can interfere with each other in the WSN as well as being subject to environmental noise. This is due to the lack of harmonious regulations and standards, as demonstrated in [65, 132]. A solution would be to eliminate the wireless aspect of the wireless network [72, 66]. Sensor Network based systems such as MITHrill [72], SMART [72], and MobiHealth [141] all employ a wired connectivity between sensors and the aggregate. However, these solutions strongly limit the usability of the system, especially for elderly people, and makes it hard to interconnect all the sensors to commodity

mobile devices, such as patients' smartphones, hence requiring ad-hoc aggregating devices which increase the overall cost of the system.

Currently, in literature several fault-tolerant techniques for WSN have been proposed. However, there still is an open issue about the validation method of these techniques both at design and runtime. Hence, one of the aims of this dissertation is also to answer some questions such as “*Given a fault-tolerant technique, am I able to obtain a certain coverage value?*”, “*Is the chosen fault-tolerant technique it convenient? Is it an expensive or cheap solution?*”; an overview of the methods and techniques for the WSN assessment, discussed in Chapter 2, will help to answer to the raised questions.

1.3 Critical WSN-based Applications

Nowadays the field of WSNs offers a multi-disciplinary and rich area of research, in which several tools and concepts can be employed to address particular kinds of applications. As such, many potentials of this field have been under study both in academia and in the industry. Only recently they have become a technology more envisioned in real applications, included industrial systems or critical scenarios as a good opportunity to drastically reduce installation, management, and maintenance costs and related times.

According to the European Commission, Critical scenarios consist of “*[...] those physical and information technology facilities, networks, services and assets which, if disrupted or destroyed, would have a serious impact on the health, safety, security or economic well-being of citizens. [...]*” [111].

This section focuses on five types of critical WSN-based monitoring systems: environment monitoring, security monitoring, object tracking, ambient intelligence and health monitoring. On the basis of the context, WSNs may be constituted by of a number of nodes that can be low in case of health monitoring or high in case of security monitoring or environmental monitoring. Nodes may be distributed on a large region or be inserted in a small area with high density.

For each application scenario, we discuss if the following WSN requirements are met:

- **Coverage**

Coverage represents the percentage value of the number of nodes that are running and connected and reachable by the sink node at a given time. To assure the monitoring of the phenomena keeping an high degree of confidence, WSN applications set a minimal value of coverage to guarantee.

- **Lifetime**

Typically in almost all critical WSN-based applications the nodes are self-powered since they are deployed in harsh environments, and hence, they can be active and operating for a finite time. The lifetime metric adopted for these scenarios is defined as the time until a given percentage of nodes in the WSN goes below a given threshold [27].

- **Data delivery resiliency**

A common WSN is realized to collect data that contain measurements and send them to the sink node. On the basis of a specific application, it is necessary to collect a minimal amount of data in order to obtain a correct monitoring (e.g. in case of health monitoring a WSN equipped with ECG has to send to the monitoring application all the necessary data thus to have a correct ECG signal).

However, interferences and other types of failures may make necessary a reconfiguration of the WSN topology that may have effects on the data delivery efficiency.

- **Timeliness**

In WSN-based monitoring applications the several data, coming from the sensors of a WSN, have to be correlated in order to obtain desired measurements. In WSNs, by means of synchronization protocols, sensors exchange periodically extra radio packets to perform continually high-precision synchronization mechanisms that compensate inaccuracies caused by the oscillators of the clocks.

1.3.1 WSN Dependability Metrics

We define following dependability metrics, already adopted in [55, 137], in order to verify the mapping with the WSN requirements (excluding timeliness):

- *Coverage* is the time interval in which the WSN can operate, while preserving a given number of nodes connected to the sink.

- *Connection Resiliency* represents the number of node failures and disconnection events that can be sustained while preserving a given number of nodes connected to the sink.
- *Data Delivery Resiliency* is the number of node failures and disconnection events that can be sustained while preserving a given number of correct packets delivered to the sink.
- *Power consumption* is the battery power consumed by each sensor, useful to estimate the expected lifetime of the WSN.

Overall these metrics allow to evaluate the expected dependability of the WSN in terms of its robustness to failure events, its capacity to cover a given area, and its duration (lifetime). Clearly, other metrics can be defined for other purposes.

1.3.2 Environmental Monitoring

To periodically measure meteorological and hydrological parameters, such as temperature, sound, vibration, pressure, motion of the earth, etc., environment monitoring [128, 56] is adopted exploiting a number of nodes equipped with sensors to have data of physical phenomena.

Dangerous phenomena, such as fires in forests [61, 40], can occur and it would be difficult to forecast; thus the role of the wireless sensors is to detect them and guarantee a dependable delivery of collected data to the sink node. Hence, lifetime and data delivery resiliency represent the most important requirement for this type of monitoring. Since packet losses are frequent it becomes hard to have a good level of resiliency because of harsh phenomena

(e.g., interference, heavy rain, intense cold) which are at the same time the most interesting episodes for data analysis. Moreover, since WSNs are characterized by a multi-hop organization, typically disconnections of nodes to the sink occur causing a partition of the network.

Usually wireless sensors used in environmental monitoring are placed in harsh environments [20]. An example of environmental monitoring is discussed in [144]. In SensorScope project wireless sensor network is deployed in remote and difficult-to-access places and for this reason it was necessary a helicopter for carrying hardware and people.

1.3.3 Security Monitoring

WSN-based security monitoring applications [25] do not collect any data; the common task of each node is to frequently check the status of its sensors, and to transmit a data report strictly only when an exception is detected (*report by exception*), such as security breaches or unauthorized access to an environment. Being important to guarantee security [123], nodes are equipped with backup power sources to address the problem of battery exhaustion. For this reason lifetime in these nodes is no critical as for other monitoring systems. The most important requirements for this type of monitoring is the data delivery resiliency since for security reasons data have to be delivered without errors.

1.3.4 Ambient Intelligence

Ambient Intelligence (AmI) is the emerging computing paradigm used to build next generation smart environments. It is claimed to provide services in a flexible, transparent, and anticipative manner, requiring minimal skills for human-computer interaction. Recently, AmI is being adopted also to build smart systems to guide human activities in critical domains, such as, healthcare, ambient assisted living, and disaster recovery.

AmI is a term coined by Philips management to conjure up a vision of an imminent future in which persons are surrounded by a multitude of fine grained distributed networks comprising sensors, and computational devices that are unobtrusively embedded in everyday objects such as furniture, clothes, and vehicles, and that together create electronic habitats that are sensitive, adaptive and responsive to the presence of people [21, 86].

Such systems are based on a large number of heterogeneous devices, from handheld and wearable devices operated by users to smart sensing and actuating devices embedded in the surrounding environment, able to interact each other spontaneously by exploiting different communication links. In this context, applications have to make effective use of the resources available on-the-fly, and adapt to different hardware and software, and even firmware configurations.

Examples of AmI applications are smart offices and buildings [100]. In a smart offices, every movement of the employees is recorded as well as their location during a certain temporal interval and the temperature of the rooms in which the employees work.

A survey of the technologies of ambient intelligence systems [43] shows several challenges

and opportunities that AmI researchers will face in the coming years. The authors organize the contributing technologies into five areas: *Sense*, *Reason*, *Act*, *Human-Computer Interface* and *Secure*. In the last area, *Secure*, they highlight some dependability issues; for example, at the sensor level, the sensor reliability, the error handling process, and the errors due to misconfiguration can create security vulnerabilities. To ensure security in sensor networks, the designer must consider these factors together with sensor communication channel reliability/availability and sensor data integrity and confidentiality. An ongoing challenge for AmI researchers is mentioned about the design of self-testing and self-repairing AmI software that can offer quantitative quality-of-service guarantees and a high degree of dependability.

Currently, in the literature, there is still a lack of a commonly accepted architecture to build the AmI systems of the future, with predictable dependability properties. This issue is considered in [129], where the author pointed out the concepts of “architecture” and “system” need to be redefined in the context of AmI systems, in order to properly define “ambient dependability” attributes, threats and means.

In [17], authors define a dynamic system able to adapt itself to the current situation. They claim that, in order to guarantee dependability requirements, the system architecture has to be manageable, controllable and it has to provide means for the prediction of the system correctness at runtime. In [108], authors proposed an integrated system approach for living assistance systems based on ambient intelligence technology. They claim that the construction of trustworthy, robust, and dependable living assistance systems is a challenging task which requires novel software engineering methods and dependability assessment tools (to

validate engineering choices), and novel approaches for dependable self-adapting software architectures, able to react to changes due to frequent failures and reconfiguration events, which become the norm, rather than the exception.

Georgalis et al. argue that the most important architectural property in an AmI architecture is the fault-tolerance [63]. The fault tolerance, in the context of an AmI architecture, has to be able to isolate failures, to eliminate single points of failure, to restart failing services before that are used by the clients, and finally to provide mechanisms for notifying the fault level about the irreparable failure of a specific service.

Coronato and De Pietro [44] pointed out that the design of AmI applications in critical systems requires rigorous software-engineering-oriented approaches. The authors proposed a set of formal tools and a specification process for AmI, which have been devised to lead the developer in designing activities and realizing software artefacts.

Zamora-Izquierdo et al. propose a platform [149] that provides a home automation architecture, called DOMESTIC, able to satisfy current and future needs in indoor environments. Although the architecture is claimed to be robust and dependable, there is not evidence nor experimental study assessing the dependability level.

Hence, having analyzed previous work, we can assert that stringent dependability requirements are called for the practical application to such domains, since even if a single component fails then there may be dangerous loss or hazard to people and machineries. Moreover, because of mobility of resources [4, 24] and the peculiarities of domains [17], AmI applications can be affected by new threats. Dependability requirements in AmI applications to consider are data delivery, timeliness and lifetime.

Finally, the attention to dependability issues in AmI systems is also witnessed by recent European Union initiatives, such as the *SERENITY (System Engineering for Security and Dependability) Project* [95] that aims to provide security and dependability in AmI systems.

1.3.5 Health Monitoring Systems

Health monitoring systems have been shown to be effective in helping to manage chronic disease, post-acute care, and monitoring the safety of the older adult population [49]. They can help older adults slow progression of chronic disease and ensure continued recovery after being discharged from an acute care setting. The implementation of such systems is gaining an increasing attention in the academia and the industry, also due to the increasing healthcare costs and the aging of the world population [67].

To this purpose, cabled measurement equipment is already used to guarantee reliable and robust control of vital signs. However such systems complicate patient autonomy and mobility. Hence, wireless technologies and mobile devices are starting to be applied to build more comfortable and patient-friendly health monitoring systems [114].

An health monitoring system is based on Wireless Body Sensor Networks (WBSNs) [110]. The term *Wireless Body Sensor Network* was coined for the first time in [147] in order to define a wireless network of wearable computing devices. A set of physiological sensors can be integrated into a wearable wireless body area network that can be involved for computer-assisted rehabilitation or early detection of medical conditions.

The basic structure of WBSN consists on a set of wireless physiological sensors, such as body

temperature, oximeter, blood pressure and ECG. By means of these sensors, we are able to monitor remotely a patient (*Home Monitoring*). It is also possible to use WBSN in helping assisted-living (see Ambient Assisted Living Applications [45, 108]) and independent-living residents by continuously and unobtrusively monitoring health-related factors such as their heart-rate, heart-rhythm, and temperature.

In a WBSN time latency has to be limited and the transmission of the vital signs has to be dependable. Moreover, coverage is a dependability metric to take in account since WBSNs are constituted by a very limited number of nodes that can be vulnerable to some failure (due to crash, interference, etc.). Thus, it is opportune to provide WBSNs of a system that, by means of messages, alerts caregivers.

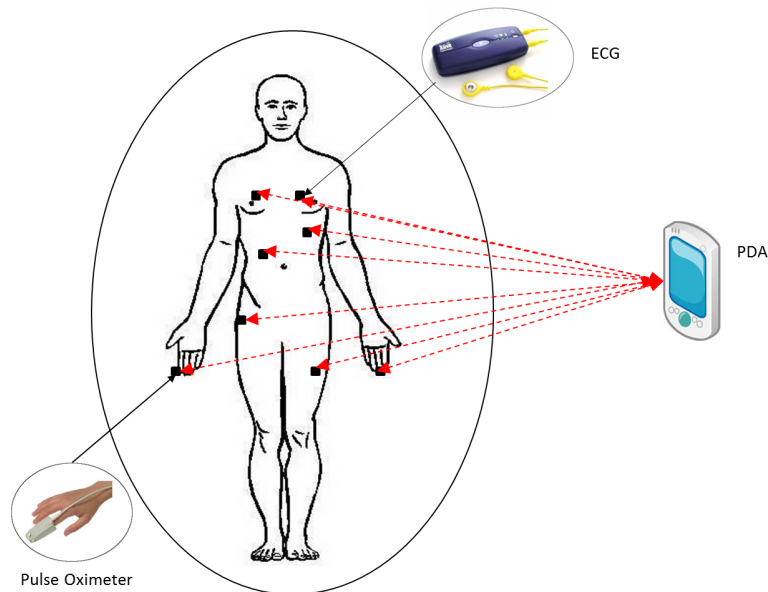


Figure 1.2: Wireless Body Sensor Network

Figure 1.2 shows a patient with a number of wireless sensors attached to the body (e.g. a pulse oximeter, an ECG,...), each of which is connected to a small processor, an antenna

and a wireless battery, and all together form a WBSN.

Nevertheless, the use of wireless technologies and the adoption of commodity hardware/-software platforms, such as smartphones, pose new challenges on the correct functioning of health monitoring systems. Wireless channels can be affected by packet loss, due to shadowing and absence of signal coverage. Smartphones can be subjected to unpredictable failures, which could affect the correct functioning of the system. Finally, cheap and wireless-enabled medical devices can exhibit wrong readings and temporary disconnections.

These issues may induce the medical staff to take wrong decisions, e.g., to administer wrong dosages of medicine, which can happen to be fatal for the patient.

For these reasons, the problem of failure detection and management in health monitoring systems is starting to be addressed in the literature, especially for mobile systems. However, several studies are based on simplistic failure assumptions or on basic fault tolerance schemes (such as, sensor redundancy), which are not assured to cover all possible failure scenarios. For instance, sensor replication is ineffective against smartphone failures.

To overcome the limitations of current solutions, several reliable mobile health monitoring systems have been proposed and designed in the last years.

Among different implementations, we chose three of the most popular: the *MedApps System*, the *Nicolet Ambulatory Monitor System* and a system used by the *Center for Technology and Aging*.

The *MedApps System* [57] provides a healthcare connectivity platform that delivers scalable and flexible remote distribution using cellular, wireless and wired technologies with cloud-based computing. This system can work with multiple internal and external devices.

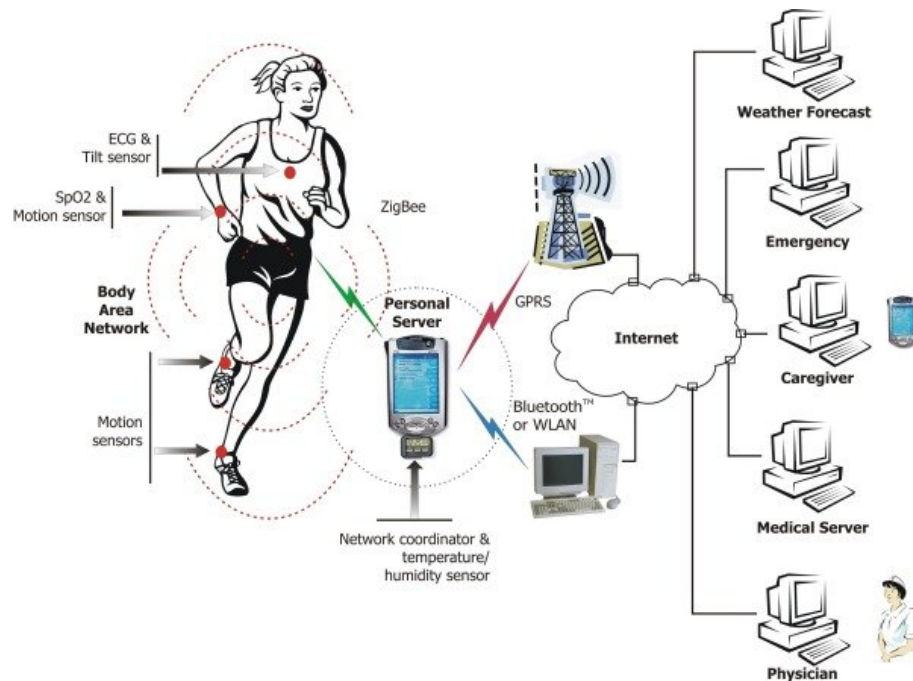


Figure 1.3: A Mobile Health Monitoring Architecture

Patient data is collected, analyzed and forwarded, via cell phone to servers, guaranteeing a more robust picture of the patients' health.

The *Nicolet Ambulatory Monitor System* [145] combines a flexible, high quality diagnostic unit, ideal for patients of all ages. It is a flexible, robust system specifically realized to provide the requirements of long-term monitoring. This system diagnoses patients' cerebral function (premature neonates to older adults) monitoring continuously ill patients at risk for brain damage and secondary injury.

Finally, authors in [62] discuss two areas of opportunity for remote patient monitoring: i) Patient Safety and ii) Chronic Disease Management and Post-Acute Care Management. In alignment with the mission of the Center for Technology and Aging, they focus on technology-enabled innovations, such as wireless connectivity, mainly aimed at improving

the health of older adults and promoting independent living in community-based, home, and long-term care settings.

A typical mobile health monitoring system is illustrated in figure 1.3.

In the health monitoring systems the transmission of vital sign data and the alarms have to be dependable in order to intervene as soon as possible and the time latency has to be limited and predictable. Moreover, coverage and lifetime of network are important, since WBSNs are composed by few sensor nodes each of which is devoted to monitor a physiological sign.

1.4 FMEA of a WSN-based Monitoring System

To fill the gap in the knowledge about the possible threats that may affect the correct functioning of a monitoring systems, in this section we present the results of a *Failure Modes and Effects Analysis (FMEA)* conducted to identify the failure modes of the main components composing such systems and which have been already published in [35, 32, 34] and here are briefly summarized.

The analysis takes advantage of past experience and detailed field studies on the dependability of mobile devices, wireless communication technologies, such as Bluetooth, and wireless sensor networks (WSNs), and builds on such results to propose a comprehensive characterization of the problems that may affect modern monitoring systems. In [35, 34], the analysis has been realized considering a mobile health monitoring system (as represented in figure 1.3) and it is based both on previous studies on different system components (such as WSNs,

smart phones, and short range communication technologies) and on FMEA results available on some subcomponents, such as devices (i.e. medical devices, environmental sensors, ...). Firstly some FMEA fundamentals are introduced to better know this kind of analysis and secondly a FMEA table, produced analyzing a typical monitoring system, is presented as a guidance tool to direct future research efforts towards the realization of more dependable monitoring systems.

FMEA fundamentals

FMEA is a team-based, systematic and proactive approach for identifying the ways that a process or design can fail, why it might fail, and how it can be made safer [84]. To properly evaluate a process or product for strengths, weaknesses, potential problem areas or failure modes, and to prevent problems before they occur, a FMEA can be conducted. The purpose of performing an FMEA, as described in US MIL STD 1629 [3], is to identify where and when possible system failures could occur and to prevent those problems before they happen. It represents a procedure for analysis of potential failure modes within a system for classification by the severity and likelihood of the failures.

An FMEA provides a systematic method of resolving the questions: *How can a process or product fail? What will be the effect on the rest of the system if such failure occurs? What action is necessary to prevent the failure?*

To realize a FMEA, the system is divided in components/functions that are divided in subcomponents/subfunctions; it considers a table in which the rows are composed by the

subcomponents/subfunctions and the columns represent respectively the failure modes, the possible causes and the possible effects. If a particular failure could not be prevented, then the goal would be to prevent the issue from affecting users of the teams in the accreditation process. The FMEA team determines the effect of each failure by failure mode analysis and identifies single failure points that are critical.

It may also classify each failure according to the criticality of a failure effect (*severity*) and its probability of occurring (*probability*). There are some motivations why this analysis technique is very advantageous. FMEA provides a basis for identifying root failure causes and developing effective corrective actions; the FMEA identifies reliability and safety critical components; it facilitates investigation of design alternatives at all phases of design; it is used to provide other maintainability, safety, testability, and logistics analyses. FMEA is thus part of a larger system of quality control, where documentation is vital to implementation. In our case, FMEA is useful to detect the main dependability threats which have to be taken in account to perform dependability assessment of a WSN.

Since FMEA is effectively dependent on the members of the team which examines the failures, it is limited by their experience of previous failures. If a failure mode cannot be identified, then external help is needed from consultants who are aware of the many different types of product failure.

FMEA results

In this paragraph the results of the FMEA performed on generic monitoring systems are presented.

Let us consider a monitoring system (figure 1.4) composed by a number of sensors, a gateway device (a handheld device) and a remote station; typical communication means are bluetooth, ZigBee (within the WSN - Intra WSN communication), WiFi and cellular (external to the WSN - Extra WSN communication). Data are sensed by sensors (i.e. accelerometer, light sensor, etc.) and transmitted to a mobile device over a bluetooth network. Afterwards, data are sent to a remote station deployed, for an example, in an office by means of either a WiFi or a cellular connection (the remote center location).

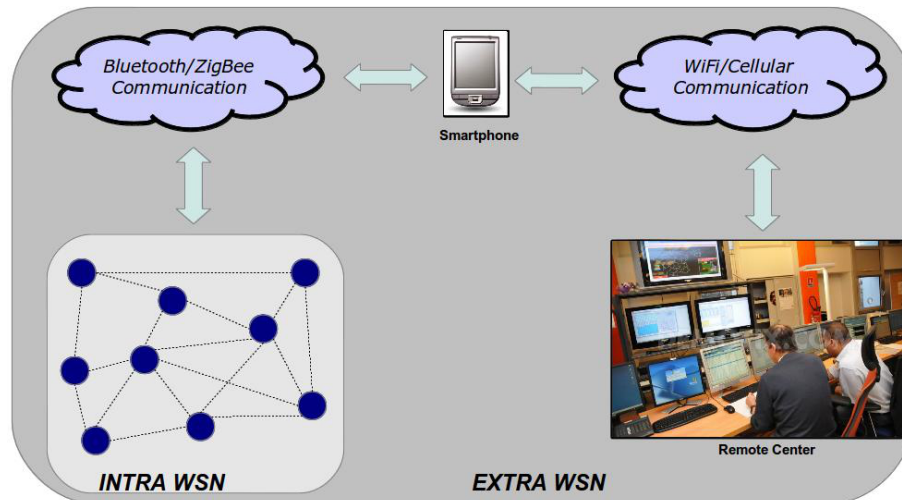


Figure 1.4: A WSN-based Monitoring Architecture

In this typical network, possible failures can occur in sensors, in the bluetooth communication, in the mobile device, during the WiFi/cellular communication and finally in the local monitoring station of the operator. The most frequent failure occurrences have been

obtained from past experiences on real architectures and from the existing literature, trying to relate failure occurrences with potential causes (faults).

Considering the general architecture, the remote center location is omitted, since it should be more reliable and under the direct control of the staff, who can immediately intervene in case of failures (e.g., they can connect to the system using a different machine). Hence, the focus is on the components which have to be used by other users, who might not be technology experts and who need to rely on a monitoring system able to work even in case of accidental failures.

To perform the FMEA four components/functions have been identified [35]: the node (i.e., the sensor used to monitor a phenomenon), the Intra WSN communication, the Extra WSN communication, and the gateway (i.e., a smartphone).

In figure 1.5 an organization of components and subcomponents is shown.

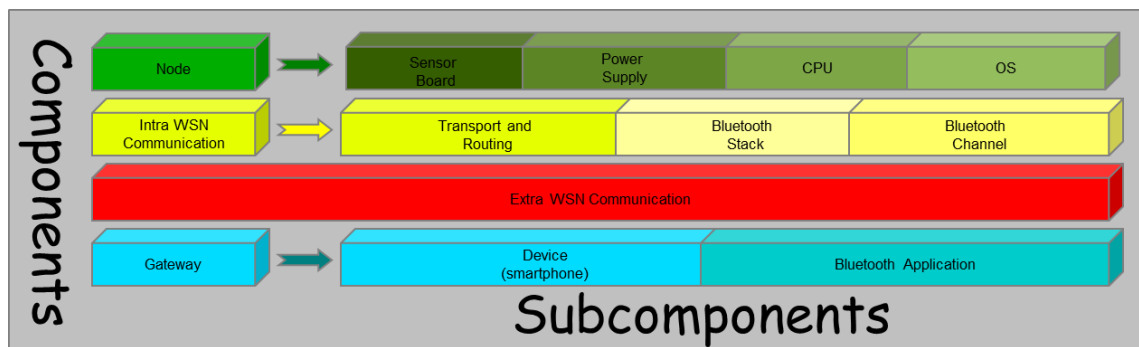


Figure 1.5: Components and subcomponents considered in the FMEA

Four sub-components/subfunctions have been identified for the node component: the sensor board, the power supply unit, the CPU, and the OS are the general components of a node, and their analysis is based on our previous study on sensor networks [36, 39]. The failures

of such devices have been identified starting from existing studies, such as [75, 131].

In table 1.1 the FMEA results are presented. Further it is reported for each failure mode the related severity and probability of occurrence that are represented by a value between 1 and 4. With lower value we identify a weak severity/probability instead with higher values a strong severity/probability [133]. For example if a failure is classified with severity 4 and probability 4 it means that the failure is very dangerous and very probable. But if a failure is classified with severity 1 and probability 1 then there is almost nothing to worry.

Table 1.1 is structured by seven columns. Every row contains the description of a single failure mode. So, considering a possible failure mode that may occur in the monitoring system, we identify from left to right the component (and the subcomponent if it exists) interested by failure mode, the failure mode, the possible effects of failure, the possible cause of failure and finally the severity and probability of occurrence, to highlight the more dangerous and frequent failures.

All of these analyzed failures cause abnormal data readings, or even it can happen that a value is not received at the remote center location; in this case an inaccurate monitoring is provided, potentially resulting in a significant hazard to monitored system. WSN-based monitoring systems must be aware of all the possible failures, in order to react to them or, at least, to detect them. For instance, in case of failure detection, a possible action can be to call to the operator or to call to an emergency contact to suddenly check the system and restore the normal operation of the system.

At the same time, a dependability assessment tool to be used at design time should emulate

Table 1.1: Failure Mode and Effect Analysis of a monitoring system

Component	Sub-component	Potential Failure Mode	Potential Effects of Failure	Potential Causes of Failure	Sev.	Prob.
Node	Sensor Board	Stuck at Zero	The device is out-of-order; it does not deliver any output to inputs	Sensing hardware	4	2
		Null Reading	The device delivers null output values	Sensing hardware	4	4
		Out of Scale Reading	The device delivers no meaningful values	Sensing hardware	3	4
	Power supply	Stuck at Zero	The device is out-of-order; it does not deliver any output to inputs	Natural energy exhaustion	4	4
		Reset	The node resets itself to its initial conditions	Anomalous current request that cannot be supplied by batteries	3	1
	CPU	Stuck at Zero	The device is out-of-order; it does not deliver any output to inputs	Micro-controller	4	4
	OS	Software Hang	The device is powered on, but not able to deliver any output	Operating system's corrupted state	4	3
Intra WSN Communication	Routing	Packet Loss	The radio packet is not delivered	Packet corruption	3	2
		Isolation	The node is not longer connected to the sink node	Failure of all forwarding nodes		
	Bluetooth Stack	Bluetooth stack failure	A Bluetooth module (e.g. L2CAP, BNEP, etc.) fails	Bluetooth stack's corrupted state	3	1
	Bluetooth Channel	Header corruption	Header delivered with errors	Packet corruption	2	1
		Header length mismatch	Header length deviates from the specified one	Packet corruption	2	1
		Payload corruption	Payload delivered with errors	Packet corruption	3	1
	Extra WSN Communication	Data Delivery Failures	The network is not able to deliver the required amount of measurements	The number of failed nodes is more than a given threshold	3	3
Cellular/WiFi network unavailable		Monitoring stopped	Area without cellular/WiFi signal	4	3	
Gateway	Device (the smartphone)	Freeze	The device's output becomes constant; the device does not respond to the user's input.	System's corrupted state	3	3
		Self-shutdown	The device shuts down itself; no service is delivered at the user interface.	Natural energy exhaustion or self-reboot due to corrupted state	4	2
		Unstable behavior	The device exhibits erratic behavior without any input inserted by the user	System/Application corrupted state	4	2
		Output failure	The device delivers an output sequence that deviates from the expected one	System/Application corrupted state	3	4
		Input failure	User inputs have no effect on device behavior	System/Application corrupted state; Natural energy exhaustion	1	1
	Bluetooth Application	Inquiry/Scan Failure	The scan procedure terminates abnormally	A Bluetooth module fails or device out of range	2	3
		Discovery Failure	The discover procedure terminates abnormally	A Bluetooth module fails or device out of range	2	3
		Connect Failure	The device is unable to establish a connection	A Bluetooth module fails or device out of range	4	3
		Packet Loss	Expected packets are not received	Packet corruption	3	1
		Data mismatch	Packets are delivered with errors in the payload	Memoryless channel with uncorrelated errors	3	1

the main failure modes, such as node failures and packet losses, to evaluate the dependability level of the WSN against possible undesired events.

