“SMART” SENSORS FOR CIVIL INFRASTRUCTURE SYSTEMS

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Manuel E. Ruiz-Sandoval, B.S.C.E., M.E.

B.F. Spencer, Jr., Director

David J. Kirkner, Director

Graduate Program in Civil Engineering and Geological Sciences

Notre Dame, Indiana

May 2004
“SMART” SENSORS FOR CIVIL INFRASTRUCTURE

Abstract

by

Manuel E. Ruiz-Sandoval

"Smart" sensors with embedded microprocessors and wireless communication links have the potential to change fundamentally the way civil infrastructure systems are monitored, controlled, and maintained. A 2002 National Research Council report noted that the use of networked systems of embedded computers and sensors throughout society could well dwarf all previous milestones in the information revolution. Structural health monitoring and control systems (SHM/C) represent one of the primary applications for new sensor technologies. This dissertation explores the use of the smart sensor technology for the SHM/C of civil infrastructure.

Following a brief introduction to smart sensor technology, a literature review of the devices developed to date is presented. The research herein concentrates on the Mote platform developed at the University of California at Berkeley. This platform offers for the first time an open software/hardware environment for a broad range of smart sensing research.

The suitability of the accelerometer on the existing Berkeley-platform for civil engi-
neering applications is then investigated. A new sensor board (called Tadeo) is developed that has a high sensitivity accelerometer, a microphone, a thermistor, and photo resistor. The accelerometer employed overcomes many of the deficiencies of the sensor on the available boards. However, a number of the challenges still remaining are identified.

An agent-based paradigm is proposed that supports implementation of SHM/C algorithms on networks of smart sensors. Because traditional algorithms for SHM/C assume that data is centrally processed, they cannot be implemented directly in the distributed computing environment employed by smart sensors. To demonstrate the efficacy of this approach, a reference implementation of the agent-based framework is provided for a SHM system employing the AR-ARX algorithm. Numerical examples indicate that the framework is effective.

This initial research demonstrates the feasibility of using smart sensors for SHM of civil infrastructure. A new sensor board is developed and shown to meet the needs of the application. An agent-based framework for smart sensing is proposed and shown to perform well. This research begins to lay the foundation from which the many opportunities offered by smart sensing technology can be pursued.
To Nancy and Tadeo,

Without whom this enterprise could not have been completed
CONTENTS

TABLES ................................................................. vi

FIGURES ............................................................... viii

ACKNOWLEDGMENTS ..................................................... xv

1 INTRODUCTION ......................................................... 1
   1.1 Smart sensors ....................................................... 4
   1.2 Smart sensor software ........................................... 7
   1.3 Overview of the dissertation .................................... 8

2 SMART SENSORS: LITERATURE REVIEW ...................... 11
   2.1 Wireless sensors .................................................. 11
   2.2 Smart sensors ..................................................... 15
   2.3 Summary ........................................................... 22

3 BERKELEY-MOTE PLATFORM ......................................... 23
   3.1 Berkeley-mote platform ......................................... 23
   3.2 Hardware features ............................................... 24
   3.3 Software description ............................................ 29
   3.4 Summary ........................................................... 30

4 HIGH SENSITIVITY ACCELEROMETER .......................... 31
   4.1 ADXL202 accelerometer ........................................ 31
4.2 Experimental setup ........................................... 32
4.3 High sensitivity accelerometer. ............................... 36
4.4 Four-pole Butterworth anti-alias filter ...................... 42
4.5 Summary ......................................................... 46

5 DEVELOPMENT OF AN AGENT-BASED FRAMEWORK FOR SHM USING SMART SENSORS

5.1 Introduction .................................................. 47
5.2 Artificial intelligence ....................................... 48
5.3 Agents .......................................................... 51
5.3.1 Agent architectures ..................................... 54
5.3.2 Agent languages .......................................... 57
5.4 Multiagent systems ......................................... 59
5.5 Agent framework for smart sensors ......................... 60
5.5.1 Gaia methodology ....................................... 62
5.5.2 SHM agent-framework ................................. 73
5.6 Summary ......................................................... 86