1.5 Open issues

The dependability level of WSN-based systems is challenged by severe impairments, due to their open and evolving nature. An WSN-based system operates proactively, does its job automatically with minimal human intervention, it interacts with humans by messages, alerts, and other forms of natural communication and it should provide its service in a stable, robust and reliable way, even in the presence of component malfunctions, power/-battery break down, or other exceptional conditions.

Generally, faults in a system are unavoidable and they make a system less available, reliable, safe and secure. This combination of heterogeneity, mobility, dynamism, sheer number of devices, accidental failures, and the presence of unavoidable software and hardware defects makes increasingly difficult to build WSN-based systems with verifiable dependability properties.

Despite these compelling issues, there is still little understanding in the literature on the dependability delivered by current research proposal for monitoring environments and on the methods and techniques needed to build more dependable WSN-based systems in the next future.

Dependability assessment of WSNs by means of monitoring systems still represents an interesting research issue on the basis of defined specific metrics.

The aim is to have a support tool that could help to i) calculate the robustness level of the WSN, ii) anticipate critical choices e.g., concerning node placement, routing, iii) mitigate risks, e.g., by forecasting the time when the WSN will not be able to perform with a suitable level of resiliency, and iv) prevent money loss, e.g., providing a criteria to plan and schedule maintenance actions effectively.

It is needed to evaluate the performance and the robustness of a WSN-based system at design time in order to check the level of the fault tolerance of the WSN and it is needed to continuously monitor the WSN when it is running to promptly detect the occurrence of failures. Usually to satisfy these two aims, it is necessary to apply two approaches: the simulative approach to study the network at design time and experimental approach to observe the behavior of the WSN at runtime; to rely on only could be much more advantageous since the same means of verification at design time could be extended to the runtime.

Although various interesting dependability evaluation techniques and tools have been proposed in literature, still there is few attention to define formal-based approaches as alternative dependability evaluation techniques able to check the behavior of the WSN both at design and runtime.

Use of formal verification could represent a crucial key for the dependability assessment of WSN since by means of specifications, defined in a human-friendly language, it is possible to build tools that can help a network manager to do critical choices about the WSN to monitor and to perform continuous monitoring [80] (or runtime verification [143]) of the deployed system using the same set of specifications adopted at design time.

Research is the process of going up
alleys to see if they are blind.

Marston Bates

Chapter 2

Related Research

The approaches adopted in literature to evaluate WSN dependability attributes can be categorized in four classes: experimental, simulative, analytical and formal. The first allows to analyze dependability at runtime, second and third at design time; none of these allows to do assessment both at design and runtime. Formal approaches offer a new opportunity for the dependability study of WSNs but until now there is no work that has proven how use a formal method to perform dependability assessment of a WSN both before and after its deployment. This Chapter revises experimental, simulative, analytical and formal approaches and tools currently used in the field of WSN dependability assessment, including related studies; a comparison of related work is presented to summarize the state-of-art and reason about what is still missing. Finally guidelines to help designers to realize correctness properties specifications of WSN are discussed.

2.1 Experimental approaches

Experimental approaches are used to measure the WSN dependability directly from a real system, during its operation. In the prototyping phase, it is possible to perform an accelerated testing, for example by forcing a fault (by means of *Fault Injection* (FI) [30, 37]); at runtime, it is possible to collect occurring failures directly from system (by means of *Field Failure Data Analysis* (FFDA) techniques [31]).

Fault Injection is defined as the dependability validation technique based on the observation of the system behavior under the presence of faults which are deliberately introduced into the system [73].

Typically FI is used to i) assess the dependability level of a target system, such as an operational systems, a system prototype, or an emulated execution environment (the last two options are used especially in the pre-deployment phase of the system), and ii) to shed some light on the design choice of a system, for instance, showing its potential dependability bottlenecks. FI tries to determine whether the response of a system matches its specifications in the presence of a defined space of faults.

Table 2.1: Fault Injection Tools

Tool	Technique	Fault Model
XCEPTION [92]	SWIFI with exception trigger	Transient faults
FERRARI [77]	SWIFI with interrupt, fork, trap	Transient and permanent faults
FIAT [14]	SWIFI with exception trigger	Bit-flip faults in the memory
NFTAPE [135]	SWIFI with exception trigger	Several types of faults (<i>arbitrary model</i>)
MESSALINE [9]	HWIFI with forcing and insertion	Faults of type <i>stuck-at-0</i> , <i>stuck-at-1</i> , <i>logical bridging</i> , <i>physical bridging</i>
AVR-INJECT [30]	SWIFI with exception trigger	Bit-flip faults in the memory area, code area and special registers

The implementation of tools for injecting faults has been the focus of several studies. Table 2.1 reports a summary of well known tools for fault injection. Beyond their inherent differences, they operate in a similar way: each of them performs a study of the fault-free target, obtaining a 'gold file'; then, it injects a fault (obtaining the 'fault file') and it compares the gold file with the fault file, to evaluate the system behavior in response to the fault.

Among of the tools mentioned in the table 2.1, there is the AVR-Inject Tool. We have

implemented this tool [30] to experimentally study a WSN; it has been useful for the realization of the FMEA discussed in the previous chapter. Unfortunately the AVR-Inject tool cannot be used at design time since it needs a prototype of the system, an assembly code that runs on the sensors and thus it needs very detailed information in design phase. For this reason we had to investigate other solutions to reach the aim of this dissertation since we can use this tool only when a system is in running.

Field Failure Data Analysis (FFDA) [23] of a system represents the set of fault forecasting techniques which are performed at runtime. By means of this analysis, the dependability attributes of an actual and deployed system are measured considering real conditions. A system which is in normal operation is observed and the natural occurring errors and failures are monitored and recorded in log files.

The FFDA is not practical, not feasible for the WSNs since they do not provide log and they have to be lightweight [22].

2.2 Simulative approaches

A simulative approach for assessing WSNs usually makes use of behavioral simulators, i.e., tools able to reproduce the expected behavior of a system by means of a code-based description. Behavioral simulators allow to reproduce the expected behavior of WSN nodes on the basis of the real application planned to execute on nodes. However, it is not always possible to observe non-functional properties of WSNs by means of simulative approaches, since models need to be redefined and adapted to the specific network to simulate.

Typical simulative approaches to evaluate WSN fault/failure models are provided in [127, 82].

In [127] authors address the problem of modeling and evaluating the reliability of the communication infrastructure of a WSN. Authors assume that failures can be categorized in node and network failures.

The first on-line model-based testing technique [82] has been conceived to identify the sensors that have the highest probability to be faulty. The effectiveness of the approach is evaluated in the presence of random noise using a system of light sensors; a fault classification taxonomy for wired sensors is introduced. This technique is not oriented to the wireless sensor networks.

Some work like [103, 126] provide code generation of sensor network applications to perform behavioral simulation and performance analysis.

In [103], a framework for modeling, simulation and code generation of WSNs is presented. The framework is based on Simulink, Stateflow and Embedded Coder; it allows engineers to simulate and automatically generate code of sensor network applications based on MathWorks tools. By means of this tool, an application developer can configure the connectivity of the sensor nodes and can start simulation and functional verification of the application. This framework is able to generate the complete application code for several target operating systems (e.g. TinyOS and MantisOS) from the simulated model.

In [126] a model-driven process (MDD) is presented to obtain a major effort of optimization for WSN applications. In this work a set of modeling languages is the starting point for code generation and performance analysis.

Finally, the network lifetime is analyzed in [58]; to calculate the lifetime of a WSN, the authors perform simulation by means of a Castalia-based approach that models path-loss.

2.2.1 Simulators

Several simulators for WSNs have been proposed in literature, such as *NS-2*, *OMNet++*, *Prowler*, *TOSSIM*, *OPNET* and *Aurora*, [109, 93, 19, 125, 89, 90, 76, 138].

NS-2 [109] is a event-based simulation tool for WSN. It is amply adopted in academic research being open source and easy to use. The simulations are written with C++/C languages and they can be observed graphically by Network AniMator (NAM).

OMNeT++ [93] is a component-based discrete network simulator. Even this simulator is based on C++ language and it has graphical tools for simulation building and evaluating results in real time. The most recent simulation environment built on OMNeT++ is Castalia [19]. This framework was realized for Wireless Sensor Networks, Body Area Networks[148] and networks of low-power embedded devices and it allows to test distributed algorithms and protocols for WSN considering some features of a real WSN like wireless channel, power consumption and considering a real node behavior. Castalia can be used to simulate a wide set of wireless sensor platforms.

Prowler [125] is an event-driven WSN simulator conceived to operate in Matlab environment. Initially it was realized to simulate MICA motes but then it has been extended also for more general platforms. Advantages of Matlab environments are simple implementing of applications, friendly GUI interface and good visualization facilities. By means of this

simulator, it is possible to perform deterministic simulation to test application code of a WSN application and to perform probabilistic simulation to observe the behavior of the sensor nodes.

TOSSIM [89, 90] is the simulator built for TinyOS applications. Actually TOSSIM is an emulator rather than a simulator since it runs actual application code; it allows to simulate the hardware of a sensor but it does not provide information about WSN dependability. Finally, TOSSIM is provided of a visualization tool, TinyViz.

OPNET [76] is a discrete event, object oriented network simulator. this tool was developed initially for military purposes but its large use grew as much to be considered also for commercial use. OPNET is a powerful software that it can be used for research purpose and also as a network design tool.

Finally Avrora [90] is a simulator that adopts an approach which is more oriented to the verification of behavioral properties or performance indicators, and not oriented to the observation of dependability properties. Avrora is a low-level emulator of the AVR processor mainly used to test the behavior of WSNs application prior to their deployment. It executes the disassembled code instruction per instruction and emulates the hardware of the processor and the hardware of the node (memory, LEDs, sensors, radio channel, etc.).

2.3 Analytical approaches

The study of the performance and dependability of WSNs can be performed by means of analytical models [55, 68, 101, 2, 29, 134, 85].

Some of these models [54] are based on a mathematical representation of the WSN characteristics and are solved by means of simulation.

In [55] authors introduce an approach for the automated generation of WSN dependability models, based on a variant of Petri nets.

An analytical model to predict the battery exhaustion and the lifetime of a WSN, *LEACH*, is discussed in [68]. In [101] the authors present a network state model used to forecast the energy of a sensor. AboElFotouh et al. [2] present a probabilistic technique to observe the WSN behavior and discuss about dependability of a WSN; they suppose that the main causes of the failures are related to the crashes, power failures and natural causes. The authors evaluate dependability on the basis of the number of packets received by the sink in a deterministic time (*decision interval*). The dependability is computed evaluating the delay of the expected message.

In [29] authors develop an analytical model to investigate the relation between energy saving and system performance and to observe the effects when sensor sleep/active mode vary. By means of this model, authors can obtain several performance metrics, such as the distribution of the data delivery delay. This work adopts analytical model specifically representing the sensor in sleep/active mode considering channel contention and routing issues. In this work authors model a WSN by means of Markovian techniques; they assess dependability using data delivery resiliency and power consumption metrics.

A linear programming model [134] is introduced to address the problem of “multi-hop lifetime aware routing”. The authors propose a Garg-Konemann-based approach to obtain the minimum cost arborescence for reaching the sink node optimizing the lifetime of sensor

nodes.

Finally in [85] the node aging problem is addressed. The authors try to solve this problem by associating a survivor function for each sensor node (using *Weibull* distribution). The aim of this work is to demonstrate that the node aging process has an important impact on the connectivity at the increasing of the hop distance. By means of a mathematical analysis and a simulation, they observe that nodes at first hop consume their energy because of the aggregation with children nodes. Hence, they assert that the consumption is related to the number of children nodes.

2.4 Formal approaches

Formal approaches are based on formal verification that consists in checking of the correctness of a system taking in account specifications or properties, using formal methods.

The formal verification is performed by providing a proof on an abstract mathematical model of the system. Typically to model systems we can consider labeled transition systems, timed automata, finite state machines, Petri nets, process algebra, hybrid automata, formal semantics of programming languages such as axiomatic semantics, operational semantics and denotational semantics.

2.4.1 Model Checking

One of the well known formal approaches is *model checking* [16]. This technique consists of a systematically exhaustive exploration of the mathematical model (this is possible for

finite models, but also for some infinite models where infinite sets of states can be effectively represented finitely by using abstraction or taking advantage of symmetry). Usually this consists of exploring all states and transitions in the model, by using smart and domain-specific abstraction techniques to consider whole groups of states in a single operation and reduce computing time. Implementation techniques include state space enumeration, symbolic state space enumeration, abstract interpretation, symbolic simulation, abstraction refinement. The properties to be verified are often described in temporal logics, such as linear temporal logic (LTL) or computational tree logic (CTL) [64]. The great advantage of model checking is that it is often fully automatic; its primary disadvantage is that it does not in general scale to large systems; symbolic models are typically limited to a few hundred bits of state, while explicit state enumeration requires the state space being explored to be relatively small.

Typically Model Checking allows to verify if a defined property of a system is satisfied. Thus, the limit of this technique is related to the prediction of a sequence of events. In other words, by means of model checking, an user is able to control if, given an event, the correctness properties are satisfied but is not able to know what will be the behavior of the system after that given event which is instead the goal of our verification approach.

In the work presented in [53] authors consider the Approximate Probabilistic Model Checker (APMC), a tool that allows to approximately check the correctness of extremely large probabilistic systems, to verify it. Instead, we rely on deterministic checking of correctness properties.

SPIN Model Checker

SPIN [71] is an efficient verification system widely diffused for models of distributed software systems. It has been implemented to detect design errors in applications ranging from high-level descriptions of distributed algorithms to detailed code for controlling telephone exchanges. It is a general tool for verifying the correctness in a rigorous and mostly automated fashion. It was written by Holzmann and others members of the Unix group of the Computing Sciences Research Center at Bell Labs.

Systems to be verified are described in Promela (Process Meta Language) [98], which supports modeling of asynchronous distributed algorithms as non-deterministic automata (SPIN stands for "Simple Promela Interpreter"). Properties to be verified are expressed as Linear Temporal Logic (LTL) formulas, which are negated and then converted into Buchi automata as part of the model-checking algorithm.

2.4.2 Linear Temporal Logic (LTL)

Linear temporal logic or linear-time temporal logic [74, 140] (LTL) is a modal temporal logic with modalities referring to time. LTL is a formalism used for specification and verification of properties in reactive systems, as Pnueli defined in his work [116]. A set of infinite sequences is described by a formula of temporal logic (*temporal property*). If all of the computations of temporal property belong to this set then the property is satisfied. The formulas of LTL are useful to define temporal properties of transition systems [60].

In [11] authors present PARADIGM and adopt a propositional LTL (PLTL) to specify and

verify behavior correctness of dynamic systems. They assert that some specifications can be automatically translated to a PLTL-based program, producing an executable model for the real system. This model is composed by a set of logic rules implying, at any time, the current state of process executions.

The complexity of the formulas to define properties in this logic formalism makes more hard its application. Axioms have to be defined by means of particular logic symbols that make difficult the understanding. Moreover, since there is not a way to determine if a property is true or false at given time, the computation of the dependability metrics is more difficult.

2.4.3 Situation Calculus

The situation calculus is a logic formalism designed for representing and reasoning about dynamical domains. It was first introduced by John McCarthy [97]. The main version of the situational calculus, that is used, is based on that introduced by Ray Reiter [120].

The situation calculus represents changing scenarios as a set of first-order logic formulas [130]. The basic elements of the calculus are: the *actions* that can be performed in the world, the *fluents* that describe the state of the world and the *situations*. Actions can be performed in the world and quantified. Fluents describe the state of the world (these are predicates and functions whose value may change depending on the situation). Situations represent a history of action occurrences [47].

A dynamic world is modeled as a series of situations, resulting from the various actions performed within the world. A domain is formalized by a number of formulas, namely: i)

Action precondition axioms, one for each action, ii) *Successor state axioms*, one for each fluent, iii) *Foundational axioms* of the situation calculus and iv) other axioms describing the world in various situations; in particular the dynamic world is axiomatized mainly by adding initial world axioms, effect axioms, and successor state axioms. The initial world axioms describe the starting state of the environment, made up of its objects, their position, their properties, and so forth; effect axioms instead describe the effect upon a fluent of performing an action in a given situation. It is also necessary to specify, for each fluent, the non-effect of other actions.

The work described in [46] focuses on detecting and identifying dangerous and abnormal situations - due to patient behaviors - in order to provide assurance to the patient of her safety and to help clinician to assess the state of the disease. It adopts Situation Calculus to define correctness properties to model a situation-awareness scenario and the Golog Logic Programming Language for situation calculus [88] to realize intelligent agents for the detection and recovery. Situation Calculus and Golog are well known for the specification and monitoring of the evolving world in case of robots performing specific actions within a controlled environment.

However the situation calculus is based on global states and the actions are hypothetical without giving information about time.

2.4.4 Event Calculus

Event Calculus was proposed for the first time in 1986 by Marek Sergot and Robert Kowalski [83] and then it was extended by Murray Shanahan and Rob Miller in the 1990s [99].

This language belongs to the family of logical languages and it is commonly used for representing and reasoning of the events and their effects [139]. *Fluent*, *event* and *predicate* are the basic concepts of Event Calculus [124]. For every timepoint, the value of fluents or the events that occur can be specified.

This language is also named *narrative-based*: in the Event Calculus, there is a single time line on which events occur and this event sequence represents the *narrative*.

The most important and used predicates of Event Calculus are: *Initiates*, *Terminates*, *HoldsAt* and *Happens*.

Supposing that e is an event, f is a fluent and t is a timepoint, we have:

- $Initiates(e, f, t)$

It means that, if the event e is executed at time t , then the fluent f will be true after t .

- $Terminates(e, f, t)$

It has a similar meaning, with the only difference being that when the event e is executed at time t , then the fluent f will be false after t .

- $HoldsAt(f, t)$

It is used to tell which fluents hold at a given time point.

- $Happens(e, t)$

It is used when the event e occurs at timepoint t .

Since the normal and failing behavior of a WSN can be characterized in terms of an event flow (for instance, a node is turned on, a packet is sent, a packet is lost, a node stops to work due to crash or battery exhaustion, or it gets isolated from the rest of the network due to the failure of other nodes, etc.), Event Calculus, that is an event-based formal language, can be used to formally specify the occurrence of such events and the response of the WSN to them, to check if given correctness properties are verified. Moreover dependability metrics can be valuated by analyzing the *narrative* generated by a Event Calculus reasoner based on the specification of the target WSN.

An example of narrative is shown in listing 2.1.

Listing 2.1: Example of Event Calculus *narrative*

```
1 1
2 Happens(Disconnect(5, 3), 1).
3 2
4 -IsLinked(5, 3).
5 Happens(Isolate(5), 2).
6 3
7 -IsReachable(5).
8 Happens(Stop(4), 3).
9 4
10 -IsAlive(4).
11 Happens(Isolate(6), 4).
12 5
13 -IsReachable(6).
14 Happens(Isolate(7), 5).
15 6
16 -IsReachable(7).
17 7
18 8
19 9
20 10
```

By the listing 2.1 we can understand that at timepoint 1 there is a disconnection of node 5 from 3 and this event causes an isolation of the node 5; at timepoint 3 node 4 stops to work causing an isolation of nodes 6 and 7. Supposing to calculate the coverage, we are able to measure it knowing the number of isolated nodes in every timepoint.

Finally several techniques are considered to perform automated reasoning in Event Calculus, such as satisfiability solving, first-order logic automated theorem proving, Answer Set Programming and logic programming in Prolog.

To check the proposed correctness properties defined in Event Calculus we use the *Discrete Event Calculus (DEC) Reasoner*. The DEC Reasoner [105, 104] uses satisfiability (SAT) solvers [106] and by means of this we are able to perform reasoning like deduction, abduction, post-diction, and model finding. It is documented in details in [107] in which its syntax is explained (e.g. the meaning of the symbols used in the formulas).

2.4.5 Formal approaches for WSN

Lifetime of WSN is defined and evaluated in [26] by means of a mathematical formalism. In this work a generic definition of sensor network lifetime is presented and it is conceived in such way to incorporate different application requirements, such as i) number of alive nodes, ii) time latency in the delivery process, iii) delivery ratio, iv) connectivity, v) coverage, and vi) availability.

Recently, different formal methods and tools have been applied for the modeling and analysis of WSNs, such as [112], [94] and [18].

In [78] authors apply a formal tool to wireless sensor networks. Authors propose a formal language to specify the WSN and a tool to simulate it. However, the formal specification has to be rewritten if the WSN under study changes.

In [94] authors propose a methodology for modeling, analysis and development of WSNs using a formal language (PAWSN) and a tool environment (TEPAWSN). Authors consider only power consumption as dependability metric that is necessary but not sufficient to assess the WSN dependability (e.g. other problems of WSN such as the isolation problem of a node have been analyzed) and also they apply only simulation.

In [18] authors describe a model-driven performance engineering framework for WSNs (called Moppet). This framework uses the Event Calculus formalism to estimate the performance of WSN applications in terms of power consumption and lifetime of each sensor node; other dependability metrics like coverage, connection resiliency and data delivery resiliency are not considered. The features related to a particular WSN have to set in the framework every time that a new experiment starts.

There are some papers ([113],[122],[115]) that considered the formal method in real-time contexts. In [113] authors model and study WSN algorithms using the Real-Time Maude formalism. Though authors adopt this formalism, they use NS-2 simulator to analyze the considered scenarios making the work very similar to simulative approaches.

The work presented in [122] describes a new formal model for the specification and the validation of WSN. Authors assert the use of rigorous formal method in specification and validation can help designers to limit the introduction of potentially faulty components during the construction of the system. They consider a WSN as a Reactive Multi-Agent

System consisting of concurrent reactive agents. In this paper dependability metrics are not treated and calculated and authors just describe the structure of a Reactive Decisional Agent by means of a formal language. Also, no case studies are reported to validate their proposal.

Patrignani et al. in [115] consider policies to monitor wireless sensor network applications in a WSN middleware characterized by a Component and Policy Infrastructure (CaPI); by means of a formalization they are able to catch dangerous or undesired effects which may compromise the correct behavior of a WSN application. In this work it has been developed a prototype that operates on the basis of a application topology in terms of communicating nodes and a set of properties to satisfy. Even if authors confirm that one of the most important benefits of formal approach is that problems occurring at runtime can be detected, they model a static and not dynamic network configuration, focusing only on security (encryption and decryption messages) and resource usage problems and in their scenario they do not consider other dependability metrics (coverage, data delivery resiliency, ...).

An open issue with formal specifications of WSNs is that they need to be adapted when changing the target WSN configuration, e.g., in terms of the number of nodes and topology. To address this problem, it is necessary to provide separated specifications and thus conceive two logical sets of specifications: a general specification for WSN correctness properties that is valid for any WSN, and a structural specification related to the topology of the target WSN, designed in order to be generated automatically.

2.5 Comparison of Related Work

In this section it is shown and discussed a comparison of the related work presented in the previous paragraphs in which it emerges a lack of a work that satisfies all the chosen characteristics.

By means of the figure 2.1, the analyzed work and their characteristics are reported.

In the grid on the rows there are the approaches, tools and models considered in the related work; on the columns there are the properties chosen to highlight the differences. For each work it has been analyzed if some formal method has been used, if it is focalized on WSN, what dependability metrics have been considered, if specifications have been separated (in case of use of formal method), if the work is supported by some tool, if What-if analysis has been considered, if Runtime Verification has been considered and finally if some case study has been presented.

In particular we have considered the following features:

- *WSN* to determine if the related work is focused on the Wireless Sensor Networks;
- *WSN Dependability metrics* to determine if the related work considers the following dependability metrics: *coverage*, *connection resiliency*, *data delivery resiliency*, *power consumption*;
- *Formal Method* to determine if the related work is based on some formal method (e.g. model checking, LTL, Event Calculus, etc.) and in particular if the work adopts an approach that provides *Separated specifications*: we want to verify if the related work applies a modular solution considering two logical sets of specifications: a general

correctness specification, valid independently of the particular WSN under study, and a structural specification related to the properties of the target WSN (e.g., number of nodes, topology, channel quality, initial battery charge);

- *Tool* to determine if the related work proposes a novel tool to support designers;
- *What-if analysis* to determine if in the related work the target system is observed under precise circumstances;
- *RV* to determine if the related work proposes a runtime verification technique;
- *Case study* to determine if the related work considers case studies in order to validate the proposed work.

From the survey of the literature it is possible to assert that few apply formal approaches to study the behavior of WSN from the same perspective we do in this work. Among the most important dependability metrics, the power consumption is the only one that has been considered extensively, instead data delivery resiliency and connection resiliency are the least analyzed.

Let us note that for APMC work it is not possible to consider it as contribution for WSN since authors assess data delivery resiliency and power consumption for cabled sensor networks.

The majority of papers propose a tool and present results by means of a case study applying what-if analysis.

There is no work that considers specifications defined separately (see the *Separated specifications* column); this is a disadvantage that it will be overcome with the ADVISES tool proposed in the next chapters: in fact ADVISES tool it is the first work that considers separated specifications considering the general specifications on one side and the structural specifications dependent on the WSN topology on another side.

Therefore looking the figure 2.1 there is no work that meets all of the important characteristics identified by the columns and this comparison helps to make the contribution of the thesis more robust and useful demonstrating its novelty in the field of dependability research for WSN.

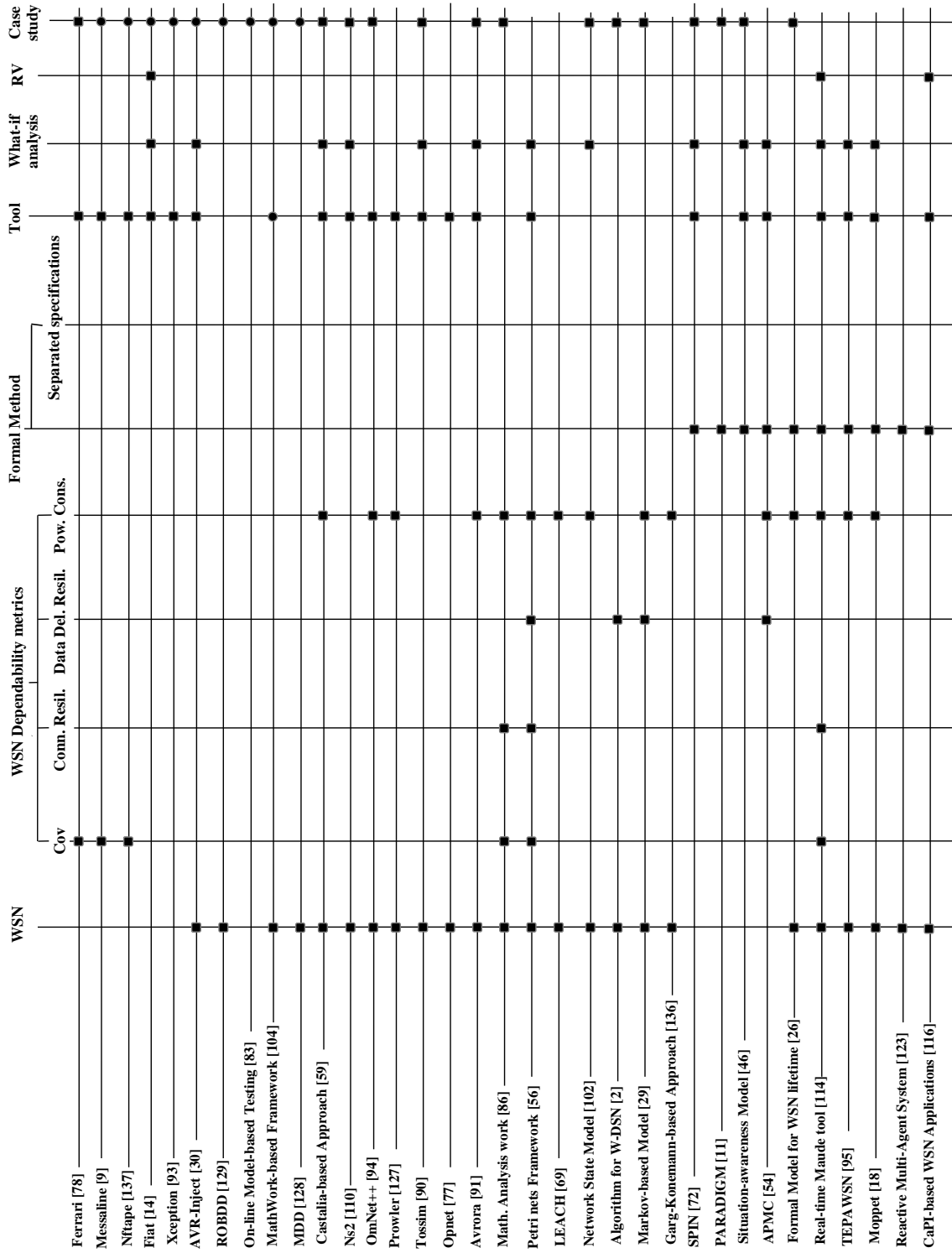


Figure 2.1: Comparison of Related Work

2.6 Discussion

All the analyzed work presented interesting methods and/or techniques which give a contribution for the dependability assessment in WSN. These methods have been grouped in four categories: experimental, simulative, analytical and formal.

Experimental methods are used to evaluate a real system and therefore they need for a existent prototype; they are useful at runtime since through these methods do experiments directly on the real system from which they collect data. Simulative and analytical may be adopted in the design phase: they model a system and make an estimate of reliability before of the system release. Finally formal methods make use of correctness specifications and they can be used at runtime too by means of runtime verification techniques.

In table 2.2 a classification of the presented approaches is shown.

Table 2.2: Approach Classification

Approach	Verification	
	Static	Dynamic
Experimental	×	✓
Simulative	✓	×
Analytical	✓	×
Formal	✓	✓

Currently there are proposals in the literature documenting the application of formal methods to model the WSN but they only focus on some dependability metric like lifetime and power consumption; it is necessary to provide a method of assessing the dependability not only in terms of lifetime and power consumption but also in terms of other important key dependability metrics, such as coverage, connection resiliency to undesired events and data delivery resiliency.

Moreover, formal methods appear to be an attractive solution for the verification of dependability both at design time that at runtime by defining one specification for the system suitable for both purposes; the lack of a formal approach that can be applied for doing static and dynamic verification in WSN remains an open issue.

The contribution of this dissertation wants to go beyond. Indeed it is proposed an automatic process, supported by a user friendly tool, for adapting the specifications to the target WSN and for computing dependability metrics automatically, starting from the analysis of the reasoning output, during the design and when the WSN operates in a real scenario; the proposed approach can help to bridge the gap between system engineers and formal methods experts. The proposed automated dependability verification, using Event Calculus, calculates the mentioned dependability metrics giving information or warning messages to the user.

2.7 Workflow towards a Methodology for Dependable WSN-based Applications

In this paragraph we present and discuss the guidelines to define specifications for WSN-based applications using Event Calculus. The workflow illustrated in figure 2.2 shows the steps that we followed to realize verifiable specifications. This workflow is characterized by four sequential blocks: *Informal Modeling*, *General Correctness Specification Modeling*, *Structural Correctness Modeling* and *Verification/Testing*; all the workflow consists of 13 tasks.

2.7.1 Informal Modeling

In the *Informal Modeling* block the application domain and correctness properties are analyzed.

Task 1.1: Informal Analysis of domain

Using natural language the application domain is described in a textual form useful to understand what are the events, the fluents, the parameters (number of sensors, number of packets, etc.).

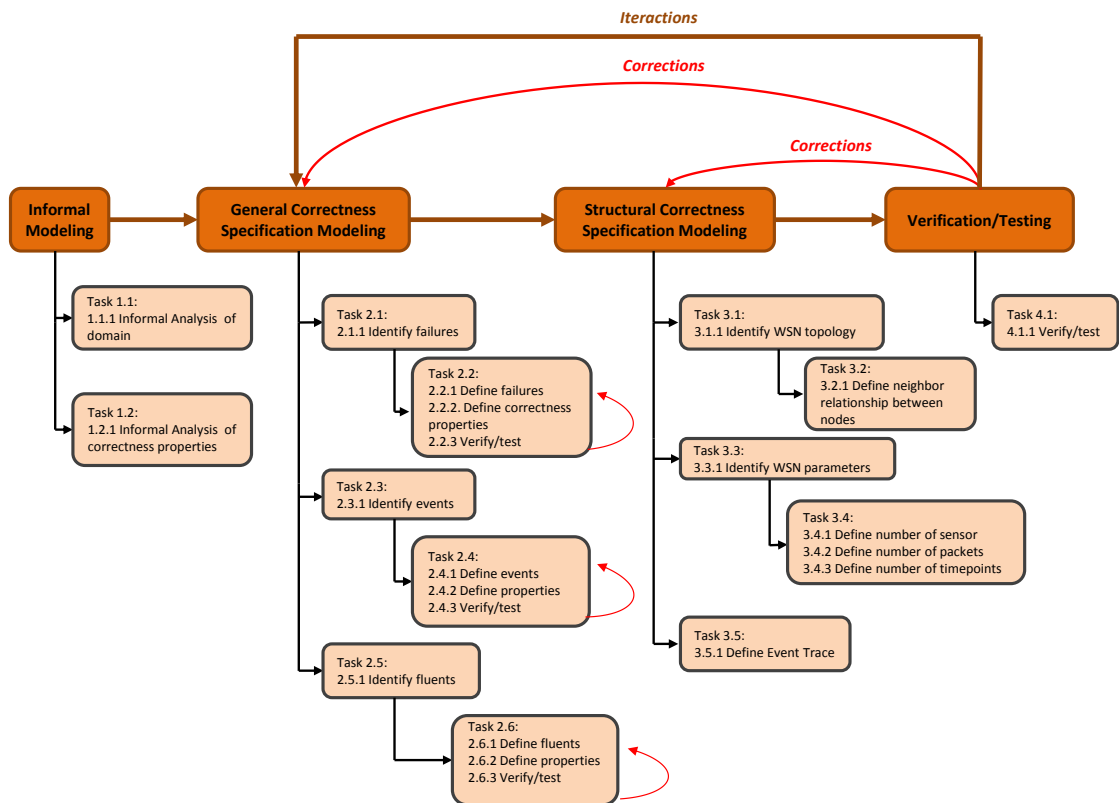


Figure 2.2: The workflow for designing specifications for WSN-based applications (4 blocks - 14 tasks).

Task 1.2: Informal Analysis of correctness properties

After having defined the application domain, in this task the correctness properties to verify are described, for example “*A sensor is completely isolated if there is no path between this node and the sink node*”, “*If a packet is not delivered to the sink node due to some failure (node crash or disconnection between a couple of sensors), the packet is lost*”, “*If a node is discharged, it stops its operation*”.

2.7.2 General Correctness Specification Modeling

This block includes the identification of the correctness properties to detect which are valid for every WSN. In particular failures, events and fluents are identified, defined and refined after a verification and testing subprocess.

Task 2.1: Identify failures

This task identifies the failures that can occur in a WSN (e.g. isolation event, packet lost event, battery exhaustion event). To perform this task we are based on results of FMEA [34, 35] discussed in the chapter 1.

Task 2.2: Define the failures, the correctness properties, and verify/test

In this task failures, that are identified in task 2.1, and related correctness properties are defined by means of Event Calculus language. Finally, verification and testing are performed in this task; this could help to catch some specification errors, requiring their modification, and thus verifying again the specification.

Task 2.3: Identify events

This task identifies the events that sensors can generate during their running. For example, a sensor can stop, restart, it can send a packet towards the sink node.

Task 2.4: Define the events, the properties, and verify/test

Formally in this task the identified events are defined in Event Calculus language; all their properties are defined (e.g. sink node cannot ever be switched off) and verification and testing of these events are performed. Also in this task, if after verification there is some error, the specification of events has to be reviewed and corrected.

Task 2.5: Identify fluents

This task identifies the fluents associated to the events. Their values are dependent on the events. An example of fluent is *IsAlive*(n): if the fluent is true then it means that the sensor n is alive (i.e. it is working) else it means that the sensor is not working.

Task 2.6: Define the fluents, the properties, and verify/test

Formally in this task the identified fluents are defined in Event Calculus language; defining their properties it is possible to establish the initial conditions of the WSN (e.g. at timepoint 0 all the sensors are reachable and are started) and verification and testing of these fluents are performed. Also in this task, if after verification there is some error, the specification of fluents has to be reviewed and corrected.

2.7.3 Structural Correctness Modeling

This block is mainly related to the topology of the WSN and completed by parameters regarding the structure of the WSN (number of sensors, number of packets) and by the chosen observation time (number of timepoints). Differently from the previous step, this specification varies on the basis of the characteristics of the target WSN.

Task 3.1: Identify WSN topology

This task focuses on the structure of topology WSN. In particular the specific topology to consider is identified.

Task 3.2: Define neighbor relationship between nodes

In this task, it is formally defined the WSN topology. To specify the topology, the predicate *Neighbor* is used to indicate how nodes are linked in the topology.

Task 3.3: Identify WSN parameters

In the Structural Specifications modeling, it is necessary identify what are the main WSN parameters to formally declare.

Task 3.4: Define number of sensors, number of packets, number of timepoints

In this task we have to declare the number of the sensors of the topology, the number of the packets that every sensor can send and the number of timepoints to select the observation time.

Task 3.5: Define Even Trace

In this task, we can formally declare, by means of *Happens* predicates, any initial sequence of events of interest for the designer, e.g., to test the defined specifications in the previous steps.

2.7.4 Verification/Testing

The last block concludes the workflow. The aim is to formally verify or test the entire set of specifications by means of a Event Calculus reasoner. If some error is detected at this phase, possible corrections may be performed in some previous block.

Task 4.1: Verify/test

The aim of this task is to verify if the defined general correctness specifications and the defined structural specifications are correct or have to be reviewed to reach the aim without errors.

Chapter 3

Static Verification

Undesired events, such as node crash and packet loss, may undermine the dependability of a WSN. Their effects need to be properly assessed from the early stages of the development process onwards to minimize the chances of unexpected problems during use.

This chapter presents a methodology for the static verification of WSN based systems using the Event Calculus formal language. In particular we show how the formal specification can be used to verify the design of a WSN in terms of its dependability properties defining a set of correctness specifications in order to control the behavior of a generic WSN, coupled with specific structural specifications describing the target network topology to evaluate. We present two different types of analysis: what-if analysis and robustness checking. Finally an application scenario concludes the chapter describing this type of verification in practice.

3.1 Rationale of the static verification technique

Formal methods are widely adopted in the literature to verify the correctness of a system specification at design time. However, their practical use for the verification of dependability properties of WSNs has received little attention, due to the distance between system engineers and formal methods experts and the need to re-adapt the formal specification to different design choices. Even if some development teams would invest on the definition of a detailed specification of WSN correctness properties, a design change (e.g., different network topology, number of produced packets) could require to rethink the formal specification, incurring in extra undesirable costs.

To address the issues listed above, a methodology for performing formal dependability verification of WSN at design time is presented.

In particular in this chapter we are going to describe the formal specification of WSN correctness as two logical sets:

1. ***general correctness specification*** - a set of correctness properties specifications, valid independently of the particular WSN under study
2. ***structural specification*** - a set of specifications and parameters related to the properties of the target WSN, e.g., number of nodes, network topology, quality of the wireless channel (in terms of disconnection probability), and initial charge of batteries.

The general specification is defined once and used across different WSN designs. The structural specification, instead, has to be adapted when changing the target WSN.

As dependability metrics, we consider *connection resiliency*, *coverage*, *data delivery resiliency* and *power consumption*, introduced in chapter 1, that are the most important dependability metrics for WSN. The proposed static verification technique is characterized by two different types of analysis (starting from the same specification):

1. ***what-if analysis*** to verify how the WSN behaves in response to a given sequence of events of interest for the designer,
2. ***robustness checking*** to verify the long term robustness of the WSN against random sequences of undesired events, useful to identify corner cases and dependability bottlenecks.

3.2 General Correctness Specification

The proposed verification framework is founded on the definition of a core formal specification in Event Calculus. The main idea is to formalize the correctness properties allowing engineers to verify if a given WSN design, specified in terms of number of nodes, position of nodes, channel quality, and initial battery level, is able to satisfy given design constraints.

The metrics, defined in chapter 1, are evaluated by analyzing the narrative generated by the reasoner based on the specification of the target WSN.

Following the *workflow for designing specifications for WSN-based applications*, discussed in section 2.7, in this phase we are performing the second step (figure 3.1).

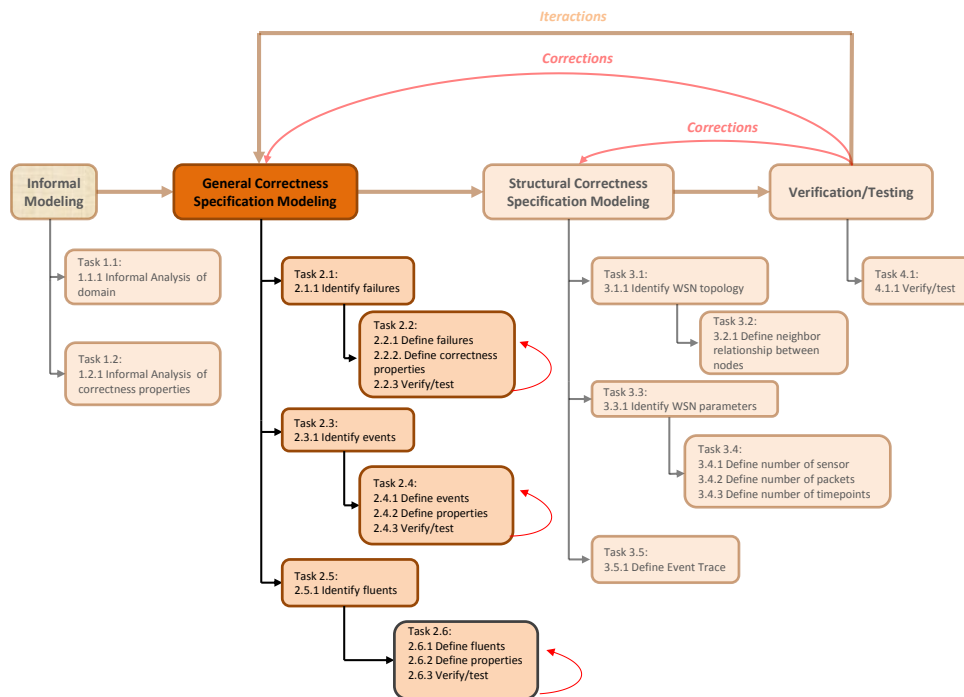


Figure 3.1: Workflow for designing specifications for WSN-based applications: General Correctness Specification Modeling

The general correctness specification is described in the following. It specifies that a WSN performs correctly if none of the following undesired events (or *failures*) happen:

1. *isolation event*, i.e., a node is no more able to reach the sink;
2. *packet loss event*, i.e., a packet is lost during the traversal of the network;
3. *battery exhaustion event*, i.e., a node stops to work since it has run out of battery.

The first two kind of events can be caused by more “basic events”, such as the stop of one or more nodes (e.g., due to crash or battery exhaustion) or the temporary disconnection of a node to its neighbor(s) due to transmission errors. The last kind of event is generated by considering the initial battery charge and the energy request of nodes due to packet sending and receiving activities (in general assumed to be power demanding activities with respect to CPU activities [118]).

We concentrate on the three main events described previously considering the results of a *Failure Modes and Effect Analysis* conducted on WSNs in [36, 35, 34] and described in chapter 1. Moreover the approach allows to extend the specification with other events, if needed.

3.2.1 Isolation event

The isolation event happens when a node is no more able to reach the *sink* of the WSN, i.e., the gateway node where data are stored or processed. The isolation might be caused by more simple, basic events, such as a stop of a node, due to an arbitrary crash or battery exhaustion, and the disconnection of a node from another node.

For instance, let us consider the figure 3.2 and let us suppose that node i is the only one allowing the transmission of data between the sink node and the subnet A . We want to check when the subnet A is isolated from the rest of network. We suppose that node i is connected with node j and k . If node i fails, the nodes j and k (and the all the nodes of the subnet A) are alive but isolated and so the whole subnet A is isolated.

More in general, if a subnet depends by a node and this node becomes isolated then all of the nodes of the subnet are isolated.

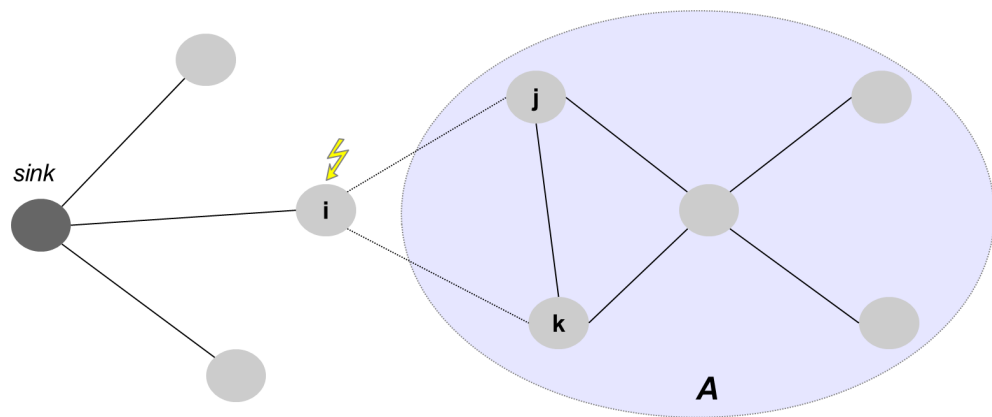


Figure 3.2: Isolation of a WSN subnet

In table 3.1 we report the basic elements (sorts, events and fluents) used for the specification. We distinguish basic events from generated events. These last events are generated by the reasoner on the basis of the specification and of the sequence of basic events actually occurred.

Listing 3.1 shows the rules that represent the core of the specification for an isolation event. In lines 1-7 we define a rule to verify when a node becomes isolated. A sensor can

Table 3.1: Basic elements of the specification for the isolation event

Elements	Name	Description
Sorts	<i>sensor</i>	reference sensor for events and fluents
	<i>to_sensor</i>	sensor used in case of connection (i.e. a sensor connects to another sensor)
	<i>from_sensor</i>	sensor used in case of disconnection (i.e. a sensor disconnects from another sensor)
Basic Events	<i>Start(sensor)</i>	it occurs when a sensor turns on
	<i>Stop(sensor)</i>	it occurs when a sensor turns off
	<i>Connect(sensor, to_sensor)</i>	it occurs when a sensor connects to another sensor
	<i>Disconnect(sensor, from_sensor)</i>	it occurs when a sensor disconnects from another sensor
Generated Events	<i>Isolate(sensor)</i>	it occurs when a sensor is isolated from the network
	<i>Join(sensor)</i>	it occurs when there is at least a connection between a sensor and one or more sensors
Fluents	<i>IsAlive(sensor)</i>	true when a <i>Start</i> event occurs for a sensor
	<i>IsLinked(sensor, to_sensor)</i>	true when a <i>Connect</i> event occurs
	<i>IsReachable(sensor)</i>	true when a sensor is reachable from the sink node

be isolated if it is initially reachable ($\text{HoldsAt}(\text{IsReachable}(\text{sensor}), \text{time})$), alive ($\text{HoldsAt}(\text{IsAlive}(\text{sensor}), \text{time})$) and, considering a link with another sensor ($\text{Neighbor}(\text{from_sensor}, \text{sensor})$), there is no sensor that is alive, reachable and connected with the sensor ($\neg \{\text{from_sensor2}\} (\text{HoldsAt}(\text{IsAlive}(\text{from_sensor2}), \text{time}) \& \text{HoldsAt}(\text{IsReachable}(\text{from_sensor2}), \text{time}) \& \text{HoldsAt}(\text{IsLinked}(\text{sensor}, \text{from_sensor2}), \text{time})) \& \text{Neighbor}(\text{from_sensor2}, \text{sensor}))$).

Listing 3.1: Correctness Specification for the Isolation event

```

1 [sensor,from_sensor, time] Neighbor(from_sensor,sensor) & HoldsAt(
2   IsReachable(sensor),time) & HoldsAt(IsAlive(sensor),time) &
3   (!{from_sensor2} (HoldsAt(IsAlive(from_sensor2),time) &
4   HoldsAt(IsReachable(from_sensor2),time) & HoldsAt(
5   IsLinked(sensor,from_sensor2),time)) &
6   Neighbor(from_sensor2,sensor)) ->
7 Happens(Isolate(sensor),time).
8
9 [sensor,from_sensor, time] ( !HoldsAt(IsReachable(sensor),time) &
10  HoldsAt(IsAlive(sensor),time) & HoldsAt(IsAlive(
11  from_sensor),time) & HoldsAt(IsReachable(from_sensor),time))
12  & HoldsAt(IsLinked(sensor,from_sensor),time) &
13  Neighbor(from_sensor,sensor) ->
14 Happens(Join(sensor),time).
15
16 [sensor,from_sensor, time] ((HoldsAt(IsAlive(from_sensor),time) &
17  HoldsAt(IsReachable(from_sensor),time) & HoldsAt(
18  IsLinked(sensor,from_sensor),time)) | !HoldsAt(
19  IsReachable(sensor),time) | !HoldsAt(IsAlive(sensor),time))
20  & Neighbor(from_sensor,sensor) ->
21 !Happens(Isolate(sensor),time).
22
23 [sensor,from_sensor,time] ( HoldsAt(IsReachable(sensor),time) |
24  !HoldsAt(IsAlive(sensor),time) | !HoldsAt(
25  IsLinked(sensor,from_sensor),time) |
26  !HoldsAt(IsAlive(from_sensor),time) | !HoldsAt(
27  IsReachable(from_sensor),time)) & Neighbor(from_sensor,sensor)->
28 !Happens(Join(sensor),time).

```

Also we report (in lines 9-14) another rule which allows to check a Join event. In particular the rule declares that if a sensor is not reachable, because is isolated, (`!HoldsAt(IsReachable(sensor),time)`) and alive (`HoldsAt(IsAlive(sensor),time)`) and its neighbor sensor is also alive and reachable (`HoldsAt(IsReachable(from_sensor),time)`)& `HoldsAt(IsLinked(sensor , from_sensor) , time)`) and there is a connection between them (`HoldsAt(IsLinked(sensor,from_sensor),time)`) then the sensor can join the network (`Happens(Join(sensor),time)`) and becomes reachable again.

3.2.2 Packet Loss event

When there is a failure in a node or a link between a couple of nodes is disrupted then all of the packets that are in delivery towards this node are lost. In turn, these packets are not delivered to the sink.

For instance, let us consider figure 3.3. Node *A* sends a packet *pkt* to node *B*. If node *B* crashes it cannot receive the packet from node *A* and forward it to a node towards the sink; so the packet is lost.

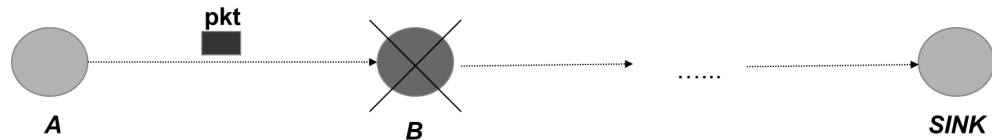


Figure 3.3: Example of packet loss

In table 3.2 we report the basic elements (sort, events and fluents) used for the specification related to the packet loss event. Again, events are divided in basic and generated ones. In the listing 3.2 there are the rules that represent the core of the specification for the packet loss event.

Listing 3.2: Correctness Specifications for the packet loss event

```

1 [pkt,sensor,to_sensor,time]Happens(Send(pkt,sensor),time) &
2   Neighbor(to_sensor, sensor) ->
3 Happens(Forward(pkt,sensor,sensor,to_sensor),time).
4
5 [sensor,from_sensor,source,pkt,time]
6   HoldsAt(IsOnChannel(pkt,source,from_sensor, sensor),time) &
7   (!{sensor2,from_sensor2,source2,pkt2} HoldsAt(
8     IsOnChannel(pkt2,source2,from_sensor2, sensor2),time) &
9     source!=source2 & sensor=sensor2) & HoldsAt(
10  IsAlive(sensor),time) & HoldsAt(IsLinked(from_sensor,
11  sensor),time) & !HoldsAt(IsLost(pkt,source),time) ->

```

```

12 Happens(Catch(pkt,source,sensor, from_sensor),time).
13
14 [sensor,from_sensor,pkt,from_sensor2, pkt2,source, source2, time]
15     HoldsAt(IsOnChannel(pkt,source,from_sensor, sensor),time) &
16     HoldsAt(IsOnChannel(pkt2,source2,from_sensor2, sensor),time) &
17     source<source2 & HoldsAt(IsAlive(sensor),time) & HoldsAt(
18     IsLinked(from_sensor,sensor),time) & !HoldsAt(
19     IsLost(pkt,source),time)->
20 Happens(Catch(pkt,source,sensor, from_sensor),time).
21
22 [sensor,to_sensor,from_sensor,source,pkt,time]
23     Happens(Catch(pkt,source,sensor, from_sensor),time) &
24     Neighbor(sensor, from_sensor) & Neighbor(to_sensor, sensor) &
25     HoldsAt(IsAlive(sensor),time) & !HoldsAt(IsLost(pkt,source),time) ->
26 Happens(Forward(pkt,source,sensor,to_sensor),time+1).
27
28 [from_sensor,source,pkt,time]Happens(Catch(pkt,source,1, from_sensor),time) ->
29 Happens(Receive(pkt,source),time).
30
31 [sensor,to_sensor,source,pkt,time]HoldsAt(IsInDelivery(pkt,source),time) &
32     !HoldsAt(IsLost(pkt,source),time) & HoldsAt(
33     IsOnChannel(pkt,source,sensor, to_sensor),time) & Neighbor(
34     to_sensor, sensor) & (!HoldsAt(IsLinked(sensor,to_sensor),time) |
35     Happens(Stop(to_sensor),time) | !HoldsAt(IsAlive(to_sensor),time) ) ->
36 Happens(PacketLoss(pkt,source),time).

```

By means of the rule defined in lines 1-3, we assert that when a Send event comes from the WSN ($\text{Happens}(\text{Send}(\text{pkt}, \text{sensor}), \text{time})$), a Forward event is generated ($\text{Happens}(\text{Forward}(\text{pkt}, \text{sensor}, \text{sensor}, \text{to_sensor}), \text{time})$) and the packet delivery starts.

In the lines 5-12, we define the Catch event ($\text{Happens}(\text{Catch}(\text{pkt}, \text{source}, \text{sensor}, \text{from_sensor}), \text{time})$) that occurs when, being the packet on the channel between a couple of nodes ($\text{HoldsAt}(\text{IsOnChannel}(\text{pkt}, \text{source}, \text{from_sensor}, \text{sensor}), \text{time})$), the receiving sensor is alive ($\text{HoldsAt}(\text{IsAlive}(\text{sensor}), \text{time})$), the packet is not lost ($\text{!HoldsAt}(\text{IsLost}(\text{pkt}, \text{source}), \text{time})$) and there is a connection between the nodes ($\text{HoldsAt}(\text{IsLinked}(\text{from_sensor}, \text{sensor}), \text{time})$). Note that we also included a concurrency control in order to manage

Table 3.2: Basic elements of the specification for the packet loss event

Elements	Name	Description
Sorts	<i>pkt</i>	it is the packet id
	<i>source</i>	it is the sensor that sends the packet
Basic Events	<i>Send(pkt,source)</i>	it occurs when a sensor starts the delivery of a packet (<i>pkt</i>) toward the sink node
Generated Events	<i>Receive(pkt, source)</i>	it occurs when sink node receives a packet
	<i>Catch(pkt,source, sensor, from_sensor)</i>	it occurs when a sensor catches a packet from one of its neighbor sensors
	<i>Forward(pkt,source, sensor, to_sensor)</i>	it occurs when a sensor, once caught a packet sent by a sensor, forwards it to its neighbor sensor
	<i>PacketLoss(pkt, source)</i>	it occurs when a packet is lost
Fluents	<i>IsInDelivery(pkt, source)</i>	true when a packet delivery starts
	<i>IsOnChannel(pkt,source, sensor, to_sensor)</i>	true when a packet is in transmission on the channel between two nodes
	<i>IsLost(pkt, source)</i>	true when a packet is lost

packets that come from different nodes ($\{sensor2, from_sensor2, source2, pkt2\} HoldsAt(IsOnChannel(pkt2, source2, from_sensor2, sensor2), time) \& source \neq source2 \& sensor = sensor2$).

Also, we can check the end of the packet delivery (lines 28-29) when it is caught ($Happens(Catch(pkt, source, 1, from_sensor), time)$) by the sink node (node 1) and so it is received ($Happens(Receive(pkt, source), time)$).

Finally, in lines 31-36 we report the rule that checks when a packet is lost.

Considering a packet, that is not lost ($\neg \text{HoldsAt}(\text{IsLost}(\text{pkt}, \text{source}), \text{time})$) and is in delivery toward the sink node ($\text{HoldsAt}(\text{IsInDelivery}(\text{pkt}, \text{source}), \text{time})$) and is on the channel between two nodes ($\text{HoldsAt}(\text{IsOnChannel}(\text{pkt}, \text{source}, \text{sensor}, \text{to_sensor}), \text{time})$), when a disconnection event between the two nodes ($\neg \text{HoldsAt}(\text{IsLinked}(\text{sensor}, \text{to_sensor}), \text{time})$) or the failure of receiving node occurs ($\text{Happens}(\text{Stop}(\text{to_sensor}), \text{time}) \mid \neg \text{HoldsAt}(\text{IsAlive}(\text{to_sensor}), \text{time})$) then there is a packet loss event ($\text{Happens}(\text{PacketLoss}(\text{pkt}, \text{source}), \text{time})$).

3.2.3 Battery Exhaustion event

A battery exhaustion event happens when a node completely consume its available energy. To this aim, we adopt a sort in the specification for each sensor node, called *level*, which represents the level of the battery of the node, and that is decremented each time the node sends, catches, or forwards a packet.

In table 3.3 we report the basic elements (sort, events and fluents) used for the specification related to the battery exhaustion event. For this specification, the basic events are the same shown for the previous specifications. Hence, we only report generated events.

In the listing 3.3 there are the rules that represent the core of the specification for the battery exhaustion event.