6 REFERENCE IMPLEMENTATION OF THE AGENT-BASED SHM SYSTEM 88

6.1 Current SHM algorithms. .................................... 88
6.2 Pattern recognition ........................................... 90
6.2.1 Statistical pattern recognition technique ............... 91
6.3 The AR–ARX method ....................................... 97
6.3.1 AR–ARX applications .................................. 102
6.4 Implementation of the AR-ARX technique in the agent-based framework 103
6.5 Numerical validation ....................................................... 115
   6.5.1 AR–ARX technique without agent-framework ................. 115
   6.5.2 AR–ARX technique using agent-framework .................... 120
   6.5.3 Comparison of results ............................................. 121
6.6 Summary ..................................................................... 123

7 CONCLUSIONS AND FUTURE STUDIES .......................... 124
   7.1 Conclusions .............................................................. 124
   7.2 Future studies........................................................... 125
      7.2.1 Implementation in smart sensor array ....................... 126
      7.2.2 Intel mote ........................................................... 126
      7.2.3 Extension of agent-based paradigm to complex systems . 128
      7.2.4 Other damage detection algorithms ......................... 133

APPENDIX : Analysis and design stage of the Gaia methodology .... 135

BIBLIOGRAPHY ............................................................... 177
TABLES

Table 3.1: Characteristics of the Mica2 and Mica2dot processor boards ........ 25
Table 3.2: Electronic characteristics of the sensors used by the sensor boards . 27
Table 3.3: Description of the available sensor boards .............................. 28
Table 4.1: Characteristics of the ADXL202E accelerometer ....................... 32
Table 4.2: Characteristics of the 393B04 accelerometer ........................... 33
Table 4.3: Characteristics of the SD-1221 accelerometer .......................... 38
Table 5.1: Operators for liveness expressions (Wooldridge et al. 2000) ....... 66
Table 5.2: Instance qualifiers (Wooldridge et al. 2000) ............................ 71
Table 5.3: Service model of ConcreteMaker ........................................ 73
Table 7.1: Characteristics of the Intel® mote prototype ............................ 127
Table 7.2: Topology for node B ..................................................... 130
Table 7.3: Back up topology for node B ............................................. 131
Table A.1: Service model of the User agent ........................................ 170
Table A.2: Service model of the UserHandler agent ............................... 170
Table A.3: Service model of the BaseStationManager agent ..................... 170
Table A.4: Service model of the BaseStationRadio ................................ 171
Table A.5: Service model of the BaseStationNetsync .............................. 171
Table A.6: Service model of the BaseStationClock ............................... 172
Table A.7: Service model of the BaseStationTimer ............................... 172
Table A.8: Service model of the NodeManager agent ......................... 172
Table A.9: Service model of the NodeRadio ................................. 173
Table A.10: Service model of the NodeNetsync ............................ 173
Table A.11: Service model of the NodeClock ............................... 174
Table A.12: Service model of the NodeTimer ............................... 174
Table A.13: Service model of the NodeSHM ................................. 174
Table A.14: Service model of the NodeSensing ............................ 175
Table A.15: Service model of the NodeInquirer ........................... 176
FIGURES

Figure 1.1: Damage in moment-resisting connections due to the Northridge earthquake. .................................................. 3

Figure 1.2: Traditional SHM system using centralized data acquisition. ............ 3

Figure 1.3: SHM system with smart sensors. ..................................................... 4

Figure 1.4: Traditional integrated sensors. ...................................................... 4

Figure 1.5: Smart sensor. .......................................................... 5

Figure 2.1: Sensor AWAIRS I (Agre et al. 1999). ................................. 17

Figure 2.2: Prototype smart sensor (Lynch et al. 2001). ......................... 20

Figure 2.3: Prototype smart sensor (EYES project 2002). ....................... 21

Figure 3.1: Berkeley-mote (Mica2 and Mica2dot) processor boards. ............. 25