When a forward event occurs (due to a sending of a packet) ($\text{Happens}(\text{Forward}(\text{pkt}, \text{source}, \text{sensor}, \text{to_sensor}), \text{time})$) and the battery level of the sensor is positive ($\text{HoldsAt}(\text{BatteryLevel}(\text{level}, \text{sensor}), \text{time}) \& \text{level} > 0$) then we consider the new consume level (lines 1-3), ($\text{Happens}(\text{Old_Consume}(\text{level}, \text{sensor}), \text{time}) \& (\{\text{levelnew}\} \text{levelnew} = \text{level} - 1 \quad \& \quad \text{Happens}(\text{New_Consume}(\text{levelnew}, \text{sensor}), \text{time}))$).

In the lines 5-7 we can see that when the battery level of a sensor, that is alive ($\text{HoldsAt}(\text{IsAlive}(\text{sensor}), \text{time})$), is zero ($\text{HoldsAt}(\text{BatteryLevel}(0, \text{sensor}), \text{time})$) then a failure for the node occurs due to battery exhaustion ($\text{Happens}(\text{Stop}(\text{sensor}), \text{time})$).

Table 3.3: Basic elements of the specification for the battery exhaustion event

Elements	Name	Description
Sorts	<i>level</i>	the current battery level of a sensor
	<i>levelnew</i>	the new battery level of a sensor after RX/TX operations related to a packet
	<i>capacity</i>	indicates the maximum battery capacity of a sensor, in terms of the maximum number of packets that it can send
Basic Events	-	-
Generated Events	<i>New_Consume(level, sensor)</i>	it occurs when the battery level of a sensor decreases after the forwarding of a packet
	<i>Old_Consume(level, sensor)</i>	event created for specification reasons
Fluents	<i>BatteryLevel(level, sensor)</i>	true when the battery charge of a sensor is equal to a certain <i>level</i>

Listing 3.3: Correctness Specification for the battery exhaustion event

```

1 [pkt, sensor, to_sensor, source, level, time]
2 Happens(Forward(pkt, source, sensor, to_sensor), time) & HoldsAt(BatteryLevel(level,
  sensor), time) & level > 0 ->
3 Happens(Old_Consume(level, sensor), time) & ({levelnew} levelnew = level - 1 & Happens(
  New_Consume(levelnew, sensor), time)).
4
5 [sensor, time]
6 HoldsAt(BatteryLevel(0, sensor), time) & HoldsAt(IsAlive(sensor), time) ->
7 Happens(Stop(sensor), time).

```

3.3 Structural Specification

General correctness specifications are complemented by a structural specification of the target WSN. This is mainly related to the topology of the WSN and completed by parameters regarding the initial level of batteries and the quality of channels. Differently from the specifications described in the previous sub-section, this specification varies on the basis of the characteristics of the target WSN.

Following the *workflow for designing specifications for WSN-based applications* in this phase we are performing the third step (figure 3.4).

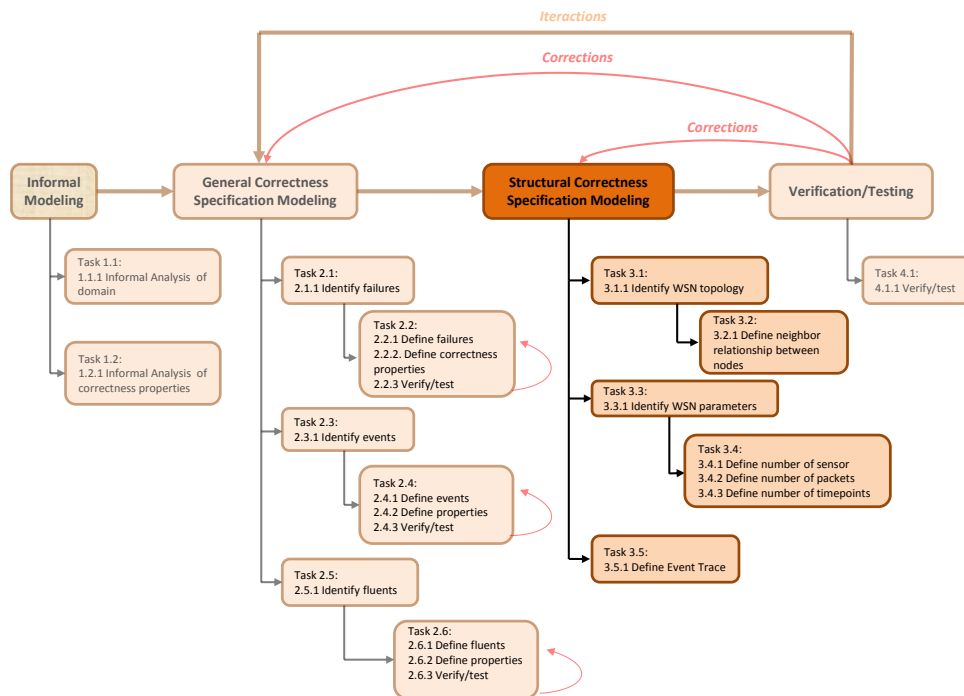


Figure 3.4: Workflow for designing specifications for WSN-based applications: Structural Correctness Specification Modeling

To specify the topology, we use the predicate *Neighbor* (already used in the previous specifications) to indicate how nodes are linked in the topology. For instance, considering the topology in figure 3.5, let us suppose node *i* is connected with *j* and *k* and let us consider a tree graph where the sink node (root node) is the node *i* and the nodes *j* and *k* are child nodes.

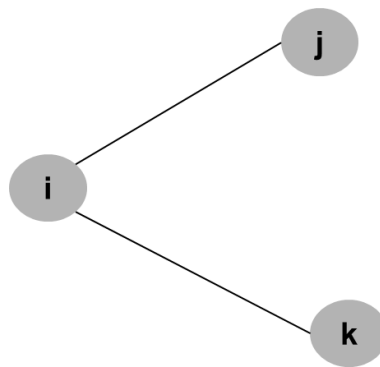


Figure 3.5: Example of topology of a WSN

The resulting specification is reported in listing 3.4, where *sensor1* is the parent node (*i*) and *sensor2* are the child nodes (*j* and *k*). Clearly, this specification can be changed easily if the topology of the WSN changes.

Listing 3.4: Example of Neighbor predicate Specification

```

1 [sensor1,sensor2] Neighbor(sensor1,sensor2) <-> (
2 (sensor1=i & sensor2=j) |
3 (sensor1=i & sensor2=k)
4 ).
```

The role of the *Neighbor* predicate is very important to understand when an axiom can be applied. Let us examine the axiom related at a possible isolation (lines 1-7 of listing 3.1)

and let us apply it for the figure 3.5. The described implication is true when, given a couple of nodes ($sensor$, $from_sensor$), the conditions about isolation are true and there is a link between nodes (in this case, between node j and i or between node k and i). This, for instance, can never be true for the couple of nodes j and k , since there is not a physical link between them.

Regarding the parameters, their values can be used to check the correctness properties of the WSN under different conditions, i.e., under different assumptions on the initial charge of batteries (e.g., to verify a WSN in the middle of its life), or under different environmental conditions affecting the quality of the channels (impacting on the probability to have a disconnection event when checking the robustness of the WSN).

3.4 Types of Analysis

To provide useful support for static verification we introduce two different types of analysis: *what-if analysis*, to verify how the WSN behaves in response to a given sequence of events of interest for the designer, and *robustness checking*, to verify the long term robustness of the WSN against random sequences of undesired events, useful to identify corner cases and dependability bottlenecks.

3.4.1 What-if Analysis

The goal of a what-if scenario analysis is to verify the behavior of the target WSN under precise circumstances. To this aim, we need to indicate an initial event sequence (Event

Trace). For example, as listing 3.5 shows, by means of *Happens* predicates, we can declare that at timepoint 1 sensor *j* stops, at timepoint 3 sensor *k* stops, etc. In this way, by means of the reasoner, we can observe the consequences of any initial sequence of events of interest for the designer, e.g., to test the robustness of the designed topology against the temporary unavailability (failure/recovery) of a given set of nodes, or to quantify to what extent a modification of the topology can be beneficial for the network.

Listing 3.5: Example of initial event trace

```
1 Happens(Stop(j),1).
2 Happens(Stop(k),3).
3 Happens(Start(k),4).
4 Happens(Disconnect(k,i),5).
5
6 completion Happens
```

3.4.2 Robustness Checking

For *Robustness Checking* we intend a technique to statically verify the behavior of a WSN in front of a number of failures that can occur during its operation.

We distinguish two types of robustness checking analysis. The first one aims to analyze the robustness of the network, in terms of coverage, against a variable number of failures (stop and disconnection events), from 1 to n , where n is selected by the user, considering all combinations without repetitions. It is particularly useful to evaluate the coverage and connection resiliency of the network and pinpoint weak points in the topology. The second one aims to analyze how the target WSN behaves during a periodic sending of packets if random failures happen, causing packets to be lost. It is useful to evaluate the data delivery

resiliency of the network. These two types of robustness checking analysis are detailed in the following.

Coverage Robustness Checking

This test is aimed to verify how many node failures the network can tolerate, while guaranteeing a given minimum level of coverage. For example if we consider a network composed by m nodes and a threshold coverage equal to 50%, we may want to understand what are the sequences of failures causing more than $m/2$ nodes to be isolated (i.e., coverage under the specified threshold) and how the resiliency level varies when varying the sequences of failures. This allows to evaluate the maximum (and minimum) resiliency level reachable by a given topology and what are the critical failure sequences, i.e., the shortest ones causing a loss of coverage. These are particularly useful to pinpoint weak points in the network.

We developed an algorithm to generate automatically the sequences of failures (stop and disconnection events) against which checking the robustness of the WSN. The algorithm is aimed to reduce the number of failure sequences to be checked. The principles are to avoid repetitions and to end the sequence as soon as the coverage level becomes lower than the user defined threshold. For instance, we start considering all the cases when there is one failure. By means of the DECReasoner we compute the coverage; if the coverage is above the threshold, the resiliency is surely greater than 1, because there is just one failure and it is tolerated in all cases. In the generic k -th step, we consider sequences of k failures. If the generic sequence $\{f_1, f_2, \dots, f_k\}$ leads to a coverage below the threshold, we do not consider

sequences starting with a $\{f_1, f_2, \dots, f_k\}$ prefix in the $(k + 1)$ – *th* step. By considering the percentage of sequences with k failures where the coverage is above the threshold, let us say $g_k\%$, we can say that the resiliency is k in $g_k\%$ of cases.

The described process is summarized by means of the algorithm for the computation of event sequences with n failures shown in listing 3.6.

Listing 3.6: Algorithm for the computation of event sequences with n failures

```

1 int f=1;
2 while (f<=failures)
3 {
4 {if f==1
5 {
6 Computation of all combinations with 1 failure
7 gen_traces()
8 Storing these traces in a combination file
9 create_and_reason() //Create and make reasoning on these traces
10 new_traces = compute_resiliency() // Resiliency computation
11 Percentage of resiliency: number of traces with coverage upper the threshold (
    dim) / all of the events (total number of single failures)
12 per_resiliency = (dim*100) / events_map.size();
13 }
14 2 or more failures
15 else {
16 for (int ev =0; ev<new_traces.size(); ev++)
17 {
18 gen_traces()
19 Create and make reasoning on these traces
20 create_and_reason(traces_pp,f_count,events_map, coverage,con_resiliency);
21 Resiliency computation: if the experiment with the 'ev'-th trace produces
    coverage false, it is deleted otherwise this trace will stay in 'traces_
    prov' (prouvisory)
22 traces_prov=calcola_resiliency(traces_pp, events_map,f_count,output_r);
23 Insert the traces, contained in 'traces_prov' and related to this \textbf{for
    } cycle, in a global file; in this last file there are all the traces to
    consider for the experiments with upper number of failures.
24 for (int traces_prov_ind =0; traces_prov_ind<traces_prov.size(); traces_prov_ind
    ++))
25 {
26 traces_all.addElement(traces_prov.elementAt(traces_prov_ind));
27 }
28

```

```
29 | Calculate total number of combinations.
30 | * Ex.: if number of failures is 2 and the number of possible failures is 12, we
   | have 12!/(12-2)! = 12*11 = 132
31 | den = fatt(events_map.size()) / fatt(events_map.size()-f_count);
32 | Percentage of resiliency: number of traces with coverage upper the threshold (
   | dim) / all of the combinations (den)
33 | per_resiliency = dim * 100 / den;
34 | new_traces = traces_all;
35 | }
36 | f_count++; // counter of the while cycle
37 | }
38 | }
```

Data Delivery Robustness Checking

In this case we want to check the robustness of the WSN in case of failures when there are packets being delivery on the network.

To do this, we model each link of the WSN on the basis of the Gilbert-Elliott Channel Model [59], and we associate to each link a disconnection probability (indicating the channel quality), set as a parameter by the user together with the structural specification. This way, the user can simulate different scenarios, e.g., environments with different channel conditions.

We simulate that, assuming a periodic WSN, every sensor sends periodically a packet to the sink. At the same time we randomly generate failures (failures node and/or disconnection events), taking into account disconnection probabilities. The data delivery resiliency is computed as the maximum number of failures that can be sustained while a given fraction of packets (above a given threshold) is still received at the sink.

3.5 Application Scenario

In this paragraph we want to show the static verification technique by means of an application scenario; the aim is to describe how it is possible to detect failures and observe their effects in a WSN with Event Calculus.

By means of the application scenario we are going to verify the defined sets of specifications.

In fact, following the *workflow for designing specifications for WSN-based applications* in this phase we are performing the fourth and last step (figure 3.6).

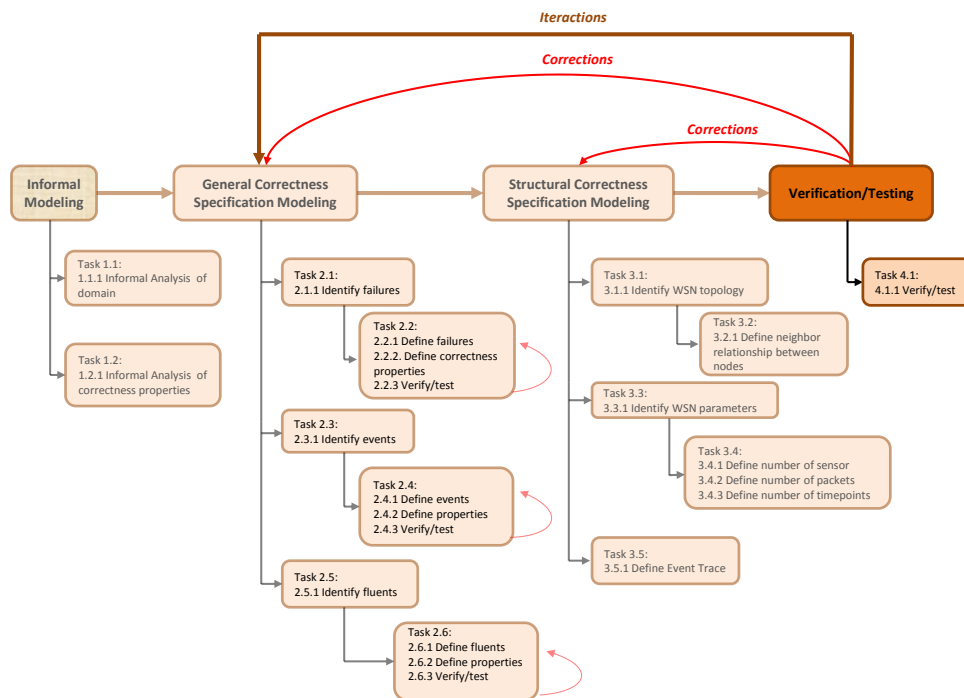


Figure 3.6: Workflow for designing specifications for WSN-based applications: Verification/Testing

Let us consider the figure 3.7.

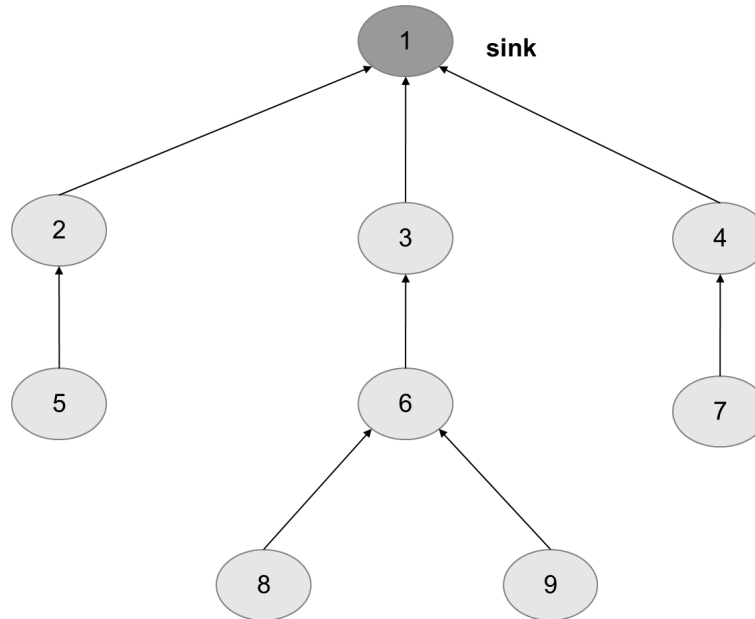


Figure 3.7: WSN topology of application scenario

The network reported in figure is composed by 9 nodes disposed on the basis of a defined topology.

We want to observe the behavior of this WSN in 10 timepoints, supposing that, after two timepoints, node 6 stops, a disconnection between node 7 and node 4 occurs at timepoint 4 and that node 2 stops at timepoint 7. In particular, we are interested to evaluate if the reasoning performed by the DECReasoner on our specifications is correct.

For a coverage threshold set at 60%, we should observe a coverage interval equals to $[0;4]$ (i.e., when node 7 disconnects from node 4 at timepoint 4, four nodes are not reachable, namely 6, 7, 8 and 9), and a connection resiliency equals to 1 (i.e., only the first stop event is tolerated).

In the following, we show the structural specification written in Event Calculus language

and created for this application scenario, and the trace produced by the DECReasoner by providing the WSN connectivity matrix and the initial event trace chosen for this example.

3.5.1 Structural Specification generated by Event Calculus

Previously, we have discussed about the application scenario we want to observe give an occurring particular sequence of events.

Now, we want to discuss about the structural specification file (listing 3.7) in which, by means of Event Calculus syntax, the files used by DECReasoner are loaded, it is formally defined the WSN topology, the event trace to consider and all the parameters that characterized the network and the study are defined. For instance, we define the links between the nodes considering the chosen topology, the events that we have supposed occur during the observation time and the number of the sensors and of the timepoints.

File loading

In the first part (lines 1-2) we have to load the main Event Calculus files (*Root.e* and *EC.e*). *Root.e* file defines the boolean, integer, predicate and function sorts; *EC.e* defines the sorts, functions, predicates and axioms for the Event Calculus.

For our aim, we have to load the *IsolationWorld.e* file, too (line 4). In this manner, the axioms defined to detect isolation events are interpreted for the topology described in this application scenario.

Listing 3.7: Structural specification written in Event Calculus language for WSN-based application scenario

```

1 load foundations/Root.e
2 load foundations/EC.e
3
4 load /home/alessandro-64/IsolationWorld.e
5
6 [sensor1,sensor2] Neighbor(sensor1,sensor2) <-> (
7 (sensor1=1 & sensor2=2) | (sensor1=1 & sensor2=3) | (sensor1=1 & sensor2=4) |
8 (sensor1=2 & sensor2=5) |
9 (sensor1=3 & sensor2=6) |
10 (sensor1=4 & sensor2=7) |
11 (sensor1=6 & sensor2=8) | (sensor1=6 & sensor2=9)
12 ).
13
14
15
16 [sensor] HoldsAt(IsAlive(sensor),0).
17 [sensor,from_sensor] Neighbor(from_sensor, sensor) <-> HoldsAt(IsLinked(sensor,
18     from_sensor),0).
19
20 Happens(Stop(6),2).
21 Happens(Disconnect(7 , 4),4).
22 Happens(Stop(2),7).
23
24
25 completion Happens
26
27 range from_sensor 1 9
28 range sensor 1 9
29 range to_sensor 1 9
30 range time 0 10
31 range offset 1 1
32 option n_models 1
33 option modeldiff on

```

Formal definition of topology

Analyzing the WSN topology, we establish (lines 6-12) how the nodes are linked by means of a predicate symbol *Neighbor*.

Initial conditions

In the lines 16-17 we set some initial conditions. The reasoner considers these conditions when it starts the reasoning.

Every sensor is alive, is reachable and is linked with another sensor on the basis of the WSN topology.

Event Trace

In the application scenario, we insert an event trace (lines 19-21) that is composed by a list of *Happens* predicates that specify nodes and timepoints in which a given event occurs.

The *completion* statement specifies that a predicate symbol (i.e. *Happens*) should be subject to predicate completion.

Parameters

Finally, in the last part we consider ranges of values for sensors and timepoints.

In this case we know that the network is composed by 9 nodes and we want to observe what it could happen in 10 timepoints.

3.5.2 Outcome and Metrics computation

Listing 3.8 reports the outcome produced by the DECReasoner.

The event trace confirms our expectations. We can observe that at timepoint 2 node 6 stops

(line 50), due to a failure, and it is not alive anymore isolating nodes 8 and 9 (lines 53-54). Thus these two nodes become not reachable (lines 56-57). Considering that, because of a disconnection from node 4 (line 58) at timepoint 4, node 7 is isolated (line 61) becoming not reachable (line 63), then this means that at timepoint 5 a total of 4 nodes are isolated. The coverage is computed as the time point of the last failure event causing such isolation, that is 4. Consequently, the connection resiliency is computed by counting the number of failures and disconnection events in the interval $[0; 4]$, excluding the last event; hence, it is equal to 1. At the last, the failure of the node 2 at timepoint 7 (line 65) (and thus the isolation of node 5 (lines 69-70)) do not alter the results.

The dependability metrics can be valuated by analyzing the event flow (the *narrative*) generated by the DECReasoner based on the specification of the target WSN.

In the lines 22-47 the outcome shows the conditions of the WSN at timepoint 0: all of the nodes are alive, are reachable and every node is linked with its neighbor on the basis of the considered topology.

Listing 3.8: Outcome produced by the DECReasoner for WSN-based application scenario

```
1 #
2 # Copyright (c) 2005 IBM Corporation and others.
3 # All rights reserved. This program and the accompanying materials
4 # are made available under the terms of the Common Public License v1.0
5 # which accompanies this distribution, and is available at
6 # http://www.eclipse.org/legal/cpl-v10.html
7 #
8 # Contributors:
9 # IBM - Initial implementation
10 #
11
12 loading /home/alessandro-64/Dottorato/EventCalculus/EC_Application/topology1803EC
   _20130318174304.e
13 loading foundations/Root.e
```

```
14 | loading foundations/EC.e
15 | loading /home/alessandro-64/IsolationWorld.e
16 | 15329 variables and 129112 clauses
17 | relsat solver
18 | 1 model
19 | ---
20 | model 1:
21 | 0
22 | IsAlive(1).
23 | IsAlive(2).
24 | IsAlive(3).
25 | IsAlive(4).
26 | IsAlive(5).
27 | IsAlive(6).
28 | IsAlive(7).
29 | IsAlive(8).
30 | IsAlive(9).
31 | IsLinked(2, 1).
32 | IsLinked(3, 1).
33 | IsLinked(4, 1).
34 | IsLinked(5, 2).
35 | IsLinked(6, 3).
36 | IsLinked(7, 4).
37 | IsLinked(8, 6).
38 | IsLinked(9, 6).
39 | IsReachable(1).
40 | IsReachable(2).
41 | IsReachable(3).
42 | IsReachable(4).
43 | IsReachable(5).
44 | IsReachable(6).
45 | IsReachable(7).
46 | IsReachable(8).
47 | IsReachable(9).
48 | 1
49 | 2
50 | Happens(Stop(6), 2).
51 | 3
52 | -IsAlive(6).
53 | Happens(Isolate(8), 3).
54 | Happens(Isolate(9), 3).
55 | 4
56 | -IsReachable(8).
57 | -IsReachable(9).
58 | Happens(Disconnect(7, 4), 4).
59 | 5
60 | -IsLinked(7, 4).
61 | Happens(Isolate(7), 5).
```

```
62 6
63 -IsReachable(7).
64 7
65 Happens(Stop(2), 7).
66 8
67 -IsAlive(2).
68 Happens(Isolate(5), 8).
69 9
70 -IsReachable(5).
71 10
72 P
```

To simplify the adoption of the specification and thus the performing of an experiment considering a WSN with its topology and a particular event trace, in the chapter 5 we are going to present a support tool designed to automatically generate the structural specifications, given the target WSN topology, perform reasoning, by means of the DECReasoner, starting from the correctness and structural specifications and from an initial event trace (i.e., to perform the *What-if* analysis of a WSN), and finally compute dependability metrics, such as connection resiliency, coverage, data delivery resiliency and power consumption, starting from the outcome produced by the reasoner.

Chapter 4

Dynamic Verification

To evaluate the WSN dependability at design time is necessary in order to increase the confidence about the robustness of the designed solution before putting it into operation; but it is also necessary to monitor the system during operation in order to avoid unexpected results or dangerous effects. Runtime Verification is a good solution to perform this type of verification and despite in literature there already are tools and languages (i.e. Reactive Event Calculus) that are able to perform runtime verification, our aim is to demonstrate that we can perform both verification techniques (static and runtime) exploiting the same sets of specifications (general and structural).

This chapter presents a rationale of the dynamic verification technique of WSN based systems. In particular we present the Runtime Verification technique and propose the design of a reliable WSN-based monitoring system based on dependable monitoring services and on the configurable and the automatic deployment of system monitors. Finally an application scenario concludes the chapter describing this type of verification in practice.

4.1 Rationale of the dynamic verification technique

Since WSNs are used in critical scenarios (Ambient Assisted Living, Home Monitoring, Health Monitoring, ...), it is not sufficient to verify them only at design time but it is also necessary at runtime since even if the system has been verified during its design, it needs to be monitored during all his life to detect unexpected differences and avoid dangerous effects. In other terms, it is needed to perform a *continuous monitoring* in order to check the dependability behavior even when the WSN is operating.

In the Chapter 3 we have defined a set of general correctness specifications and a set of

structural specifications dependent on a target WSN; after these definitions we have presented a static verification technique.

In this chapter we want to show that it is possible also to perform dynamic verification exploiting the same specifications defined for the static verification. The aim of the dynamic verification is not only to detect the criticalities of the WSN in the current state but it also is to perform a prediction of the possible future criticalities that may occur in the WSN.

To monitor a WSN-based system at runtime we can adopt the *Runtime Verification* (RV) technique.

RV is a technique that is dedicated to studying, to developing and to implementing of solutions that verify whether correctness properties defined for a system are met or violated.

In particular among the RV techniques there is one that is based on the Event Calculus formalism: the *Reactive Event Calculus* (REC). Though REC could be a valid solution for our aim, we chose to not consider it since it would have implied the using of a different tool, and to re-adapt all the specifications for Prolog since the current REC tool is based on Prolog.

Moreover, since the events, defined in the previous chapter (i.e. *Start*, *Stop*, *Send*, etc...) are not simulated anymore but they have to be detected from the system, in this chapter we also propose the design of a reliable WSN-based monitoring system based on dependable monitoring services and on the configurable and the automatic deployment of *system monitors* with the task of catching events to consider to perform reasoning for the dynamic verification. For example, if a node fails in a WSN while it is running, this failure event is detected by a system monitor; by means of a reasoning, on the basis of the received event,

dependability metrics are calculated both for the current status of the WSN and to assess the criticality of the network, in order to alert the user about possible future hazardous scenarios considering the new network conditions.

4.2 Runtime Verification

Runtime verification [121] is the discipline of computer science that deals with the study, development, and application of verification techniques that allow checking whether a run of a system under scrutiny satisfies or violates a given correctness property [87]. The field of runtime verification arose from the desire to check the correctness of complex programs without needing to check every possible execution of the software.

The goal of runtime verification [13] is to determine, at every time step, if the system is currently meeting its correctness requirement (in our case, dependability attributes); a description of system correctness is a set of formally specified, high-level and time-evolved behaviors that have been determined to be necessary for correct system operation. It can be used to automatically evaluate test runs, either on-line, or off-line analyzing stored execution traces; or it can be employed on-line, during the operation of the system, potentially steering the application back to a safety region if a property is violated.

In detail, runtime verification enables the checking of correctness properties with respect to system implementations [48]. It concerns the application of a lightweight formal verification during the execution of the system by checking traces of events generated from the system run against the correctness properties. When a property is violated, a recovery strategy

is triggered. The technique scales well since just one model of computation is considered, rather than the entire state space like in model-checking [42].

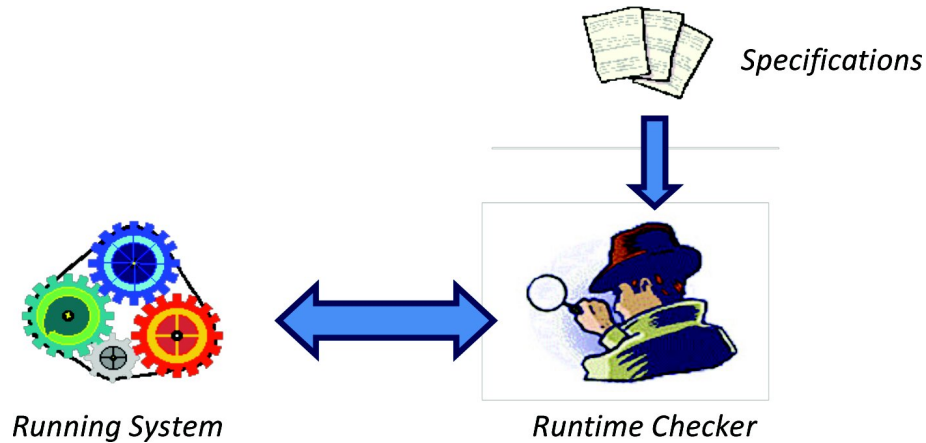


Figure 4.1: Running System with support of the Runtime Checker

The Runtime verification may become a major verification technique especially for systems in which the behavior depends heavily on the environment and operational conditions [33]. In other words, the behavior of highly dynamic systems such as adaptive, self-organizing, self-healing, or pervasive systems, depends heavily on the environment and changes over time, which makes their behavior hard to predict and analyze prior the execution. To ensure certain correctness properties, the runtime verification can become part of the architecture of dynamic systems, independently from the specific architectural model adopted. Figure 4.1 shows an example of use of a runtime checker: while the system is running, the Runtime Checker checks if the system is currently meeting its correctness properties (the specification).

4.2.1 Reactive Event Calculus

The *Reactive Event Calculus* (REC) is a *reactive* version of the logic-based Event Calculus language [28]. In the REC fluents are initiated and terminated by dynamically occurring events.

A REC based system everytime receives a new event (or more events), it *reacts* by calculating the new sequence of events (the *narrative*) and by consequently extending the obtained result to the previously computed result so to have the complete sequence of events.

The axiomatization of the REC is performed in accord to the *SCIFF Abductive Logic* [7] and it is fully declarative axiomatization: operational specifications are not necessary.

The most common tool used to perform specification defined in REC language is the jREC [102]. It is a JAVA+Prolog-based tool for reasoning upon the dynamics of an event-based system with respect to an Event Calculus specification. More specifically, jREC dynamically acquires the event occurrences characterizing a running execution of the system and monitors their effects by updating the evolution of the corresponding fluents.

Despite REC is a valid solution, we chose to not use it since it would have implied the using of a different tool, and to re-adapt all the specifications for Prolog (REC tool is based on Prolog); for this reason REC is not suitable for our aim.

4.3 Event Capture in a Dynamic Verification context

In this section we present the monitoring services conceived to automatically detect undesired events and potentially react to them by means of RV. The services have been proposed

in [34] and they are discussed with respect to a reference WSN-based monitoring system, depicted in figure 4.2. First we present the services offered to applications, and their role. Then, we introduce the concept of monitor and describe the monitor components introduced in the system.

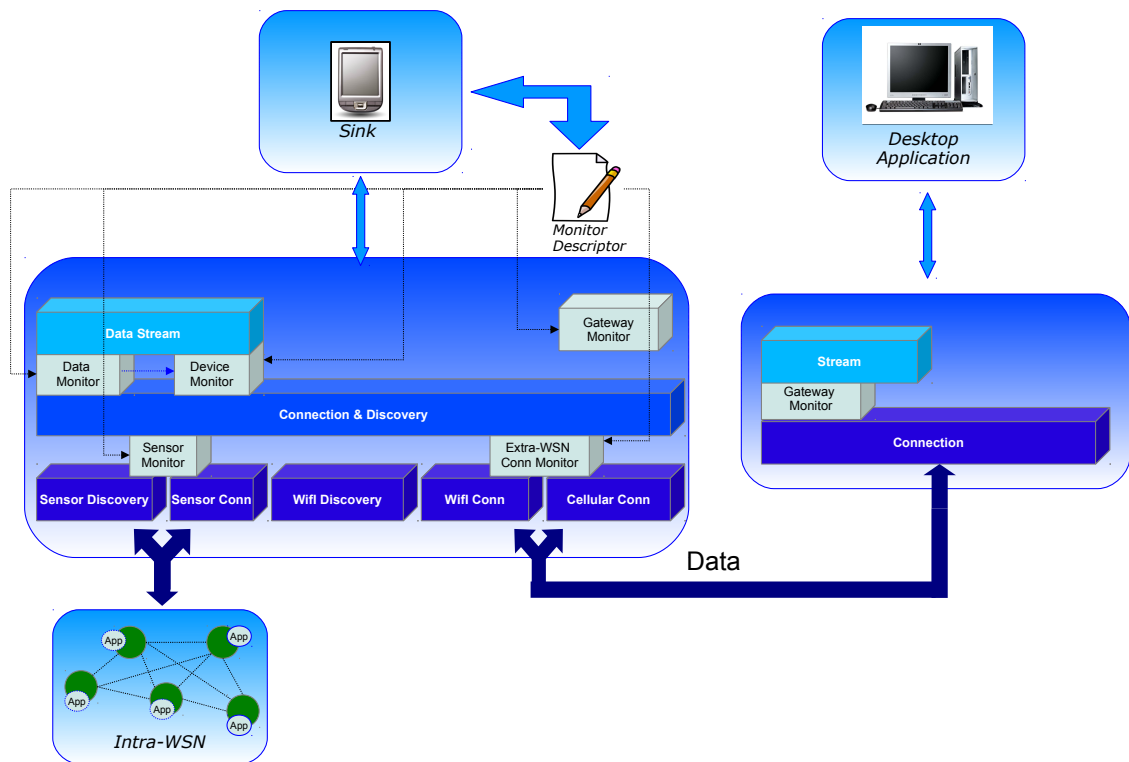


Figure 4.2: WSN-based system with event monitors

The architecture of the monitoring system is shown in 4.2 and it is structured in four main parts: *Intra-WSN*, *Gateway Services*, *Remote Center Services* and *External Applications*.

4.3.1 Intra-WSN

The Intra-WSN is a particular network constituted by a set of sensor nodes that communicate among each other and with the gateway. Every node is provided of an running application that is in charge of to collect all the data of interest and send them to the sink.

4.3.2 Gateway Services

The services offered on the Gateway side (sink) are summarized in the following:

- *Sensor Connection*: this service provides the communication between a sensor of the Intra-WSN and the gateway device (i.e. a PDA, a notebook, etc...);
- *Sensor Discovery*: this service is used to discover the sensors in the Intra-WSN;
- *Wifi Discovery*: this service is used to verify if there is an access point for the WiFi connection;
- *Wifi Connection*: this service provides the WiFi communication to transmit the data stream to the remote center;
- *Cellular Connection*: this service provides the cellular communication (GPRS, UMTS, etc...) to transmit the data stream to the remote center when a WiFi access point is not available;
- *Connection & Discovery*: this service is placed at an upper layer and it has the role to hide to applications the details on the communication technology (Bluetooth, WiFi

and cellular). Further, it provides technology-agnostic discovery services, which are then specialized for Bluetooth and WiFi;

- *Data Stream*: it provides streaming services for sensed data to the application.

4.3.3 Remote Center Services

The services offered for the Remote Center side (desktop) are:

- *Connection*: this service provides the needed communication interface to receive data from the Gateway and to send commands to manage the monitoring;
- *Stream*: it is the service that dialogues with the desktop application of the Remote Center. It reports the data acquired by the sensors.

4.3.4 External Applications

Finally, we have to consider external application for both side (mobile and desktop). These applications generally include the GUI used by users on the mobile side (e.g. patients in case of health monitoring) and by the operators on the fixed side (e.g. foresters in case of environmental monitoring), and implement application specific data interpretation and reporting functions.

4.3.5 System Monitors

Once introduced the general services of the WSN-based monitoring system, we enrich the system with event capture capabilities. To this aim, we introduce the concept of *System*

Monitors [34].

In order to start the runtime verification, a system monitor has to be able to detect basic events that we report in the table 4.1 and that have been presented in the chapter 3.

Table 4.1: Basic events detected by the system monitors

Event	Description
<i>Start</i>	it occurs when a sensor turns on
<i>Stop</i>	it occurs when a sensor turns off
<i>Connect</i>	it occurs when a sensor connects to another sensor
<i>Disconnect</i>	it occurs when a sensor disconnects from another sensor
<i>Send</i>	it occurs when a sensor starts the delivery of a packet towards the sink node

A monitor is a service instantiated on-demand on the basis of the events that have to be detected. By means of a *Monitor Descriptor file* (i.e. a XML file), the developer can set the events that he wants to observe. This file is provided to the Sink that, in turn, dynamically creates the requested monitors, which run in the background and are managed transparently from developers.

Specifically, first we introduce the following monitors that are used for detection of the considered basic events:

- *App Monitor*

It is deployed on the sensor node and it detects all of the events that occur inside. This monitor can be considered as a event logger for the node, since it can store its state before of the failure event, and report it to the gateway when the device is recovered. Its support is very important for detection of *Send* and *Disconnect* (or *Connect*)

events. When this monitor notices of packet being sent from a node then it detects the *Send* event; when it does not receive anymore any packet from a neighboring node then it detects the *Disconnect* event. Both events are sent to the *Sensor Monitor* by means of the *Sensor Connection* service. On the basis of packet loss event specification, the combination of *Send* and *Disconnection* events can generate a *Packet Loss* event (see section 3.2.2).

- *Sensor Monitor*

It is deployed on top of the *Sensor Discovery* and *Sensor Connection* services. *Sensor* and *App Monitor* are very essential for detecting all basic events. This monitor, in addition to receive the detected event from the *App Monitor*, is able to detect *Stop* (or *Start*) event. Let us suppose that every node in the WSN periodically sends a packet (e.g. one packet every hour); if the system monitor observes that in a certain time interval does not receive any packet from sensor x then it detects a *Stop(x)* event. On the contrary, if starts again to receive packets from a node x that was stopped then it detects a *Start(x)* event.

Therefore Sensor and App monitors are helpful to detect the basic events we have defined and that are summarized in table 4.1.

Then we briefly present the other event monitors present in the WSN-based system; they aim to detect the failure events that have been presented in the FMEA study (see section 1.4):

- *Extra-WSN Monitor*

checks the availability of the WiFi and Cellular connections. It can be efficiently used to manage the hand off process between the two technologies.

- *Data Monitor* checks if there are anomalies in the data stream acquired.
- *Device Monitor* detects mainly if a sensor node is unavailable. In this case, to better analyze the event it is necessary to require extra information to the Sensor Monitor.
- *Gateway Monitor* is present both in the Gateway and in the Remote Center. The aim of this monitor is to check if the gateway operates correctly. The Gateway Monitor in the Remote Center can only detect if the gateway becomes unavailable but it cannot know the cause. Instead the Gateway Monitor in the sink side, can keep track of occurred failure events, such as a freeze, self-shutdown, etc, following for instance the logging approach proposed in [38].

4.4 Application Scenario

In this paragraph we want to show the dynamic verification technique by means of an application scenario; the aim is to describe how it is possible to catch events and observe their effects in a WSN at runtime.

In the figure 4.3, from the left to right, we can see when a event occurs in a wireless sensor node, by means of a wireless communication, it is detected by a system monitor that runs on a gateway. The failure event (for instance $Stop(n)$) is managed by the monitor and

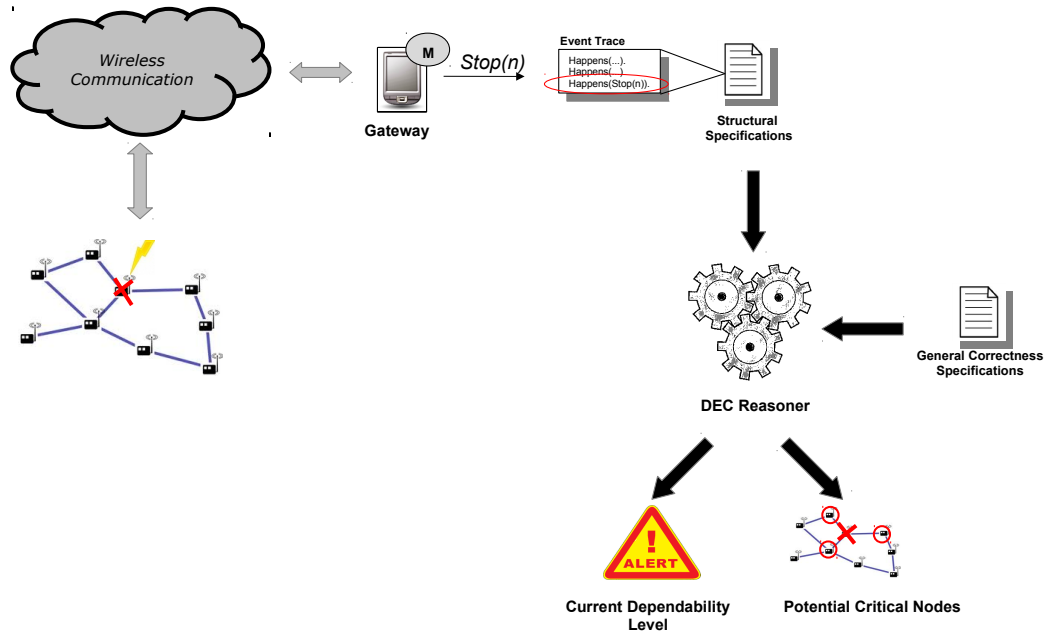


Figure 4.3: Application scenario in dynamic context

added to the current event trace to perform the reasoning.

The new event trace is included in the structural specification; thus the DECReasoner receives the structural specifications with the last occurred event and considering the general correctness specifications performs the reasoning returning a couple of outcomes: i) the *current dependability level* of the WSN and ii) the *potential critical nodes*.

The former informs the user about the WSN dependability at current state calculating the current metrics (current coverage, current connection resiliency, current data delivery resiliency and current power consumption). The latter returns a prediction about possible

criticalities that may occur after the current event.

Moreover in the dynamic verification we have to take in account the changes of the WSN topology: if a sensor fails, all the packets delivered to it have to be sent to another sensor and thus the topology, defined in Event Calculus through the predicate *Neighbor*, has to be updated at runtime. The dynamic verification technique supports this feature.

The typical components of this scenario are: the *Running System* (the WSN), the *Runtime Detector* (the system monitor that runs on the gateway and the DECRReasoner) and the *Recovery Process* (the outcomes that inform about criticalities of the WSN).

The engineer is the key figure in the material progress of the world; he translates scientific knowledge into tools, resources, energy and labor to bring them into the service of man.

Sir Eric Ashby

Chapter 5

The ADVISES Tool

In this chapter we propose a framework to investigate the correctness of the design of a WSN from the point of view of its dependability, i.e., resilience to undesired events. The framework, named ADVISES tool, is based on the Event Calculus formalism and it is aimed to simplify the adoption of our approach by network maintainers. The ADVISES tool allows to specify the target WSN in a user-friendly way and it is able to generate automatically the Event Calculus specifications used to check correctness properties and evaluate dependability metrics, such as connection resiliency, coverage and lifetime. It is able to work at design and runtime. In particular at runtime the ADVISES tool is like a server that is in waiting for new events coming from the WSN and, performed the reasoning using the same specifications, is able to do prediction about future criticalities of the WSN.

5.1 Introduction

A Java-based tool, called *ADVISES* (**A**utomate**D**Ver**I**fication of **w**S**n** with **E**vent calculu**S**), has been designed and implemented to facilitate the application of the two proposed verification techniques (static and dynamic).

The goal of the ADVISES tool is to automate the instrumentation of the Event Calculus scripts to obtain specification for any network.

The ADVISES tool is able to work in both modes: at design time and at runtime; in particular in the last case it is like a server that is in waiting for new events coming from the WSN and that are detected by means of a system monitor. The main aim of the ADVISES

tool is to automatize all the steps that have been considered during the description of the application scenarios in static and dynamic verification discussed in chapters 3 and 4.

A workflow is presented to show the functioning of the ADVISES tool: from the creation of the structural specification file for a target WSN to the computation of the dependability metrics and the sending of information/warning messages.

Next we present the *Graphic User Interface* (GUI) of the ADVISES tool realized for performing static verification and another one for performing dynamic verification: in the static we need to select several parameters that in dynamic are not necessary.

5.2 Workflow

In this paragraph we explain the workflow that describes the operation of the ADVISES tool realized using the Java programming language, the Event Calculus as a formal language and the DECRReasoner to produce the narrative.

Let us observe the workflow represented in figure 5.1 to better understand how the ADVISES tool works.

Initially the *Pre-processing* block receives the operator's inputs (see table 5.1) and settings, many of which can be chosen only at design time since at runtime the most of information comes from the monitored WSN, (i.e. the disconnection probability for every link of the WSN, the initial battery levels of each node and the initial event trace in case of What-if scenario). By means of this block the ADVISES tool is able to automatically generate the *Structural Specification* file (a file with ".e" extension) with initial conditions in terms of

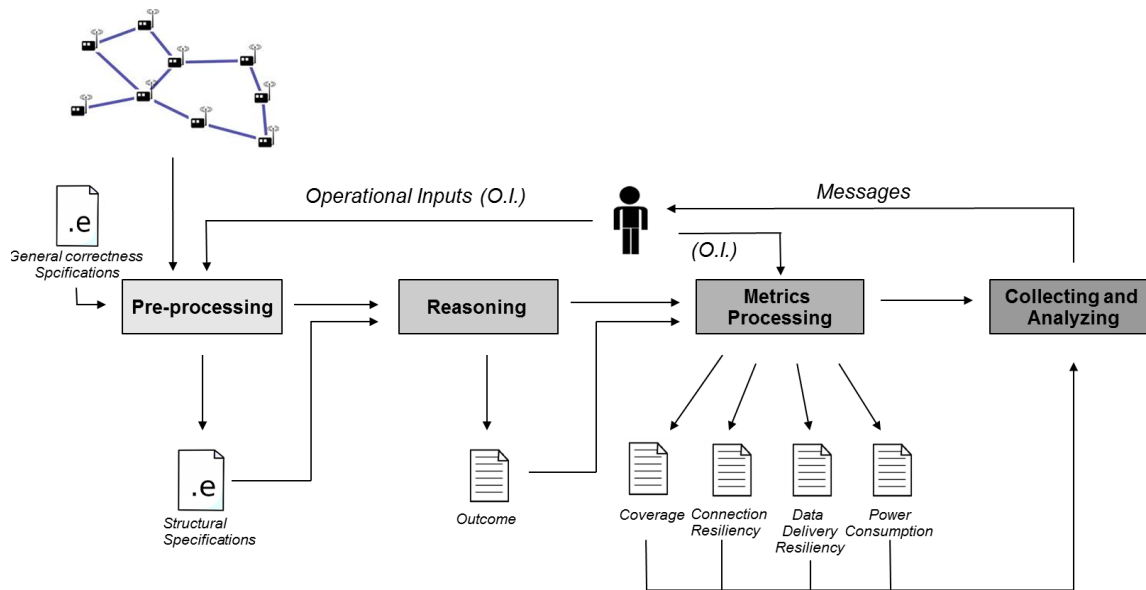


Figure 5.1: Workflow

Event Calculus formalism.

The general correctness specifications are included in the structural specification file which is then provided by the ADVISES tool as input to the DECReasoner, which produces the outcome as the result of the reasoning (*Reasoning* block). The DECReasoner generates the output (the *narrative*) in a single textual file (*Outcome*) or in a set of them, when we perform Robustness Checking.

The *Metric Processing* block analyzes the outcome produced by the previous block and calculates the coverage, the connection resiliency, the data delivery resiliency and the power consumption. The metrics are calculated on the basis of their definitions and considering the fluent values produced by the DECReasoner at each timepoint for each model.

Finally the ADVISES tool sends messages to the operator in order to inform or alert him

Table 5.1: Tool Inputs

Input	Description
<i>Topology File</i>	A file with *.txt extension (textual file). It is a topology file in which links between nodes are represented.
<i>Number of timepoints</i>	User has to indicate the observation time of behavior of the considered WSN. It is a positive integer number.
<i>Number of sensors</i>	User has to indicate the number of the sensors that compose the WSN topology. It is a positive integer number.
<i>Number of packets</i>	It is the number of packets that every node can send to the sink node. It is a positive integer number.
<i>Number of failures</i>	It is the number of failures which the user considers in order to know the robustness level of the WSN. It is a positive integer number.
<i>Battery capacity</i>	It is the maximum battery capacity of a sensor expressed in Joule. It is a positive integer number.
<i>TX/RX energy</i>	It is the energy expressed in μ Joule necessary for every node to perform a TX/RX operation. It is a positive integer number.
<i>Threshold</i>	It is the threshold value associated to the coverage and data delivery resiliency metrics. It is a percentage value within the range 0-100.

about WSN robustness. In particular in case of dynamic verification, the ADVISES tool is able to make prediction about possible future critical nodes and thus to suggest operator to make some change at the WSN topology.

5.3 ADVISES Tool GUI

Since the static and dynamic verification do not require the same number of input parameters, we present a GUI for static verification and another GUI for dynamic verification.

The ADVISES tool GUI for static verification is rich and it needs several parameters like the number of packets, the number of failures (for performing the robustness checking at

design time), the battery capacity value of a node, the initial event trace, etc.

The ADVISES tool GUI for dynamic verification is simpler because it works on the basis of the events that receives from the WSN.

5.3.1 ADVISES Tool for Static Verification

By means of the GUI shown in figure 5.2, an user can simply specify i) the topology of the target WSN, using a connectivity matrix, ii) the formal correctness specifications (for checking isolation events, packet losses and battery exhaustion), iii) the temporal window size to consider, in terms of the number of timepoints, iv) the number of packets that each sensor can send, v) the number of failures (in case of coverage robustness checking), vi) the battery capacity of a sensor (in Joule) and the needed energy for RX/TX operations (in μ Joule), vii) the metrics to calculate (for coverage and data delivery resiliency it is necessary also the threshold value), viii) the channel model (in case of data delivery robustness checking), ix) the initial battery level, x) the initial event trace (to perform what-if scenario analysis).

The GUI is subdivided by five panels.

In the first panel there is a message that reports the application title. The *Topology* section is dedicated at the loading/creating of a network topology. The user has to select a topology file that can be found by means of *Open* button or created on-the-fly using the *New topology* button. *Show topology* button allows to display the selected topology in a separate frame.

In this file links between nodes are represented.

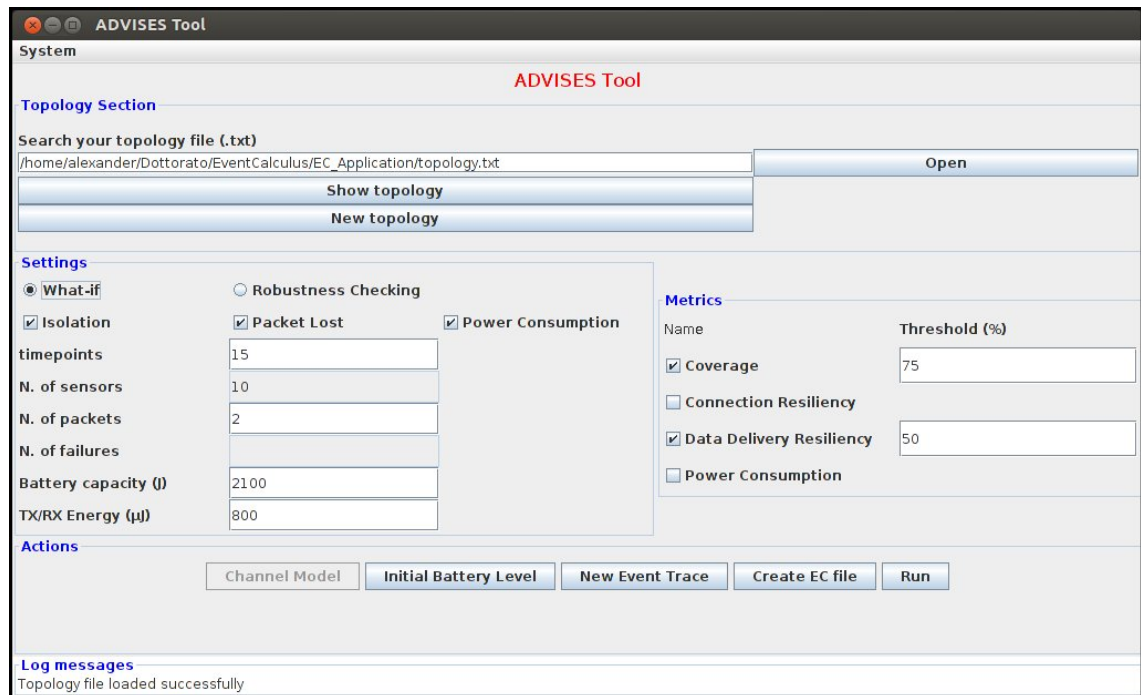


Figure 5.2: ADVISES GUI at design time

Selected the topology, let us analyze the *Settings* section panel. By means of this panel the user can set the several parameters (numbers of timepoint, of packets, of failures, battery capacity and TX/RX energy) that will be considered for the static verification; the number of sensors is automatically calculated by the ADVISES tool. Moreover the user can choose what general correctness specifications (*Isolation*, *Packet Loss* and *Power Consumption*) are necessary for a particular verification and whether to perform a what-if analysis or a robustness checking: this option helps to avoid that user can do wrong choices (e.g. insertion of the number of failures in case of what-if analysis or choice of an initial event trace in case of robustness checking).

In the *Metrics* section, the user with a tick chooses the desired metrics and if he selects

coverage or data delivery resiliency he has to write the percentage values (default value is 50).

In the fourth panel (*Actions*) the ADVISES tool presents the possible steps that user can perform.

- *Channel Model*

Pressing this button the users can select the disconnection probability for every link of the WSN topology choosing a percentage value (0%=never disconnection; 100%=ever disconnection). This action is disabled if user wants to perform What-if analysis or robustness checking but without considering the packet loss. In fact the channel model is defined to randomly generate disconnection failures in order to verify the packet loss event.

Figure 5.3a shows an example of the dialog associated with this button.

- *Initial Battery Level*

Pressing this button the users can set the initial battery level for every node of the WSN topology choosing a percentage value (0%=completely discharged node; 100%=completely charged node). This action is disabled if user wants to perform experiments without considering the *Power Consumption* specification in *Settings* section.

Figure 5.3b shows an example of the dialog associated with this button.

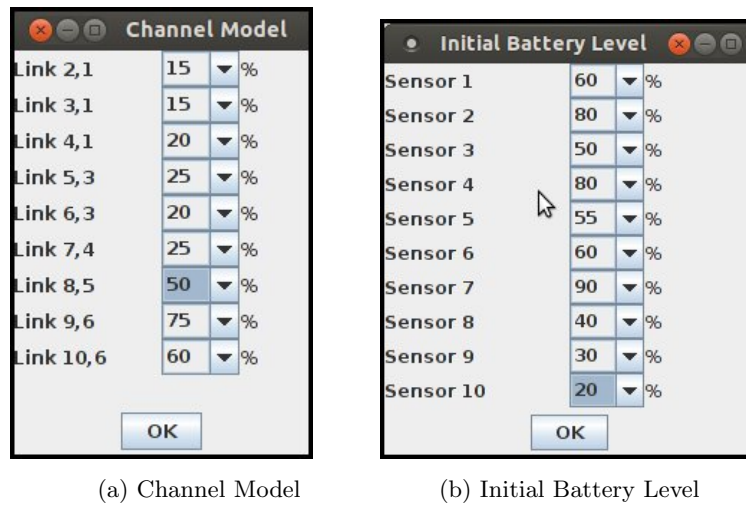


Figure 5.3: Channel Model and Initial Battery level frames

- *New Event Trace*

Pressing this button the users can set the initial event trace that could occur in the WSN. This action is disabled if user wants to perform robustness checking: the initial event trace is considered only in case of What-if analysis.

Figure 5.4 shows an example of dialog associated with the *New Trace Event* button; note that the dialog is so smart to guide the user in the adding new events avoiding wrong values (i.e. a link for a *Start* or *Stop* event or a node for a *Disconnection* event).