Figure 3.2: MTS101CA sensor board. .................................................. 26

Figure 3.3: MTS300CA sensor board. .................................................. 26

Figure 3.4: MTS310CA sensor board. .................................................. 27

Figure 3.5: Motorboard (Bergbreiter & Pister 2003) ............................. 28

Figure 3.6: CotsBots robot (Bergbreiter & Pister 2003). ......................... 29

Figure 4.1: Accelerometer ADXL202E (Analog Devices, Inc. 2004). ............... 31

Figure 4.2: Accelerometer PCB 393B04 (PCB Piezotronics, Inc.). ................. 33

Figure 4.3: UCIST shaking table (Quanser Consulting Inc.). ...................... 34

Figure 4.4: Time domain plot acceleration response for a random excitation. .... 35

Figure 4.5: Power spectra density of the acceleration for a random excitation. .... 35
Figure 4.6: Time domain plot of acceleration response for a random excitation. . . 36
Figure 4.7: Power spectra density of the acceleration for a random excitation. . . 37
Figure 4.8: MAX1682 Voltage doubler, Maxim Integrated Products, Inc. (2004) . 38
Figure 4.9: LT1761ES5-5 Linear regulator (Linear Technology Corporation, 2004) 38
Figure 4.10: Prototype of the high sensitivity accelerometer .......................... 39
Figure 4.11: Tadeo sensor board. ................................................................. 40
Figure 4.12: Tadeo’s top layer connection diagram. ................................. 40
Figure 4.13: Tadeo’s bottom layer connection diagram. ............................. 41
Figure 4.14: Time domain plot of acceleration response for a random excitation. . . 41
Figure 4.15: Power spectra density of the acceleration for a random excitation. . . 42
Figure 4.16: Sallen & Key second order active low-pass filter ........................ 43
Figure 4.17: Sallen & Key four order active low-pass filter ............................ 43
Figure 4.18: Four-pole Butterworth board. ............................... 44
Figure 4.19: Time domain plot of acceleration response for a random excitation. . . 45
Figure 4.20: Power spectra density of the acceleration for a random excitation. . . 45
Figure 5.1: Agent in its environment (Wooldrige, 1999). .............................. 53
Figure 5.2: Layered agent architectures (Wooldridge, 1999). ....................... 55
Figure 5.3: Touring machine framework (Ferguson, 1992). ........................... 57
Figure 5.4: Historic development of agents (Lind, 2000). .............................. 58
Figure 5.5: Agent-based modeling (Petrie, 2000). ..................................... 62
Figure 5.6: Relationships between Gaia’s models (Wooldrige et al. 2000). ....... 64
Figure 5.7: Analysis concepts (Wooldrige et al. 2000). .............................. 65
Figure 5.8: Template of role schemata (Wooldrige et al. 2000). ................. 67
Figure 5.9: Definition of protocols (Wooldrige et al. 2000) ............................... 68
Figure 5.10: Schema for role CONCRETEMAKER ............................................. 70
Figure 5.11: Definition of Make protocol ............................................................. 70
Figure 5.12: Agent type tree. ................................................................................. 72
Figure 5.13: The CONCRETEMAKER agent model. ........................................... 72
Figure 5.14: Example’s acquaintance model ......................................................... 74
Figure 5.15: Typical network array. ....................................................................... 75
Figure 5.16: Topology formation. ............................................................................ 76
Figure 5.17: Topology rules for tree formation. ...................................................... 77
Figure 5.18: Topology update. ................................................................................. 79
Figure 5.19: Sequence of events after SHM encounters a damaged position. ....... 81
Figure 5.20: Flow-diagram of the base station smart sensor. ............................... 82
Figure 5.21: Flow-diagram of a node smart sensor. ............................................... 83
Figure 5.22: The agent model. ................................................................................. 86
Figure 5.23: The acquaintance model ................................................................. 87
Figure 6.1: Block diagram of a classifier (Fukunaga, 1972) ................................. 92
Figure 6.2: Block diagram of pattern recognition (Fukunaga, 1972) ...................... 92
Figure 6.3: Model for statistical pattern recognition (Jain et al. 2000) .................. 93
Figure 6.4: Example of group classification using Bayesian classifiers ............... 95
Figure 6.5: Example of group classification using kth nearest neighbors .......... 96
Figure 6.6: Example of group classification using neural networks .................... 96
Figure 6.7: Example of Linear and nonlinear regressions analysis ...................... 97
Figure 6.8: Example of an analysis of outliers ....................................................... 97
Figure 6.9: Flow chart for implementing a SHM program (Sohn and Farrar, 2001). 98
Figure 6.10: AR–ARX SHM system implementation (Lynch, 2002) ............. 104
Figure 6.11: StateFlow driven-event. .................................................. 104
Figure 6.12: Radio range imposed limitations. ................................. 105
Figure 6.13: Implementation of the USERHANDLER agent. ......... 106
Figure 6.14: Radio communication links between agents. ............... 107
Figure 6.15: Base Station Agent. ..................................................... 108
Figure 6.16: Node agent ................................................................. 109
Figure 6.17: NODENETSYNC state ............................................ 110
Figure 6.18: Node agent: OPERATIONAL_STATES. ......................... 111
Figure 6.19: Node agent: NODESENSING sub-states. ...................... 111
Figure 6.20: Node agent: SEN_DATABASE state. ......................... 112
Figure 6.21: Node agent: SEN_DAMAGE state .................................. 112
Figure 6.22: Node agent: NODESTRUCTURALHEALTHMONITORING states group. .......................................................... 113
Figure 6.23: Node agent: AR_ARX_DATABASE state ....................... 114
Figure 6.24: Node agent: AR_ARX_DAMAGE state ....................... 114
Figure 6.25: Node agent: DAMAGE_INDEX state ....................... 115
Figure 6.26: Complete Simulink model ...................................... 116
Figure 6.27: Ten degree-of-freedom shear building. ...................... 117
Figure 6.28: Response of a single degree-of-freedom system subject to a unit impulse. ................. 118
Figure 6.29: Damage indices plot when damage 20% is imposed in the column between 4th and 5th floors. .................................................. 119
Figure 6.30: Damage indices plot when 20% stiffness damage is imposed in the column between 7th and 8th floors. .................................................. 120
Figure 6.31: Damage indices plot when damage 20% is imposed in the column between 4th and 5th floors. ................................................................. 122