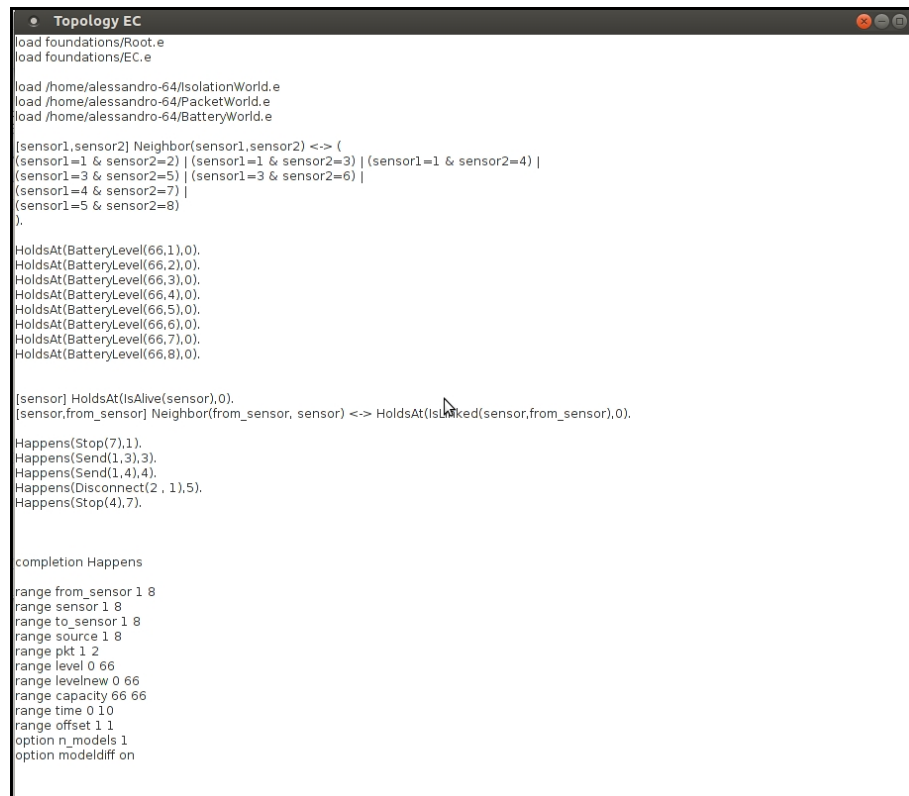
Timepoint	Event	Node	Packet
1	Send	10	1
3	Send	7	1
4	Disconnect	2, 1	-
5	Send	10	2
7	Stop	5	-
9	Connect	2, 1	-

Figure 5.4: Example of an Initial Event Trace specified by the user

- *Create EC file*

Pressing this button the ADVISES tool automatically generates an Event Calculus file in the same directory of the topology file. This file wraps the general correctness specifications chosen in the *Settings* section and the structural specifications. Once EC file is created, it is displayed in a separate frame.

Figure 5.5 shows an example of the frame generated by the ADVISES tool.



```

Topology EC
load foundations/Root.e
load foundations/EC.e

load /home/alessandro-64/IsolationWorld.e
load /home/alessandro-64/PacketWorld.e
load /home/alessandro-64/BatteryWorld.e

[sensor1,sensor2] Neighbor(sensor1,sensor2) <-> (
(sensor1=1 & sensor2=2) | (sensor1=1 & sensor2=3) | (sensor1=1 & sensor2=4) |
(sensor1=3 & sensor2=5) | (sensor1=3 & sensor2=6) |
(sensor1=4 & sensor2=7) |
(sensor1=5 & sensor2=8)
).

HoldsAt(BatteryLevel(66,1),0).
HoldsAt(BatteryLevel(66,2),0).
HoldsAt(BatteryLevel(66,3),0).
HoldsAt(BatteryLevel(66,4),0).
HoldsAt(BatteryLevel(66,5),0).
HoldsAt(BatteryLevel(66,6),0).
HoldsAt(BatteryLevel(66,7),0).
HoldsAt(BatteryLevel(66,8),0).

[sensor] HoldsAt(IsAlive(sensor),0).
[sensor,from_sensor] Neighbor(from_sensor, sensor) <-> HoldsAt(IsLinked(sensor,from_sensor),0).

Happens(Stop(7),1).
Happens(Send(1,3),3).
Happens(Send(1,4),4).
Happens(Disconnect(2,1),5).
Happens(Stop(4),7).

completion Happens

range from_sensor 1 8
range sensor 1 8
range to_sensor 1 8
range source 1 8
range pkt 1 2
range level 0 66
range levelnew 0 66
range capacity 66 66
range time 0 10
range offset 1 1
option n_models 1
option modeldiff on

```

Figure 5.5: Example of a EC file generated by the ADVISES tool

- *Run*

To obtain the output of the DECReasoner the user has to press this button; analyzing the fluent values contained in the outcome produced by the DECReasoner, the ADVISES tool computes the desired dependability metrics. A pop-up message will appear on the screen in order to notify user the end of the computation.

Finally in the *Log messages* panel, the ADVISES tool reports all the useful messages in order to perform the user if some error occurs.

5.3.2 ADVISES Tool for Dynamic Verification

Ultimated the design phase of a WSN, however it is necessary to monitor the robustness of the network at runtime. Before to present the GUI for dynamic verification, it is necessary to know how the ADVISES tool can work at runtime.

Figure 5.6 shows an application schema.

When a failure occurs in a wireless sensor node, it is detected, by means of a wireless communication, by a system monitor that runs on a gateway. The event failure (for instance $Stop(n)$) is managed by the monitor and sent to the ADVISES Tool that is in listening on a port ready to receive events and start reasoning.

The ADVISES tool, received the output from the DECReasoner, calculates the selected dependability metrics to establish both the current status of the WSN and the robustness and therefore a possible reaction in the case of further failures.

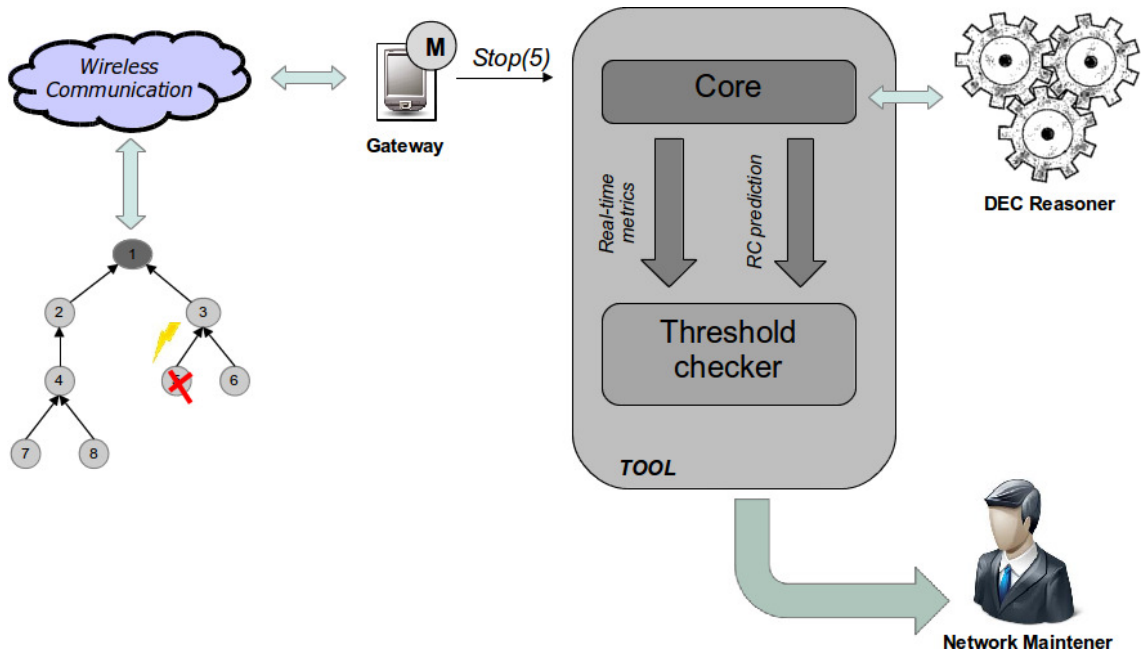


Figure 5.6: Use of the ADVISES Tool at runtime

Then, verified the obtained metrics values with the thresholds set by the user, the ADVISES tool sends messages to network maintainer which may be purely informative or alerts in case these values are under the desired threshold.

Therefore the ADVISES tool “*advises*” the network maintainer of problems related to the network and reports its critical points.

The flow chart in figure 5.7 describes the ADVISES operating at runtime. Once started, the ADVISES Tool is in *server mode* and in waiting for events coming from the WSN (*Waiting for events*); if a new event is detected (*New detected event?*) then it receives this event (*Receiving event*) and automatically updates the sequence (*Updating event sequence*) of received events and generates the EC file in order to perform the reasoning with the DECReasoner (*Reasoning*). Finally it calculates the current metrics and performs robustness

checking to predict the future criticalities (*Current metrics and RC prediction*). After the last step, the ADVISES tool continues to work waiting next detected events.

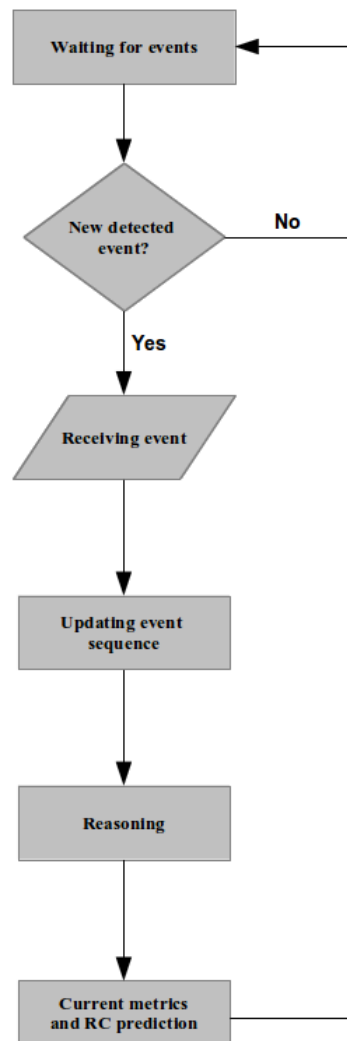


Figure 5.7: Flow chart of the ADVISES operating at runtime

For this purpose we have realized another GUI of the ADVISES Tool in order to receive events from a WSN in real-time detected through a system monitor and to start the runtime verification evaluating both current and future criticalities.

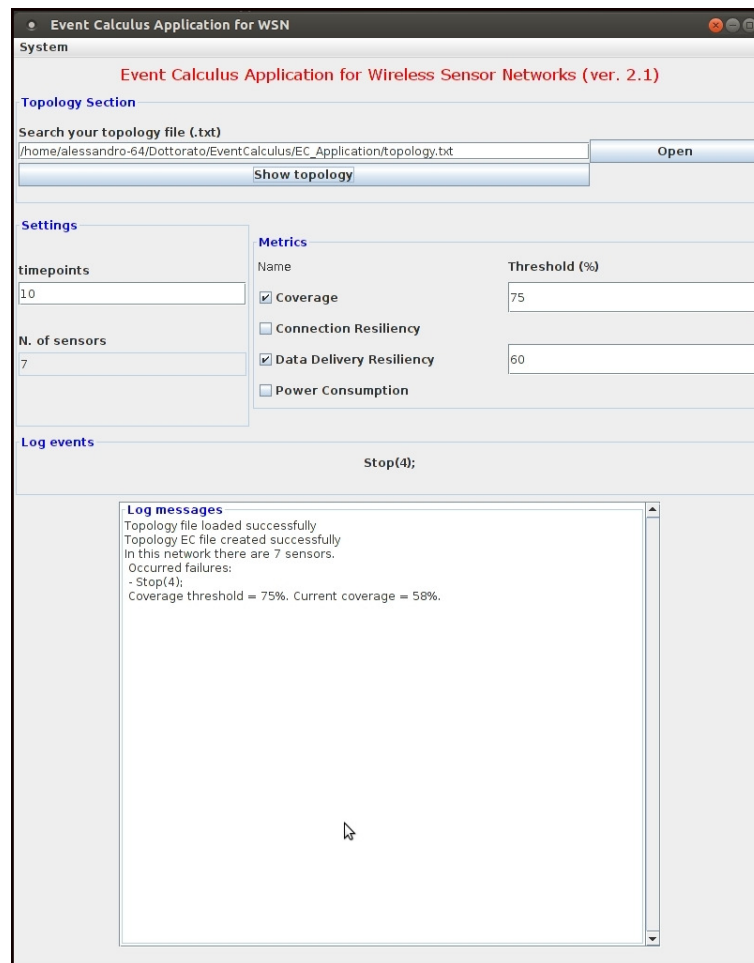


Figure 5.8: ADVISES GUI at runtime

Figure 5.8 shows this GUI.

By means of this GUI, a user can simply specify i) the topology of the target WSN, using a connectivity matrix, ii) the temporal window size to consider, in terms of the number of timepoints, iii) the metrics to calculate (for coverage and data delivery resiliency it is necessary also the threshold value).

Even this GUI is subdivided by five panels but with a different goal; for example the *Actions* section disappears and replaced by the *Log events* panel since user does not have to choose

anything but only to observing the behavior of the WSN.

The first panel is identical to the GUI for static verification. Before to start the dynamic verification, the user has to set the topology of the WSN to monitor.

Selected the topology, let us analyze the *Settings* panel. In dynamic verification the number of sensors is automatically calculated by the ADVISES tool and the user has just to set the number of timepoints to consider: in this context this value represents the time range for observing the effects of the current event.

The *Metrics* panel is identical to the GUI for static verification. In the *Log events* panel the ADVISES tool reports all the events coming from the WSN and that have been detected by the system monitor.

Finally in the *Log messages* panel, the ADVISES tool reports all the performed computations about current state of the WSN and the risks that may affect its robustness. Also there are useful messages in order to perform the user if some error occurs.

5.4 Metrics computation

By means of a parser that analyzes the traces produced by the DECReasoner, the ADVISES tool calculates the dependability metrics.

The computation of *Coverage* and *Connection Resiliency* depends on a threshold parameter, to be indicated as a percentage by the user in the GUI (see the “Threshold” text field in the Metrics section in figure 5.2). The threshold expresses the fraction of failed and isolated nodes that the user can tolerate, given its design constraints. For instance, over a WSN

of 20 nodes, a threshold set to 100% means that all the 20 nodes have to be connected, whereas 50% means that the user can tolerate at most 10 isolated nodes.

Considering the threshold value, we calculate the Coverage analyzing the *IsReachable(sensor)* and *IsAlive(sensor)* fluents found to be true in the event trace produced by the reasoner: if a *-IsReachable(x)* or a *-IsAlive(sensor)* fluent is true in the event trace, this means that node x became isolated or it stopped. For example in the case of coverage at 50%, for a WSN with 7 nodes, there is coverage when at least 4 nodes are not isolated (i.e., they are reachable). Hence, as soon as 4 different nodes are no reachable nor alive (looking at the fluents), the network is not covered anymore. The coverage can be then evaluated as the interval $[0, t]$, being t the timepoint of the last failure or disconnection event before the isolation (e.g., the timepoint of the event that caused the isolation of a number of nodes exceeding the threshold).

The Connection Resiliency can then be evaluated as the number of failure and disconnection events (namely, *Stop(sensor)* and *Disconnect(sensor, from_sensor)* events) that happen within the coverage interval, excluding the last failure/disconnection event, that is, the one that actually leads the number of isolated nodes to overcome the threshold. For example, if we have coverage in the interval $[0, 6]$, and during this period 3 failure/disconnection events can be counted, than the Connection Resiliency is 2, that is, the WSN was able to tolerate 2 failures or disconnections while preserving more than 50% of the nodes connected.

Even the computation of the *Data Delivery Resiliency* depends on a threshold parameter indicated by the user as the percentage value. By means of this percentage value, the user can express the fraction of the number of packets that can be lost. For instance, over a

WSN of 30 nodes, a sensor sends 10 packets; if this threshold is equal to 60%, it means that at most 4 packets can be lost. Considering the threshold value, the ADVISES tool calculates the data delivery resiliency analyzing the $IsInDelivery(pkt,source)$ fluent in the event trace produced by the reasoner: if a $+IsInDelivery(pkt,source)$ fluent is true in the event trace, this means that a packet has been sent by a sensor; if a $-IsInDelivery(pkt,source)$ fluent is true, this means that a packet has been received by the sink node. If the number of instances where the fluent $-IsInDelivery(pkt,source)$ is equal to the number of instances where the fluent $+IsInDelivery(pkt,source)$, then every packet has been correctly received by the sink node, otherwise this means that some packet is lost. As soon as the fraction of lost packets becomes lower than the threshold value then the ADVISES tool verifies the number of failures (Stop and Disconnect events) occurred; the number of the occurred failures represents the value of the data delivery resiliency.

Power consumption is computed analyzing the $BatteryLevel(level,sensor)$ fluent. For each timepoint, if a $+BatteryLevel(level,sensor)$ fluent is true then the ADVISES tool updates the battery consumption of the related sensor. After the last timepoint, for each sensor the ADVISES tool computes the percentage of power consumption. If the behavior of nodes is known (e.g., each node send packets periodically, with a known period), this information can be used to evaluate the lifetime of the network as the time when the number of alive and not isolated nodes falls below the coverage threshold.

There are three principal means of acquiring knowledge: observation of nature, reflection, and experimentation. Observation collects facts; reflection combines them; experimentation verifies the result of that combination.

Denis Diderot

Chapter 6

Case Studies

In this chapter we report the results from five representative case studies. The first two case studies are based on what-if scenario analysis, and their aim is to evaluate if the reasoning performed by the DECReasoner on our specifications is correct. They also aim to show how the ADVISES tool can be used to verify the behavior of a WSN in a precise scenario at design time. The third and fourth are based on robustness checking. In these cases, the aim is to show how the proposed approach can be helpful to analyze a design and drive engineers' choices. Finally the last shows how the ADVISES tool works at runtime.

6.1 Case study 1 (what-if analysis): A Wireless Body Sensor Network

6.1.1 Scenario

As first case study for what-if analysis let us consider a wireless body sensor network realized by Quwaider et al. in [119] shown in figure 6.1a. This Wireless Body Sensor Network (WBSN) is constructed by mounting seven sensor nodes attached on two ankles, two thighs, two upper-arms and one in the waist area. Each node consists of a 900 MHz Mica2Dot MOTE (running Tiny-OS operating system).

In figure 6.1b it is reported the related node tree graph in which the arrows indicate the relationship between a couple of nodes (i.e. node 2 depends on node 1, node 3 and 4 depend

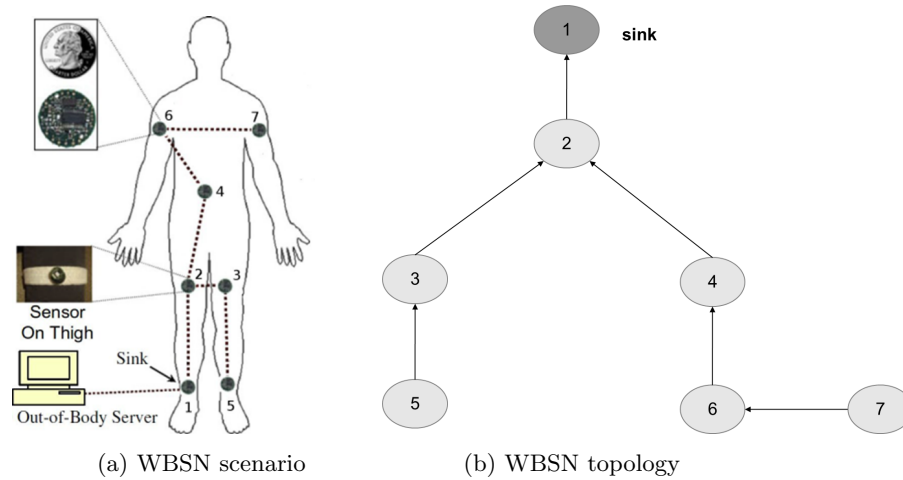


Figure 6.1: WBSN case study

on node 2, etc.). We only have to load the topology file in the GUI, to allow the axioms defined for correctness properties to be interpreted and automatically created for the given topology.

We suppose that the following basic events occur: $Disconnect(5, 3)$ at timepoint 1 and $Stop(4)$ at timepoint 3. For a coverage threshold set at 50%, we should observe a coverage interval equals to $[0, 3]$ (i.e., when node 4 stops at time point 3, 4 nodes are not reachable, namely 4, 5, 6 and 7), and a connection resiliency equals to 1 (i.e., only the disconnection event is tolerated).

6.1.2 Results

Figure 6.2 shows the loading of the topology for this case study. To perform the analysis, we charge an initial event trace in the ADVISES tool (figure 6.3); the user has to select the timepoint, the event and the node (in case of $Start/Stop/Send$ event) or the couple of nodes

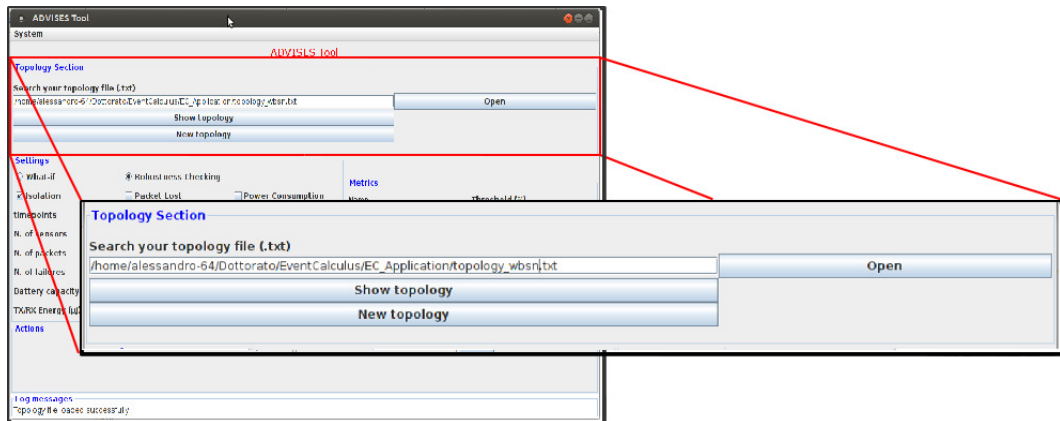


Figure 6.2: Topology loading with ADVISES tool

(in case of *Disconnect/Connect* event) to include in the event trace. This is translated in a list of *Happens* predicates that specify nodes and timepoints in which a given event occurs.

Timepoint	Event	Node	Packet
1	Disconnect	5, 3	-

Figure 6.3: Topology loading with ADVISES tool

The *completion* statement specifies that a predicate symbol (i.e. *Happens*) should be subject to predicate completion.

Listing 6.1: Initial event trace for a WBSN

```

1 Happens(Disconnect(5 , 3),1) .
2 Happens(Stop(4),3) .
3
4 completion Happens

```

Finally, in the last part we consider ranges of values for sensors and timepoints.

In this case we know that the network is composed by 7 nodes and we want to observe what it could happen in 10 timepoints.

Listing 6.2: Parameters for a WBSN

```
1 range sensor 1 7
2 range time 0 10
```

Listing 6.3 reports the outcome (the *narrative*) produced by the DECReasoner when invoked by the ADVISES tool. The event trace confirms our expectations. We can observe that after the stop of node 4, nodes 6 and 7 becomes not reachable. Considering that node 5 was already not reachable, this means that a total of 4 nodes are isolated. The coverage is computed as the time point of the last failure event causing such isolation, that is 3. Consequently, the connection resiliency is computed by counting the number of failure and disconnection events in the interval $[0, 3]$, excluding the last event; hence, it is equal to 1.

Listing 6.3: Outcome of the DECReasoner for a WBSN

```
0
1
Happens(Disconnect(5, 3), 1).
2
-IsLinked(5, 3).
Happens(Isolate(5), 2).
3
-IsReachable(5).
Happens(Stop(4), 3).
4
-IsAlive(4).
Happens(Isolate(6), 4).
5
-IsReachable(6).
Happens(Isolate(7), 5).
6
-IsReachable(7).
```

7
8
9
10

6.2 Case study 2 (what-if analysis): A more complex WSN

6.2.1 Scenario

As second case study, we consider the topology of a WSN that has been realized in our laboratory using TMOTE sensors.

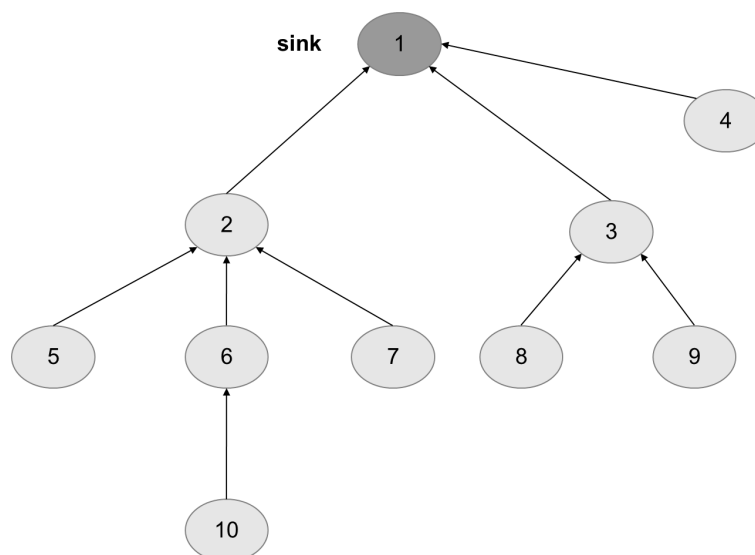


Figure 6.4: Topology of a more complex WSN

This WSN (figure 6.4) is constructed by mounting ten sensor nodes. Node 1 is the sink of the network. Each node consists of a TMOTE SKY XM1000 (Tiny-OS and ContikiOS compatible) which follows the IEEE 802.15.4 standard.

We are interested to evaluate the behavior of the network, in terms of coverage, connection

resiliency, and data delivery resiliency, when the following sequence of basic events occurs: *Send(1,5)* at timepoint 1, *Send(1,9)* at timepoint 2, *Stop(4)* at timepoint 4 and *Disconnect(3 , 1)* at timepoint 7.

For a coverage threshold set at 70%, we should observe that coverage interval is equals to $[0, 7]$ (i.e., when node 3 disconnects from node 1 at time point 7, 4 nodes are not reachable, namely 3, 4, 8 and 9), and a connection resiliency equals to 1 (i.e., only the disconnection event is tolerated).

For a data delivery resiliency threshold set at 60%, we should observe that the packets sent by node 5 and node 9 are received by the sink without being lost.

6.2.2 Results

The initial event trace produced by the ADVISES tool is given in Listing 6.4.

Listing 6.4: Initial event trace for a more complex WSN

```
1 Happens(Send(1,5),1) .
2 Happens(Send(1,9),2) .
3 Happens(Stop(4),4) .
4 Happens(Disconnect(3 , 1),7) .
5
6 completion Happens
```

The ADVISES tool generates the values for the range of sensors and timepoints, again analyzing user inputs. In this case, we know that the network is composed by 10 nodes that can send at most 1 packet and we want to observe what it could happen in 10 timepoints.

Listing 6.5: Parameters for a more complex WSN

```

1 range sensor 1 10
2 range pkt 1 1
3 range time 0 10

```

Also in this case, the outcome produced by the DECReasoner (see listing 6.6) reports the event trace by means of which we can validate our assumptions. In fact, we can observe that when node 4 stops, there are still 9 reachable nodes. When there is a disconnection between nodes 3 and 1, there are 6 reachable nodes. For this reason the coverage is 7 because in the interval $[0, 7]$ the covered nodes are 9 on 10 (90%); the connection resiliency is equal to 1. Finally, the data delivery resiliency is 100%; the packets sent by node 5 and node 9 are correctly received by the sink.

Listing 6.6: Outcome of the DECReasoner for a more complex WSN

```

0
1
Happens(Forward(1, 5, 5, 2), 1).
Happens(Send(1, 5), 1).
2
+IsInDelivery(1, 5).
+IsOnChannel(1, 5, 5, 2).
Happens(Catch(1, 5, 2, 5), 2).
Happens(Forward(1, 9, 9, 3), 2).
Happens(Send(1, 9), 2).
3
-IsOnChannel(1, 5, 5, 2).
+IsInDelivery(1, 9).
+IsOnChannel(1, 9, 9, 3).
Happens(Catch(1, 9, 3, 9), 3).
Happens(Forward(1, 5, 2, 1), 3).
4
-IsOnChannel(1, 9, 9, 3).
+IsOnChannel(1, 5, 2, 1).
Happens(Catch(1, 5, 1, 2), 4).
Happens(Forward(1, 9, 3, 1), 4).
Happens(Receive(1, 5), 4).

```



```
Happens(Stop(4), 4).
5
-IsAlive(4).
-IsInDelivery(1, 5).
-IsOnChannel(1, 5, 2, 1).
+IsOnChannel(1, 9, 3, 1).
Happens(Catch(1, 9, 1, 3), 5).
Happens(Receive(1, 9), 5).
6
-IsInDelivery(1, 9).
-IsOnChannel(1, 9, 3, 1).
7
Happens(Disconnect(3, 1), 7).
8
-IsLinked(3, 1).
Happens(Isolate(3), 8).
9
-IsReachable(3).
Happens(Isolate(8), 9).
Happens(Isolate(9), 9).
10
-IsReachable(8).
-IsReachable(9).
```

6.3 Case study 3 (robustness checking): Is it worth to add a node?

6.3.1 Scenario

As a first case study for robustness checking let us consider a simple WSN with 6 nodes (Figure 6.5). This topology is commonly adopted to monitor a linear structure, such as a tunnel or an oil pipeline. From the figure, it is intuitive to conclude that node 2 represents the most critical dependability bottleneck for this topology, since it has to route the packets from all other nodes to the sink.

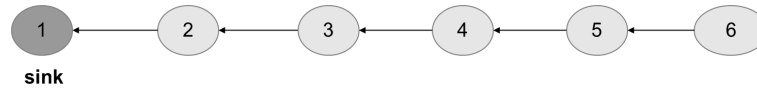


Figure 6.5: WSN with line topology

The simplicity of the topology allows to reason on dependability bottlenecks, and on potential improvements. In particular, the objective of the case study is to quantify the benefits, in terms of connection resiliency, of adding one extra node (see figure 6.6).

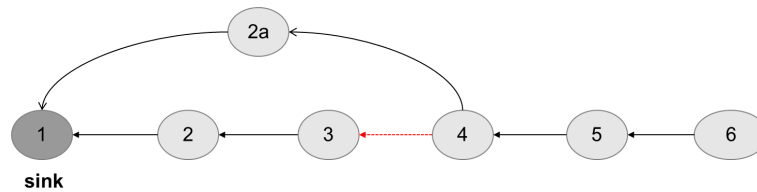


Figure 6.6: WSN with extra node in line topology

6.3.2 Results

In order to obtain the results, shown in table 6.1, we have considered in both topologies a reasoning performed on 10 timepoints, a coverage threshold value equal to 70% (5 reachable nodes), and a number of failures from 1 to 4.

Table 6.1: Results of simple linear topology

Outcome	Simple linear topology							
	6 sensors				6 sensors + 1 extra			
No. of Failures	1	2	3	4	1	2	3	4
Connection Resiliency	1	0	0	0	1	2	0	0
Percentage of Cases	20%	-	-	-	66%	24%	-	-

For both topologies we report the number of failures, the connection resiliency, and the percentage of the cases in which the connection resiliency is not 0.

For example in simple linear topology (without extra node), exploring all of the cases in which 1 failure occurs, we have connection resiliency equal to 1 in 20% of the cases. In the other cases, the coverage is below 70%. If 2 failures occur, we have no cases in which coverage is above 70% and so, the maximum connection resiliency level achievable is 1. This confirms numerically that the topology is extremely fragile and susceptible to failures in the majority of cases when undesired events occur.

If we add an extra node then we gain benefits because we triple the chances (66%) to have connection resiliency with coverage $>70\%$ in case of 1 failure, and we have coverage upper than threshold value also when 2 failures occur (in the 24% of the cases). Hence, in this case the maximum connection resiliency level is 2. In this way we can assert that adding a node (accounting for 17% of extra cost) in the proximity of the sink allows to significantly boost the robustness of the WSN, by triplicating the chances of survival in case of 1 failure, and by doubling the maximum connection resiliency achievable.

6.4 Case study 4 (robustness checking): Checking robustness in harsh environments

6.4.1 Scenario

In this case study we consider the WSN with 8 nodes reported in figure 6.7. In particular we want to observe how the WSN reacts and consequently, how the percentage of data delivery changes with different disconnection probabilities. This is useful to check the robustness of the WSN in different deployment scenarios, e.g., from environments with a good channel

quality (for instance, an outdoor scenario in good weather conditions, with no interferences) to harsh environments (such as, indoor scenario with fading due to obstacles and walls and electromagnetic interferences due to the presence of other wireless devices).

Specifically, we analyze the percentage of delivered packets when every link has a probability of disconnection ranging from 5% to 40% with step 5%. We expect that, the more the disconnection probability grows, the more failures occur, and the less is likely that a packet is delivered to the sink. In particular, we want to check under which operational conditions the network is still able to deliver more than 50% of packets to the sink.

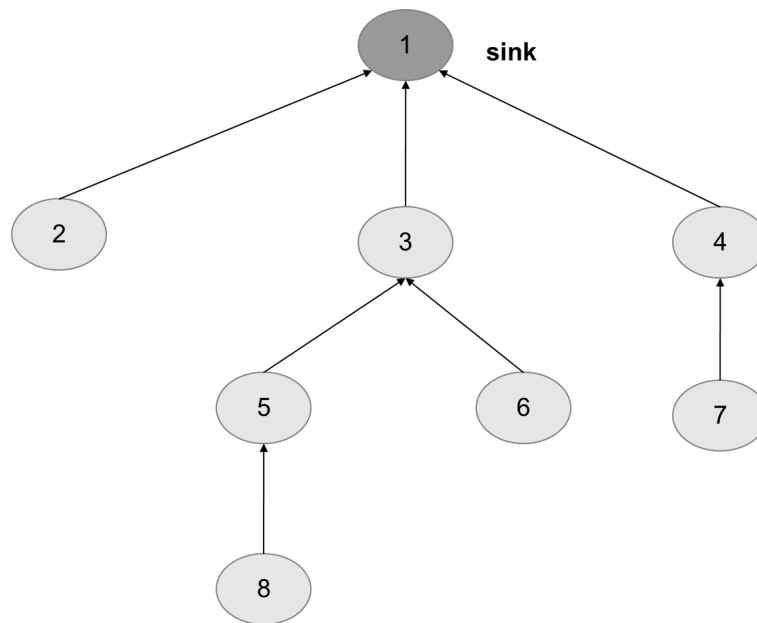


Figure 6.7: WSN topology with 8 nodes

6.4.2 Results

The graph in figure 6.8 shows the results of the study, where O.F. (Occurred Failures) represents the number of the occurred failures and D. Pkt. (Delivered Packets) the percentage

of delivered packets. Both the values are reported as a function of the disconnection probability from 5% to 40%. Each point of the graph is obtained by repeating the experiment 3 times, letting the ADVISES tool generate different random sequences starting from the disconnection probability set by the user. For this reason, each point represents a mean value, and also standard deviation bars are reported.

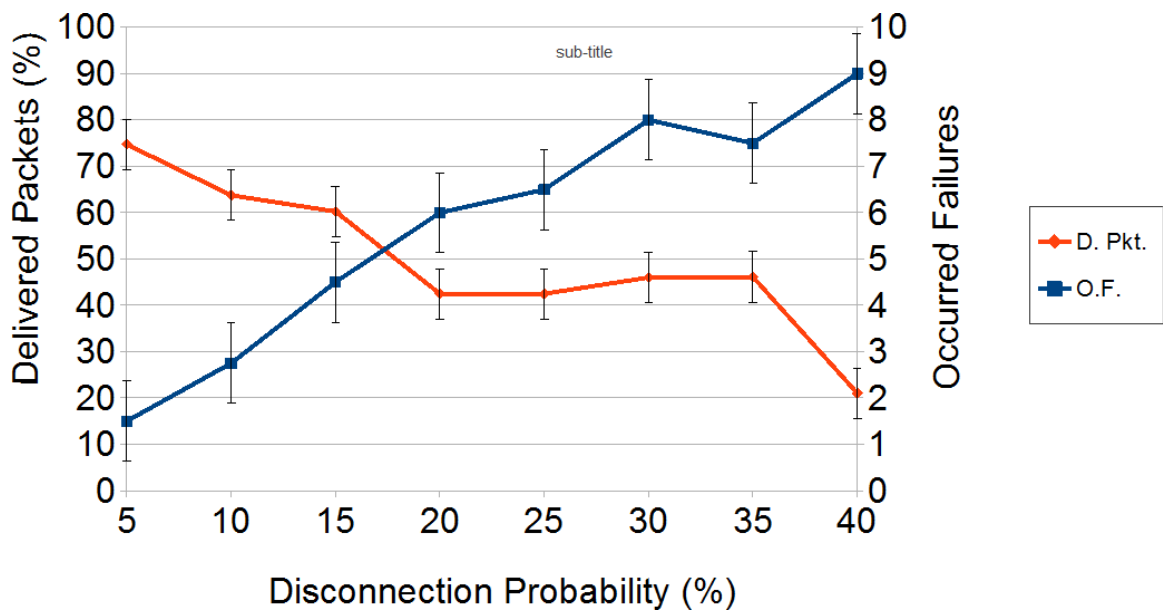


Figure 6.8: Results of the case study

From the graph, we can observe the expected inverse relationship between the trend of the occurred failures and the trend of delivered packets: when the probability of disconnection increases, also the number of the failures increases, whereas the number of the delivered packets decreases.

It is interesting to observe a certain resilience of the network for disconnection probability values ranging from 20% to 35%. In this range it seems that, even if the number of failures keeps increasing as expected, the network is redundant enough to tolerate them. Hence the

percentage of delivered packets remains the same, and we need to stress the network up to a 40% of disconnection probability to observe a more significant loss in the percentage of delivered packets.

Considering instead our requirement on checking up to which conditions the WSN is able to deliver more than 50% of packets to the sink, we can observe that this requirement is satisfied up to a disconnection probability of 15%, and a number of failures below 5. This means that the data delivery resiliency for this network is 5 (with the threshold set to 50%), and that the WSN is able to conform to expectations only if deployed in an environment where quality of channels is such that the disconnection probability of links does not overcome the critical level of 15%.

6.5 Case study 5 (runtime verification): WSN robustness checking at runtime

6.5.1 Scenario

In the last case study we consider the WSN topology composed by 8 nodes and reported in figure 6.9.

We want to verify this WSN at runtime and thus we consider a Runtime Verification scenario (illustrated in figure 6.10). We see how a failure that occurs in node 5 (*Stop(5)*) is notified to a monitor that is running on the gateway. The monitor, received the event, sends it to the tool which starts the reasoning by means of the DECReasoner.

Then we suppose that *Stop(7)* event occurs and it is notified again to a monitor in charge

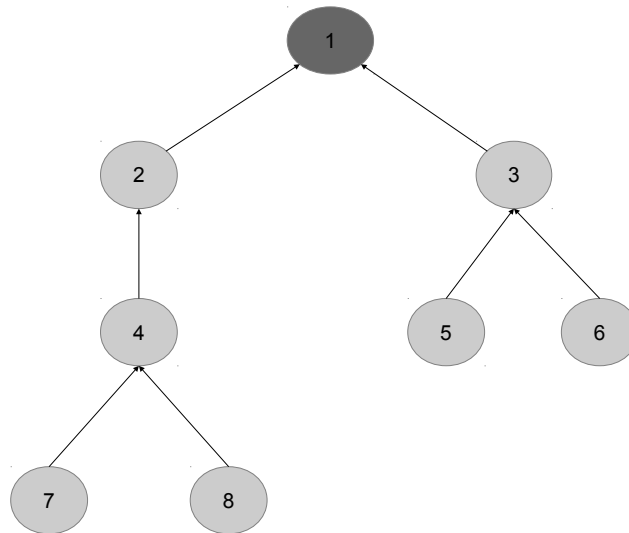
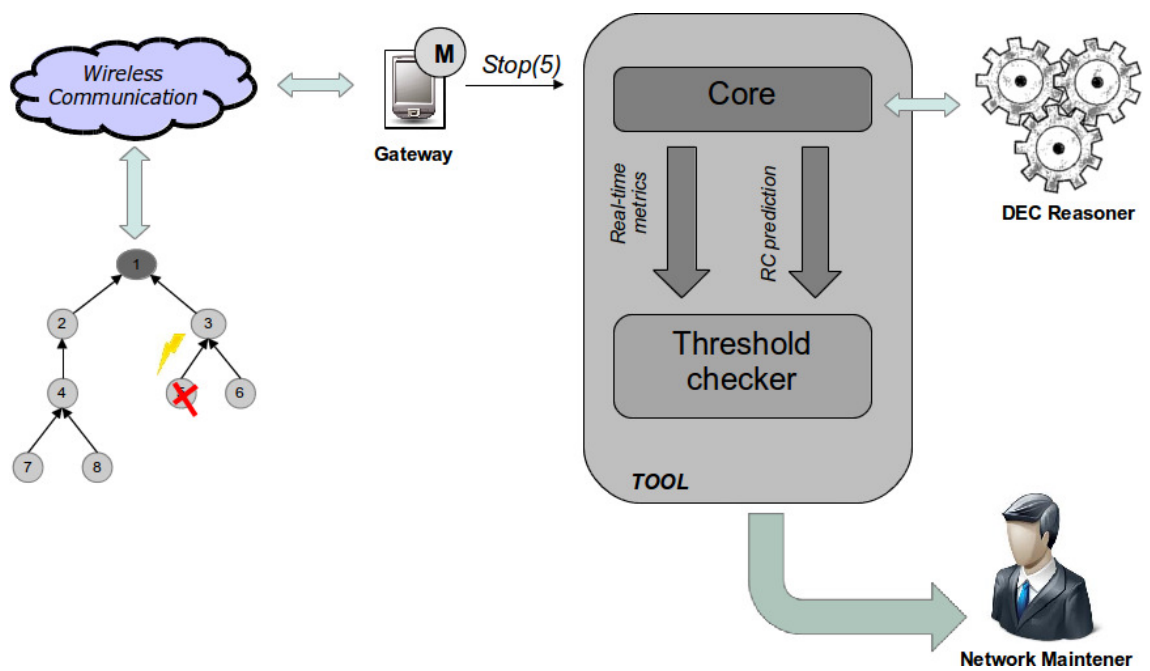


Figure 6.9: WSN topology in dynamic scenario

Figure 6.10: Scenario with *Stop(5)* event

of sending the event to the ADVISES tool (Figure 6.11).

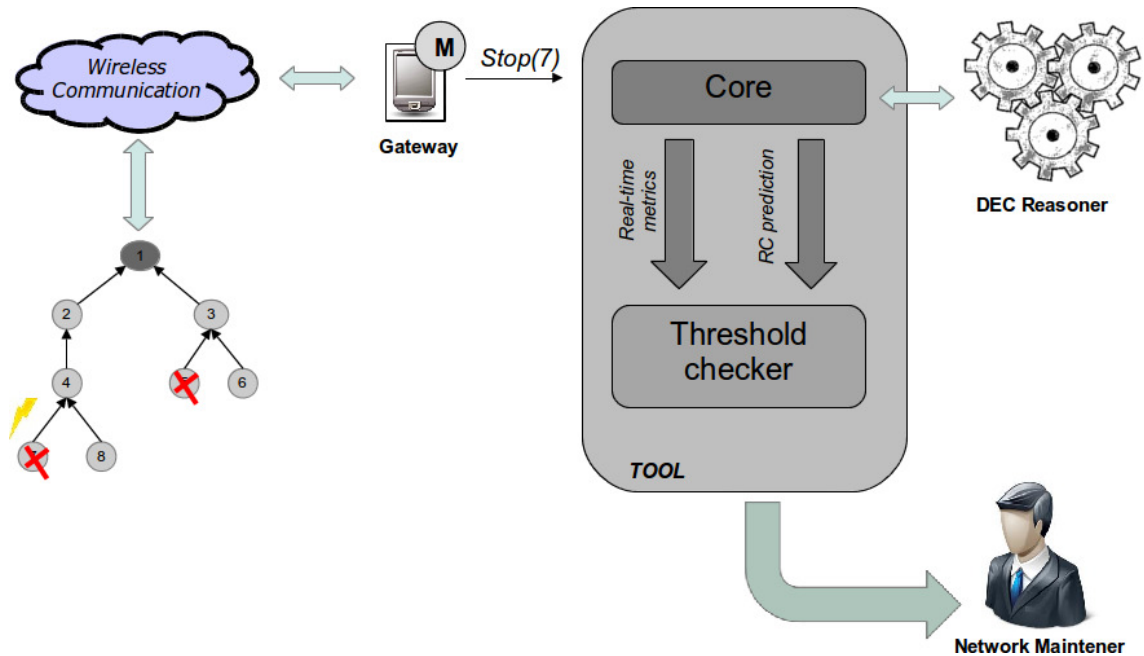


Figure 6.11: Scenario with $Stop(7)$ event

We are interested to detect the criticalities of the WSN at runtime when a sequence of events occurs.

For this scenario no initial event trace has to be defined and no channel model or number of possible occurring failures. The ADVISES is in running and initially is in a sleeping phase until it receives an event from the system monitor.

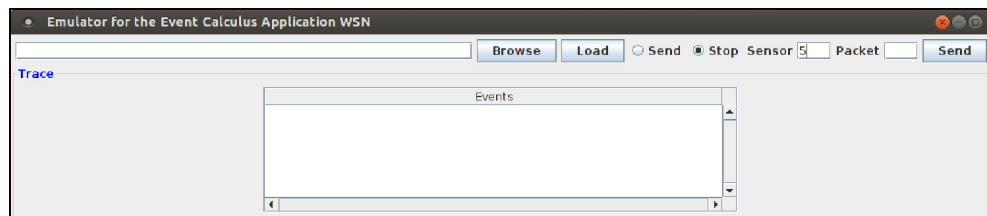
For a coverage threshold set at 65%, we should observed that when it occurs the first event ($Stop(5)$), there would be 7 reachable nodes and thus the coverage would be higher than the threshold value; moreover it would be necessary to control the nodes 2, 3, 4 because if they failed there would be at most 5 reachable nodes and the coverage would go under the desired threshold.

Then when it occurs the second event ($Stop(7)$) there would be 6 reachable nodes and thus the coverage still would be higher than the threshold value; even in this case it would be necessary to continue controlling the nodes 2, 3, 4.

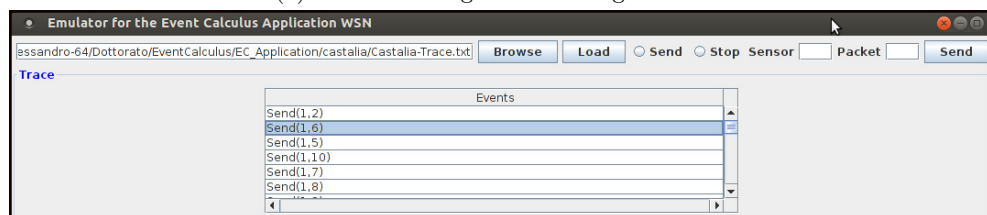
Moreover in this case study, for every received event, the ADVISES tool performs forecasting at 2 and 3 levels (that is performing robustness checking with two and three subsequent failures included the current event). In this way the network maintainer can know not only the next critical nodes but also the nodes that could become dangerous in a second moment and thus intervening with changes (e.g. adding a node) to avoid undesired consequences.

6.5.2 Results

To test the runtime behavior of the ADVISES tool, the events are emulated and sent to the tool by means of a simple java-based program (figure 6.12).



(a) Emulator for generic sending of events



(b) Emulation by means of Castalia file

Figure 6.12: Java-based emulator

It has been realized with a two functionalities: the first one consists in the manually generating *Stop* and *Send* events (figure 6.12a); the second one in extracting, from a file obtained with Castalia simulator, a real sequence of events that may occur in the target WSN file (figure 6.12b). Even if we use an emulator, the server-side interface of the tool does not change and its behavior would be the same even if the events were sent by a WSN.

ADVISES, having loaded the topology file, knows the WSN topology and it is listening in *server mode*; it just receives *Stop(5)* event, it automatically generates the EC file and starts the reasoning invoking the DECReasoner (figure 6.13); the received event is annotated in the *Log events* panel.



Figure 6.13: ADVISES GUI in dynamic verification

The first result is shown in figure 6.14: the ADVISES tool is able to inform the network maintainer about current robustness of the network: for a coverage level set to 65% we have current coverage at 88%.

After this result the ADVISES tool starts the computation about possible critical nodes; it performs a reasoning to detect the next failure (2-level forecasting) and then the next couple of failures (3-level forecasting) that could cause a coverage value under the threshold value. Results are shown in figure 6.15 and 6.16 respectively.

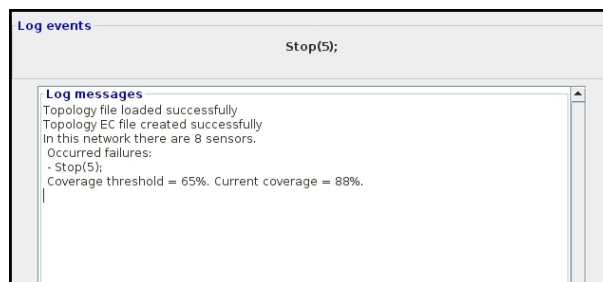


Figure 6.14: ADVISES GUI receives first event from the WSN

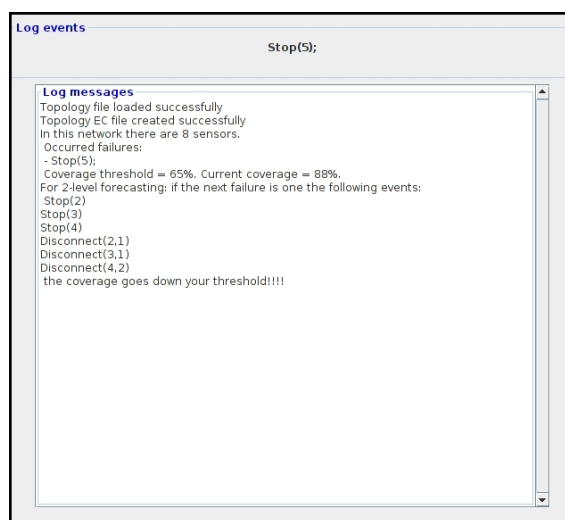


Figure 6.15: 2-level forecasting with first detected event

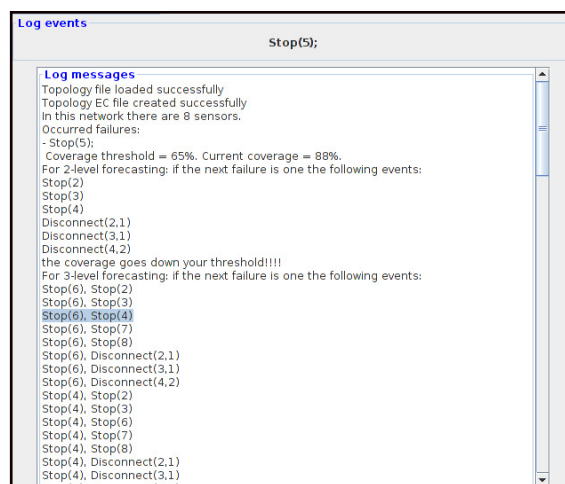


Figure 6.16: 3-level forecasting with first detected event

From these figures we can see that nodes 2, 3, 4 are particularly critical: if it occurs a failure in one of these nodes, the WSN is not dependable anymore on the basis of the set constraints. Moreover from these results we also know all the sequences of two events that can be dangerous: for example it is true that if it occurs $Stop(6)$ after $Stop(5)$, the coverage still is good but if it occurs $Stop(6)Stop(4)$ the effect can be dangerous.

This information can help the network maintainer to making modifications at the WSN in order to improve the its quality and the robustness; for instance, to add a redundant node between nodes 2 and 3 after the failure of node 5. Given these results, the ADVISES tool becomes again in a sleeping state ready to intervene in case of next detected event.

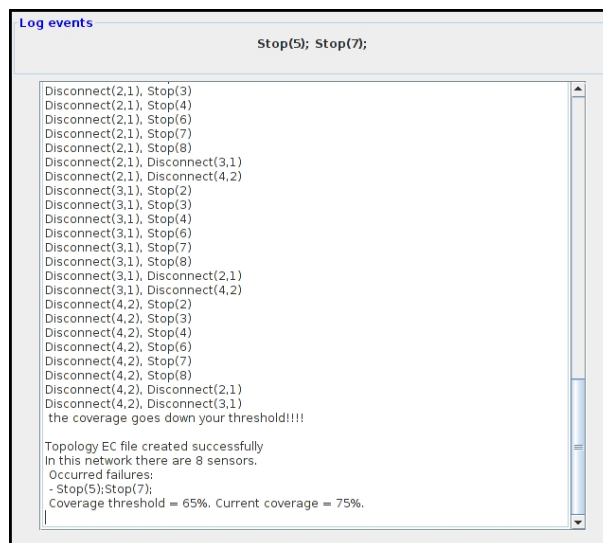


Figure 6.17: ADVISES GUI receives second event from the WSN

To conclude let us observe the scenario after the $Stop(5)$ event; in the WSN the $Stop(7)$ event occurs. Figure 6.17 shows the GUI updated with the new event and the result about new current coverage: the coverage value is decreased from 88% to 75% but it still is higher than the threshold value. The reader can note that in Log event panel it is reported the

sequence of the events detected.

Conclusions

This thesis addressed the problem of the dependability assessment of WSNs with formal methods. Assessing the dependability of WSNs is a crucial task since this kind of networks are more and more used into critical application scenarios where the level of trust becomes an important factor; depending on application scenarios, different dependability requirements can be defined, such as, node lifetime, network resiliency, and coverage of the monitoring area. From a preliminary analysis it emerged the need of verifying a WSN at design time in order to increase the confidence about the robustness of designed solutions before putting it into operation and the need of monitoring a WSN during operation in order to avoid unexpected results or dangerous effects and thus to perform what in the literature is defined *continuous monitoring*.

We have seen that formal methods are widely adopted in the literature to verify the correctness of a system specification but until now not much applied for checking of dependability properties in the WSNs.

To address these issues, the research activity dealt with the definition of formal specifications used for the behavioral checking of WSN based systems in static and runtime phases; a set of correctness specifications applied to a generic WSN has been defined using Event

Calculus as formal language since the behavior of a WSN can be characterized in terms of an event flow (e.g. a node turns on, a packet is sent, a node stops due to failure, etc.) and the Event Calculus formalism allowed to easily specify the system in terms of events.

In particular, the main contributions of this dissertation have been: i) the definition of the formal specification of WSN correctness as two logical sets: a *general correctness specification*, valid independently of the particular WSN under study, and a *structural specification* related to the properties of the target WSN (e.g., number of nodes, topology, channel quality, initial battery charge), designed to be generated automatically, ii) the use of specific WSN dependability metrics, such as *connection resiliency*, *coverage*, *data delivery resiliency* and *lifetime*, as drivers to guide design choices, iii) the realization of two types of verification techniques exploiting the same set of formal specifications (*static verification* and *dynamic verification*), iv) the development of an automated verification tool, named *ADVISES* (**A**utomate**D** **V**er**I**fication of **w**S**n** with **E**vent calculu textbf**S**), to simplify the adoption of the proposed approach and v) the presentation of the usefulness of the approach in the context of case studies, to show how the proposed framework and tool can help system engineers to take decisions upon key design and runtime questions.

The ADVISES tool has been conceived: i) to operate in double mode: static and dynamic, ii) to automatically generate the structural specifications given the properties of a target WSN (e.g. topology), iii) to perform the reasoning starting from the correctness and structural specifications, iv) to compute dependability metrics starting from the event trace produced by the reasoner, and v) to receive events in real-time from a WSN to start runtime verification and to evaluate current and future criticalities.

Other than providing specific considerations on the presented case studies, in this chapter we discuss on some lessons which have been learned and that can be reasonably considered when performing the dependability assessment of WSNs with formal methods.

Lessons Learned

Static and Dynamic Verification Using Same Specifications

The first main lesson learned from this dissertation has been the proposal of a methodology that merges static and dynamic verification exploiting a single set of correctness properties and one tool as mean for adapting this methodology for any WSN.

Hence by means of a single set of correctness specifications for WSNs (in the Event Calculus formalism) we have been able to perform three different techniques: *What-if Analysis*, *Robustness Checking* and *Runtime Verification*. The first two for static verification, the third for dynamic verification. This has been possible by relying on a single tool (ADVISES) able to automatically generate on the basis of a target WSN complete specifications (general correctness specifications + structural specification) and from these calculate desired dependability metrics.

Considerations About Case Studies

The effectiveness of the approach has been shown in the context of five case studies. They allowed to illustrate that the reasoning performed on the defined specification produced results which are consistent with our expectations. More importantly, they have shown how

the results have been useful to drive design choices (e.g., whether is worth to add a node) and to check limit operational conditions (the minimum channel quality required to let the WSN work to expectations).

In particular in the first two case studies we have proved the validation of the general correctness specifications of WSNs defined in this thesis; it has also been an opportunity to prove the correct operating of the proposed tool that automatizes our methodology and to comment the outcome generated by the DECReasoner.

More interesting results have been obtained with the last three case studies.

Firstly, the third case study has allowed us to show one of the capacities of the robustness checking technique performed with our methodology. It is possible to perform static verification in order to study possible improvements in a WSN. In fact in this case study we have seen that, starting from a simple linear topology, adding a node between the node 4 and the sink node has been advantageous since we gained benefits: we tripled the chances (66%) to have connection resiliency with coverage $>70\%$ in case of 1 failure, and we have had coverage upper than threshold value also when 2 failures occurred (in the 24% of the cases). Hence, the maximum connection resiliency level has gone from 1 to 2. In this way we have asserted that adding a node (accounting for 17% of extra cost) in the proximity of the sink allowed to significantly boost the robustness of the WSN, by triplicating the chances of survival in case of 1 failure, and by doubling the maximum connection resiliency achievable.

In the fourth case study we have studied the robustness of a WSN in which every node periodically sent a packet towards the sink node. We have observed that the WSN has been

able to deliver more than 50% of packets to the sink keeping a disconnection probability lower of 15% and tolerating a number of failures below 5. This meant that the data delivery resiliency for this network is 5 (with the threshold set to 50%), and that the WSN has to be deployed in an environment where quality of channels is such that the disconnection probability of links does not overcome the critical level of 15%.

The fifth and last case study has shown the runtime verification technique performed by means of Event Calculus without using external tools and languages (like Reactive Event Calculus). We have seen how a generic event coming from the WSN (i.e. *Stop(5)*) is detected by a monitor that is running on the gateway and sent to the ADVISES tool which starts the reasoning by means of the DECReasoner. The aim of this case study has been to describe practically the dynamic verification technique. From an event we are able to have several outcomes, such as the current critical nodes that can compromise the robustness of the network, the next critical nodes that have to be particularly monitored.

On the Criticality of the Reasoning Time

The definition of the general correctness specifications (for isolation, packet loss and battery exhaustion event) has been an hard and long task since we had to write their informal descriptions in Event Calculus formalism, to realize some initial simple scenario and verify/test them by means of the DECReasoner.