Figure 6.32: Damage indices plot when damage 20% is imposed in the column between 7th and 8th floors. ................................................................. 122

Figure 7.1: Intel® mote prototype (Kling, 2003). ................................. 126

Figure 7.2: Intel® mote project. ............................................................. 127

Figure 7.3: Backbone network. ............................................................. 129

Figure 7.4: Radio transmission levels of the network. ............................. 130

Figure 7.5: Network formation of backbone nodes. ............................... 132

Figure 7.6: Damage indices plot when 10% stiffness damage is imposed in column between 4th and 5th floors. ................................................. 133

Figure 7.7: Damage indices plot when 50% stiffness damage is imposed in column between 7th and 8th floors. ............................................... 134

Figure A.1: Schema for role USER ..................................................... 136

Figure A.2: Schema for role USERHANDLER. ..................................... 137

Figure A.3: Schema for role BASESTATIONMANAGER ....................... 138

Figure A.4: Schema for role BASESTATIONRADIO. ............................ 139

Figure A.5: Schema for role BASESTATIONNETSYNC ....................... 140

Figure A.6: Schema for role BASESTATIONCLOCK ............................ 141

Figure A.7: Schema for role BASESTATIONTIMER .............................. 142

Figure A.8: Schema for role NODEMANAGER .................................... 143

Figure A.9: Schema for role NODERADIO ......................................... 144

Figure A.10: Schema for role NODENETSYNC .................................. 145

Figure A.11: Schema for role NODECLOCK ....................................... 146