Among them we found particular criticality of the packet loss event specification and thus battery exhaustion event specification (because the last one is linked to the packet loss event). While the reasoning performed on the specification for the isolation event took few

time during the reasoning, the reasoning performed on the specification for packet loss (and for battery exhaustion) took much more time, producing the desired output anyway.

This is due to a more complex logic that characterizes this specification and surely it is necessary to review it in order to improve the reasoning time and so the performances of the proposed methodology.

Moreover due to long reasoning time we had to consider WSNs with topologies composed by few nodes (about 10); performing some tests, both with 100 and 30 nodes we observed that the DECReasoner took much time without returning the outcome but an error message that at the moment is not documented; after several tests we asserted that the limit of the nodes that can be considered is about 10 (considering health monitoring scenario it is a reasonable number for a WBSN that is characterized usually from not more of 5/6 nodes). Probably the long reasoning time depends on our defined specifications that may be optimized in the future work.

Next Steps

This dissertation demonstrated that it is possible to assess the dependability of WSNs by means of the formal methods in particular Event Calculus formalism.

However, the proposed approach has also to resolve some design challenges (criticalities of the reasoning time) as documented in the last point of the lessons learned. As future steps, we want to review all the general correctness specifications optimizing them and improving the reasoning time in order to apply the approach at WSN with high number of nodes.

Alternatively we can suppose to follow the *divide et impera* concept: to subdivide the WSN in subnets and to apply the approach for each subnet thus to collect all the obtained data. Moreover we want to apply the entire methodology to a real WSN, to define new metrics of dependability not oriented towards the WSNs but considering the failure events that may occur in other components of the WSN-based systems.

Finally another aim is to adopt all of this approach (static and dynamic verification with formal methods) in completely different scenarios: for example adopting the approach for gesture and activity recognition for patients with cognitive disorders.

Bibliography

- [1] Emile Aarts and Reiner Wichert. Ambient intelligence. *Technology Guide*, pages 244–249, 2009.
- [2] H.M.F. AboElFotouh, S.S. Iyengar, and K. Chakrabarty. Computing reliability and message delay for cooperative wireless distributed sensor networks subject to random failures. *Reliability, IEEE Transactions on*, 54(1):145 – 155, march 2005.
- [3] U.S. National Aeronautics and Space. Us mil std 1629 1980: Procedure for performing a failure mode, effect and criticality analysis, method 102, November 1980.
- [4] S.I. Ahamed, M. Zulkernine, and S. Anamanamuri. A dependable device discovery approach for pervasive computing middleware. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pages 8 pp.–, April.
- [5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A Survey on Sensor Networks. *IEEE Communications Magazine*, 40(8):102–114, August 2002.
- [6] Ian F Akyildiz and Mehmet Can Vuran. *Wireless sensor networks*, volume 4. Wiley, 2010.
- [7] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, and Paolo Mello, Paola Torroni. Verifiable agent interaction in abductive logic programming: the sciff framework. *ACM Transactions on Computational Logic*, V(N):1–41, 2007.
- [8] Hande Alemdar and Cem Ersoy. Wireless sensor networks for healthcare: A survey. *Computer Networks*, 54(15):2688–2710, 2010.
- [9] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell. Fault injection for dependability validation: a methodology and some applications. *Software Engineering, IEEE Transactions on*, 16(2):166–182, Feb 1990.
- [10] G. Asada, M. Dong, T.S. Lin, F. Newberg, G. Pottie, W.J. Kaiser, and H.O. Marcy. Wireless integrated network sensors: Low power systems on a chip. In *Solid-State Circuits Conference, 1998. ESSCIRC '98. Proceedings of the 24th European*, pages 9 – 16, sept. 1998.
- [11] Juan Carlos Augusto and Rodolfo Sabás Gómez. A procedure to translate paradigm specifications to propositional linear temporal logic and its application to verification. *International Journal of Software Engineering and Knowledge Engineering*, 13(06):627–654, 2003.
- [12] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004.

-
- [13] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In Bernhard Steffen and Giorgio Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science*, pages 44–57. Springer Berlin Heidelberg, 2004.
- [14] J.H. Barton, E.W. Czeck, Z.Z. Segall, and D.P. Siewiorek. Fault injection experiments using fiat. *Computers, IEEE Transactions on*, 39(4):575–582, Apr 1990.
- [15] S. Baskiyar. A real-time fault tolerant intra-body network. In *Local Computer Networks, 2002. Proceedings. LCN 2002. 27th Annual IEEE Conference on*, pages 235 – 240, nov. 2002.
- [16] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. *Bounded model checking*, volume 58. Elsevier, 2003.
- [17] Juergen Bohn, Felix C. Gartner, and Harald Vogt. Dependability Issues of Pervasive Computing in a Healthcare Environment. In *first International Conference on Security in Pervasive Computing*, 2003.
- [18] Pruet Boonma and Junichi Suzuki. Moppet: A model-driven performance engineering framework for wireless sensor networks. *Comput. J.*, 53(10):1674–1690, 2010.
- [19] Athanassios Boulis. Castalia: revealing pitfalls in designing distributed algorithms in wsn. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 407–408, New York, NY, USA, 2007. ACM.
- [20] W Bourgeois, P Hogben, A Pike, and RM Stuetz. Development of a sensor array based measurement system for continuous monitoring of water and wastewater. *Sensors and Actuators B: Chemical*, 88(3):312–319, 2003.
- [21] Commissioned by European Commission for the FP6. Study on economic impact of e-health in ambient intelligence.
- [22] Gabriella Carrozza. *Software Faults Diagnosis in Complex OTS-Based Critical Systems*. PhD thesis, PhD Thesis, Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II, www.mobilab.unina.it/tesiDottorato.html, 2008.
- [23] Gabriella Carrozza and Marcello Cinque. Modeling and Analyzing the Dependability of Short Range Wireless Technologies via Field Failure Data Analysis. *Journal of Software*, 4:707–716, 2009.
- [24] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. Toward distributed service discovery in pervasive computing environments. *Mobile Computing, IEEE Transactions on*, 5(2):97–112, Feb.
- [25] A. Chehri, P. Fortier, and P.-M. Tardif. Security monitoring using wireless sensor networks. In *Communication Networks and Services Research, 2007. CNSR '07. Fifth Annual Conference on*, pages 13–17, May.
- [26] Yunxia Chen and Qing Zhao. On the lifetime of wireless sensor networks. *Communications Letters, IEEE*, 9(11):976 – 978, nov. 2005.
- [27] Zhao Cheng, M. Perillo, and W.B. Heinzelman. General network lifetime and cost models for evaluating sensor network deployment strategies. *Mobile Computing, IEEE Transactions on*, 7(4):484–497, April.
- [28] Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. A logic-based, reactive calculus of events. *Fundam. Inf.*, 105(1-2):135–161, January 2010.

- [29] C.-F. Chiasserini and M. Garetto. Modeling the performance of wireless sensor networks. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 4 vol. (xxxv+2866), march 2004.
- [30] M. Cinque, D. Cotroneo, C. Di Martino, S. Russo, and A. Testa. Avr-inject: A tool for injecting faults in wireless sensor nodes. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8, may 2009.
- [31] Marcello Cinque. *Dependability evaluation of mobile distributed systems via field failure data analysis*. PhD thesis, PhD Thesis, Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II, www.mobilab.unina.it/tesiDottorato.html, 2006.
- [32] Marcello Cinque, Antonio Coronato, and Alessandro Testa. Dependability analysis of a vital sign monitoring application. *Lecture Notes in Engineering and Computer Science*, 2195.
- [33] Marcello Cinque, Antonio Coronato, and Alessandro Testa. On dependability issues in ambient intelligence systems. *International Journal of Ambient Computing and Intelligence (IJACI)*, 3(3):18–27, 2011.
- [34] Marcello Cinque, Antonio Coronato, and Alessandro Testa. Dependable services for mobile health monitoring systems. *IJACI*, 4(1):1–15, 2012.
- [35] Marcello Cinque, Antonio Coronato, and Alessandro Testa. A failure modes and effects analysis of mobile health monitoring systems. In Khaled Elleithy and Tarek Sobh, editors, *Innovations and Advances in Computer, Information, Systems Sciences, and Engineering*, volume 152 of *Lecture Notes in Electrical Engineering*, pages 569–582. Springer New York, 2013.
- [36] Marcello Cinque, Domenico Cotroneo, Catello Di Martino, and Stefano Russo. Modeling and assessing the dependability of wireless sensor networks. In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems, SRDS '07*, pages 33–44, Washington, DC, USA, 2007. IEEE Computer Society.
- [37] Marcello Cinque, Domenico Cotroneo, Catello Di Martino, and Alessandro Testa. An effective approach for injecting faults in wireless sensor network operating systems. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 567–569. IEEE, 2010.
- [38] Marcello Cinque, Domenico Cotroneo, Zbigniew Kalbarczyk, and Ravishankar K Iyer. How do mobile phones fail? a failure data analysis of symbian os smart phones. In *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*, pages 585–594. IEEE, 2007.
- [39] Marcello Cinque, Domenico Cotroneo, and Alessandro Testa. A logging framework for the on-line failure analysis of android smart phones. In *Proceedings of the 1st European Workshop on Approaches to MOBiquitous Resilience*, page 2. ACM, 2012.
- [40] Marcello Cinque, Catello Di Martino, and Alessandro Testa. icaas: interoperable and configurable architecture for accessing sensor networks. In *Proceedings of the 3rd international workshop on Adaptive and dependable mobile ubiquitous systems*, pages 19–24. ACM, 2009.
- [41] Marcello Cinque, Catello Di Martino, and Alessandro Testa. Analyzing and modeling the failure behavior of wireless sensor networks software under errors. In *IWCMC*, pages 1136–1141, 2012.
- [42] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Cheking*. Mit Press, 1999.

- [43] Diane J Cook, Juan C Augusto, and Vikramaditya R Jakkula. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4):277–298, 2009.
- [44] A. Coronato and G. De Pietro. Formal design of ambient intelligence applications. *Computer*, 43(12):60–68, dec. 2010.
- [45] Antonio Coronato. Uranus: A middleware architecture for dependable aal and vital signs monitoring applications. *Sensors*, 12(3):3145–3161, 2012.
- [46] Antonio Coronato and Giuseppe De Pietro. Situation awareness in applications of ambient assisted living for cognitive impaired people. *Mobile Networks and Applications*, pages 1–10, 2012.
- [47] Antonio Coronato, Massimo Esposito, and Giuseppe De Pietro. A multimodal semantic location service for intelligent environments: an application for smart hospitals. *Personal and Ubiquitous Computing*, 13(7):527–538, 2009.
- [48] Antonio Coronato and Alessandro Testa. Runtime verification of location-dependent correctness and security properties in ambient intelligence applications. In *Internet Communications (BCFIC Riga), 2011 Baltic Congress on Future*, pages 153–160. IEEE, 2011.
- [49] Antonio Coronato and Alessandro Testa. Long-term monitoring of vital signs for mobile patients. In *PECCS'12*, pages 15–20, 2012.
- [50] Jun-Hong Cui, Jiejun Kong, M. Gerla, and Shengli Zhou. The challenges of building mobile underwater wireless networks for aquatic applications. *Network, IEEE*, 20(3):12–18, may-june 2006.
- [51] Professor David Culler, Robert Szewczyk, Alec Woo, and Jason Hill. A software architecture supporting networked sensors. Technical report, Master's thesis, 2000.
- [52] Daniel-Ioan Curiac, Constantin Volosencu, Dan Pescaru, Lucian Jurca, and Alexa Doboli. A view upon redundancy in wireless sensor networks. In *Proceedings of the 8th WSEAS international conference on Signal processing, robotics and automation, ISPRA'09*, pages 341–346, Stevens Point, Wisconsin, USA, 2009. World Scientific and Engineering Academy and Society (WSEAS).
- [53] Akim Demaille. Probabilistic verification of sensor networks. In *In Proc. 4th IEEE Int. Conf. on Comput. Sci., Research, Innovation and Vision for the Future (RIVF'06)*, pages 45–54. IEEE Computer Society, 2006.
- [54] Catello Di Martino. *Resiliency assessment of wireless sensor networks: a holistic approach*. PhD thesis, PhD Thesis, Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II, www.mobilab.unina.it/tesiDottorato.html, 2009.
- [55] Catello Di Martino, Marcello Cinque, and Domenico Cotroneo. Automated generation of performance and dependability models for the assessment of wireless sensor networks. *IEEE Trans. Comput.*, 61(6):870–884, June 2012.
- [56] Catello Di Martino, Gabriele D'Avino, and Alessandro Testa. icaas: An interoperable and configurable architecture for accessing sensor networks. *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)*, 1(2):30–45, 2010.
- [57] Kent E. Dicks. Telemedicine 2.0 has arrived. *Future Healthcare Magazine*, 2007.

- [58] K. Doddapaneni, E. Ever, O. Gemikonakli, I. Malavolta, L. Mostarda, and H. Muccini. Path loss effect on energy consumption in a wsn. In *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on*, pages 569–574, march 2012.
- [59] E. O. Elliott. Estimates of Error Rates for Codes on Burst-Noise Channels. *Bell System Technical Journal*, 42:1977–1997, September 1963.
- [60] E Allen Emerson. Temporal and modal logic. *Handbook of theoretical computer science*, 2:995–1072, 1990.
- [61] GM Foody. Remote sensing of tropical forest environments: towards the monitoring of environmental resources for sustainable development. *International Journal of Remote Sensing*, 24(20):4035–4046, 2003.
- [62] Center for Technology and Aging. Technologies for remote patient monitoring in older adults. December 2009.
- [63] Yannis Georgalis, Dimitris Grammenos, and Constantine Stephanidis. Middleware for ambient intelligence environments: Reviewing requirements and communication technologies. *Universal Access in Human-Computer Interaction. Intelligent and Ubiquitous Interaction Environments*, pages 168–177, 2009.
- [64] Thilo Hafer and Wolfgang Thomas. Computation tree logic ctl and path quantifiers in the monadic theory of the binary tree. *Automata, Languages and Programming*, pages 269–279, 1987.
- [65] Salim. A Hanna. Regulations and standards for wireless medical applications. In *Third International Symposium on Medical Information and Communication Technology (ISMICT)*, 2009.
- [66] M.A. Hanson, H.C. Powell, A.T. Barth, K. Ringgenberg, B.H. Calhoun, J.H. Aylor, and J. Lach. Body area sensor networks: Challenges and opportunities. *Computer*, 42(1):58–65, jan. 2009.
- [67] Yang Hao and Robert Foster. Wireless body sensor networks for health-monitoring applications. *Physiological Measurement*, 29(11):R27–R56, November 2008.
- [68] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, page 10 pp. vol.2, jan. 2000.
- [69] J.L. Hill and D.E. Culler. Mica: a wireless platform for deeply embedded networks. *Micro, IEEE*, 22(6):12–24, nov/dec 2002.
- [70] C.M. Holloway. Why engineers should consider formal methods. In *Digital Avionics Systems Conference, 1997. 16th DASC., AIAA/IEEE*, volume 1, pages 1.3–16–22 vol.1, Oct.
- [71] Gerard J Holzmann. The model checker spin. *Software Engineering, IEEE Transactions on*, 23(5):279–295, 1997.
- [72] Y. Hovakeemian, K. Naik, and A. Nayak. A survey on dependability in body area networks. In *Medical Information Communication Technology (ISMICT), 2011 5th International Symposium on*, pages 10–14, march 2011.
- [73] Mei-Chen Hsueh, Timothy K Tsai, and Ravishankar K Iyer. Fault injection techniques and tools. *Computer*, 30(4):75–82, 1997.

- [74] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*, volume 2. Cambridge University Press Cambridge,, UK, 2004.
- [75] Carmen B Hayes James R Cook, Joe Konwinski. Failure mode effects and criticality analysis (fmeca) - home ecg test kit. 2004.
- [76] Petr Jurčik and Anis Koubâa. The iee 802.15.4 opnet simulation model: Reference guide v2.0. Technical report, IPP-HURRAY!, May 2007.
- [77] G.A. Kanawati, N.A. Kanawati, and J.A. Abraham. Ferrari: a flexible software-based fault and error injection system. *Computers, IEEE Transactions on*, 44(2):248–260, Feb 1995.
- [78] K. Kapitanova and S.H. Son. Medal: A compact event description and analysis language for wireless sensor networks. In *Networked Sensing Systems (INSS), 2009 Sixth International Conference on*, pages 1–4, 2009.
- [79] Marc-Olivier Killijian, David Powell, Michel Banâtre, Paul Couderc, and Yves Roudier. Collaborative backup for dependable mobile applications. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 146–149. ACM, 2004.
- [80] Hye-Jin Kim, Ho-Sub Yoon, and Jae-Hong Kim. User recognition based on continuous monitoring and tracking. In *Human-Robot Interaction (HRI), 2011 6th ACM/IEEE International Conference on*, pages 163–164, March.
- [81] Alexander Kott and Curtis Arnold. The promises and challenges of continuous monitoring and risk scoring. *IEEE Security & Privacy*, 11(1):90–93, 2013.
- [82] F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentelli. On-line fault detection of sensor measurements. In *Sensors, 2003. Proceedings of IEEE*, volume 2, pages 974 – 979 Vol.2, oct. 2003.
- [83] R Kowalski and M Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, January 1986.
- [84] Robert J. Latino and Anne Flood. Optimizing fmea and rca efforts in health care. *Journal of Healthcare Risk Management*, 24(3):21–28, 2004.
- [85] Jae-Joon Lee, Bhaskar Krishnamachari, and C-C Jay Kuo. Impact of energy depletion and reliability on wireless sensor network connectivity. In *Proceedings of SPIE*, volume 5440, pages 169–180, 2004.
- [86] Ren-Guey Lee, Yih-Chien Chen, Chun-Chieh Hsiao, and Chwan-Lu Tseng. A mobile care system with alert mechanism. *Information Technology in Biomedicine, IEEE Transactions on*, 11(5):507–517, Sept.
- [87] M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009. cited By (since 1996) 56.
- [88] Hector J Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B Scherl. Golog: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1):59–83, 1997.
- [89] Philip Levis and Nelson Lee. *Tossim: A simulator for tinyos networks*, page 24. 2003.
- [90] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems, SenSys '03*, pages 126–137, New York, NY, USA, 2003. ACM.

- [91] Menno Lindwer, Diana Marculescu, Twan Basten, R Zimmennann, Radu Marculescu, Stefan Jung, and Eugenio Cantatore. Ambient intelligence visions and achievements: linking abstract ideas to real-world concepts. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 10–15. IEEE, 2003.
- [92] H. Madeira and J. G. Silva. Xception: Software fault injection and monitoring in processor functional units. In *in Processor Functional Units, DCCA-5, Conference on Dependable Computing for Critical Applications*, pages 135–149, 1995.
- [93] C. Mallanda, A. Suri, V. Kunchakarra, S. S. Iyengar, R. Kannan, A. Durresi, and S. Sastry. Simulating Wireless Sensor Networks with OMNeT++.
- [94] K. L. Man, T. Vallee, H.L. Leung, M. Mercaldi, J. van der Wulp, M. Donno, and M. Pastrnak. Tepawsn - a tool environment for wireless sensor networks. *Industrial Electronics and Applications, 2009. ICIEA 2009. 4th IEEE Conference on*, pages 730–733, May.
- [95] A. Mana, C. Rudolph, G. Spanoudakis, V. Lotz, F. Massacci, M. Melideo, and J. M. Lopez-cobo. *Security Engineering for Ambient Intelligence: A Manifesto*. IGI Global, Hershey, Pa., 2007.
- [96] K. Martinez, J.K. Hart, and R. Ong. Environmental sensor networks. *Computer*, 37(8):50 – 56, aug. 2004.
- [97] John McCarthy and Patrick Hayes. *Some philosophical problems from the standpoint of artificial intelligence*. Stanford University, 1968.
- [98] Erich Mikk, Yassine Lakhnech, Michael Siegel, and Gerard J Holzmann. Implementing statecharts in promela/spin. In *Industrial Strength Formal Specification Techniques, 1998. Proceedings. 2nd IEEE Workshop on*, pages 90–101. IEEE, 1998.
- [99] Rob Miller and Murray Shanahan. Reasoning about discontinuities in the event calculus. In *in Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 63–74. Morgan Kaufmann, 1996.
- [100] Daniel Minder, Pedro José Marrón, Andreas Lachenmann, and Kurt Rothermel. Experimental construction of a meeting model for smart office environments. In *Proceedings of the First Workshop on Real-World Wireless Sensor Networks (REALWSN 2005), SICS Technical Report T2005:09*, June 2005.
- [101] A. F. Mini, Badri Nath, and Antonio A. F. Loureiro. A probabilistic approach to predict the energy consumption in wireless sensor networks. In *In IV Workshop de Comunicacao sem Fio e Computao Mvel. So Paulo*, pages 23–25, 2002.
- [102] Marco Montali. Jrec. <http://www.inf.unibz.it/montali/tools.html>.
- [103] M.M.R. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, and S. Olivieri. A framework for modeling, simulation and automatic code generation of sensor network application. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08. 5th Annual IEEE Communications Society Conference on*, pages 515 –522, june 2008.
- [104] E. Mueller. Decreasoner. <http://decreasoner.sourceforge.net>.
- [105] Erik T. Mueller. Event calculus reasoning through satisfiability. *Journal of Logic and Computation*, 14:2004, 2004.
- [106] Erik T. Mueller. A tool for satisfiability-based commonsense reasoning in the event calculus. In *FLAIRS Conference'04*, pages –1–1, 2004.

- [107] Erik T. Muller. Discrete event calculus reasoner documentation. page <http://decreasoner.sourceforge.net/csr/decreasoner.pdf>, 2008.
- [108] Jürgen Nehmer, Martin Becker, Arthur Karshmer, and Rosemarie Lamm. Living assistance systems: an ambient intelligence approach. In *Proceedings of the 28th international conference on Software engineering*, pages 43–50. ACM, 2006.
- [109] The Network Simulator NS-2. <http://www.isi.edu/nsnam/ns/>.
- [110] T. O’Donovan, J. O’Donoghue, C. Sreenan, D. Sammon, P. O’Reilly, and K.A. O’Connor. A context aware wireless body area network (ban). In *Pervasive Computing Technologies for Healthcare, 2009. PervasiveHealth 2009. 3rd International Conference on*, pages 1–8, april 2009.
- [111] Commission of the European communities. Communication from the commission to the council and the european parliament: Critical infrastructure protection in the fight against terrorism. page 702, October 2004.
- [112] P.C. Olveczky and S. Thorvaldsen. Formal modeling and analysis of wireless sensor network algorithms in real-time maude. *Parallel and Distributed Processing Symposium, International*, 0:157, 2006.
- [113] Peter Ölveczky and José Meseguer. Specification and analysis of real-time systems using real-time maude. *Fundamental Approaches to Software Engineering*, pages 354–358, 2004.
- [114] M. Paksuniemi, H. Sorvoja, E. Alasaarela, and R. Myllyla. Wireless sensor and data transmission needs and technologies for patient monitoring in the operating room and intensive care unit. *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pages 5182–5185, 2005.
- [115] M. Patrignani, N. Matthys, J. Proenca, D. Hughes, and D. Clarke. Formal analysis of policies in wireless sensor network applications. In *Software Engineering for Sensor Network Applications (SESENA), 2012 Third International Workshop on*, pages 15–21, june 2012.
- [116] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.
- [117] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson. *Analysis of wireless sensor networks for habitat monitoring. Wireless sensor networks.399–423*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.
- [118] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems, SenSys ’04*, pages 95–107, New York, NY, USA, 2004. ACM.
- [119] Muhannad Quwaider and Subir Biswas. Dtn routing in body sensor networks with dynamic postural partitioning. *Ad Hoc Netw.*, 8(8):824–841, November 2010.
- [120] Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, 27:359–380, 1991.
- [121] Grigore Roşu and Klaus Havelund. Rewriting-based techniques for runtime verification. *Automated Software Engg.*, 12(2):151–197, April 2005.

- [122] R. Romadi and H. Berbia. Wireless sensor network a specification method based on reactive decisional agents. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1–5, april 2008.
- [123] Peter Rothenpieler, Daniela Krüger, Dennis Pfisterer, Stefan Fischer, Denise Dudek, Christian Haas, and Martina Zitterbart. Flegsens - secure area monitoring using wireless sensor networks. In *In Proceedings of the 4th Safety and Security Systems in Europe*, 2009.
- [124] Murray Shanahan. The Event Calculus Explained. *Lecture Notes in Computer Science*, 1600:409–430, 1999.
- [125] Ajay K. Sharma and Deepti Gupta. Performance evaluation of routing protocols for wsns based on energy-aware routing with different radio models. *International Journal of Computer Applications*, 3(12):6–14, July 2010. Published By Foundation of Computer Science.
- [126] Ryo Shimizu, Kenji Tei, Yoshiaki Fukazawa, and Shinichi Honiden. Model driven development for rapid prototyping and optimization of wireless sensor network applications. In *Proceedings of the 2nd Workshop on Software Engineering for Sensor Network Applications, SESENA '11*, pages 31–36, New York, NY, USA, 2011. ACM.
- [127] A. Shrestha, Liudong Xing, and Hong Liu. Infrastructure communication reliability of wireless sensor networks. In *Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on*, pages 250–257, 29 2006–oct. 1 2006.
- [128] Slobodan N. Simić and Shankar Sastry. Distributed environmental monitoring using random sensor networks. In Feng Zhao and Leonidas Guibas, editors, *Information Processing in Sensor Networks*, volume 2634 of *Lecture Notes in Computer Science*, pages 582–592. Springer Berlin Heidelberg, 2003.
- [129] L. Simoncini. Architectural challenges for "ambient dependability". In *Object-Oriented Real-Time Dependable Systems, 2003. WORDS 2003 Fall. Proceedings. Ninth IEEE International Workshop on*, pages 245–249, Oct.
- [130] Raymond M Smullyan. *First-order logic*. Dover Publications, 1995.
- [131] Ian Sommerville. *Software Engineering (7th Edition)*. Pearson Addison Wesley, 2004.
- [132] J. Spadotto K. M. SE., Hawkins. Ict convergence, confluence and creativity: The application of emerging technologies for healthcare transformation. In *In Proc. of the 3rd Int. Symp. on Medical Information and Communication Technology*, February 2009.
- [133] Dean H Stamatis. *Failure mode and effect analysis: FMEA from theory to execution*. Asq Press, 2003.
- [134] Jeffrey Stanford and Sutep Tongngam. Approximation algorithm for maximum lifetime in wireless sensor networks with data aggregation. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2006. SNPD 2006. Seventh ACIS International Conference on*, pages 273–277. IEEE, 2006.
- [135] David T. Stott, Benjamin Floering, Daniel Burke, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. Nftape: A framework for assessing dependability in distributed systems with lightweight fault injectors. In *In Proceedings of the IEEE International Computer Performance and Dependability Symposium*, pages 91–100, 2000.
- [136] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys '04)*, pages 214–226, 2004.

- [137] Alessandro Testa, Antonio Coronato, Marcello Cinque, and Juan Carlos Augusto. Static verification of wireless sensor networks with formal methods. In *Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on*, pages 587–594. IEEE, 2012.
- [138] Ben L. Titzer, Daniel K. Lee, and Jens Palsberg. Avrora: scalable sensor network simulation with precise timing. In *Proceedings of the 4th international symposium on Information processing in sensor networks, IPSN '05*, Piscataway, NJ, USA, 2005. IEEE Press.
- [139] F. Van Harmelen, V. Lifschitz, and B. Porter. *Handbook Of Knowledge Representation*. Foundations of Artificial Intelligence. Elsevier, 2008.
- [140] Moshe Vardi. An automata-theoretic approach to linear temporal logic. *Logics for concurrency*, pages 238–266, 1996.
- [141] K. Wac, R. Bults, B. van Beijnum, I. Widya, V. Jones, D. Konstantas, M. Vollenbroek-Hutten, and H. Hermens. Mobile patient monitoring: The mobihealth system. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pages 1238–1241, sept. 2009.
- [142] B. Warneke, M. Last, B. Liebowitz, and K.S.J. Pister. Smart dust: communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, jan 2001.
- [143] C. Watterson and D. Heffernan. Runtime verification and monitoring of embedded systems. *Software, IET*, 1(5):172–179, October.
- [144] Daniel J. Weiss and Stephen J. Walsh. Remote sensing of mountain environments. *Geography Compass*, 3(1):1–21, 2009.
- [145] C.O.M. Ximo. *Carefusion*. Ject Press, 2012.
- [146] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A wireless sensor network for structural monitoring. In *Proceedings of the 2nd international conference on Embedded networked sensor systems, SenSys '04*, pages 13–24, New York, NY, USA, 2004. ACM.
- [147] G.Z. Yang. *Body Sensor Networks*. Springer-Verlag London Limited, 2006.
- [148] Kamyā Yekeh Yazdandoost, Kamran Sayrafian-Pour, et al. Channel model for body area network (ban). *IEEE P802*, 15, 2009.
- [149] Miguel A Zamora-Izquierdo, José Santa, and Antonio F Gómez-Skarmeta. An integral and networked home automation solution for indoor ambient intelligence. *Pervasive Computing, IEEE*, 9(4):66–77, 2010.