Figure A.12: Schema for role NODETIMER ...................................... 147
Figure A.28: Definition of protocols associated with NodeSHM: (a) SHMRequest, (b) SensingRequest, (c) SHMResponse, and (d) GetInquire. .......................... 166

Figure A.29: Definition of protocols associated with NodeSensing: (a) SensingRequest, and (b) SensingReply. ......................................................... 168

Figure A.30: Definition of protocols associated with NodeInquirer: (a) InquireRequest, (b) GetDamage, and (c) InquireResponse. ................................. 169
ACKNOWLEDGMENTS

When I decided to start this project five years ago, I knew that I could not do it alone. The support of my wife Nancy has been invaluable. Whenever I felt defeated or afraid, or whenever I doubted myself, she was (and is) always there for me. This achievement is also hers. Nancy and I have been blessed with the arrival of a precious boy, our son Tadeo. Together, we are now embarking on another difficult project, his education.

I also would like to thank my mother for her support and loving care. She had the tough job of keeping everything up and running at home. Thank you to my father for his encouragement, and philosophy of life. And thank you to my sister, Georgina, and my brother, Ulises, for their example and support.

The vision, expertise, and support of my advisor, Dr. B. F. Spencer, Jr., guided me in the development of this research. I tested many times his incredible patience, but he always gave me back my confidence whenever I lost it. I will forever be indebted to him.

I would like to thank Dr. Kirkner and his wife Carol for their friendship, support, and sympathy for me and my family. They have been like a second set of parents to us. I am also indebted to my friend Nelson Duran for his invaluable support and friendship.

Thanks to Dr. Ahsan Kareem, Dr. Yahya Kurama, and Dr. Lynn Salvati for reading this dissertation and offering constructive comments. Thanks to Dr. Narito Kurata at Kajima Corporation for his friendship and collaboration as together we learned the Berkeley-Mote platform.
I am indebted to Tomonori Nagayama for the innumerable discussions that we had together, and for his support in the design, manufacture and test of the anti-aliasing filter, and to Dr. Yoshihiro Nitta for his ideas in the development of the anti-aliasing filter. Finally, many thanks to my friends at the Smart Structures Technology Laboratory: Dr. Guangqiang Yang, Yong Gao, Narutoshi Nakata, Dr. Saang Bum Kim, Meghan Myers, Young Suk Kim and Juan Carrion for their friendship and companionship.

As part of my interdisciplinary formation, I greatly appreciate interaction I've had with Dr. Michael D. Lemmon, Dr. Panos Antsaklis, and their research group at Notre Dame. Later on, I had the opportunity to participate in a partnership established with Dr. Gul Agha and the Open Systems Laboratory (OSL) at the University of Illinois at Urbana-Champaign. With Dr. Wooyoung Kim, Kirill Mechitov, Sameer Sundresh, and Young Min Kwon, I had enriching discussions about various topics in computer science. I thank them for sharing their knowledge.

I would like to thank to Dr. Hoon Sohn and Dr. Charles Farrar of Los Alamos National Labs for providing me with insight into the AR-ARX methodology. Also to Dr. Jerome P. Lynch for his comments regarding the Tadeo sensor board and his willingness to share his experience with the AR-ARX method. Thanks to Kathleen Ricker for her invaluable editing job.

The research in this dissertation has been supported in part by the Universidad Autonoma Metropolitana, Consejo Nacional de Ciencia y Tecnología, Fundación Fulbright-García Robles, the CUREE-Kajima Joint Research Program Phase V, and the NSF grant CMS 0301140 (Dr. S.C. Liu, program manager).
CHAPTER 1

INTRODUCTION

The design, fabrication, and construction of smart structures is one of the ultimate challenges to engineering researchers today. Because they are the eyes and ears of the intelligent system, one of the cores of smart structures technology centers on innovative sensors and sensor systems. Structural Health Monitoring and Control systems (SHM/C) represent one of the primary applications for new sensor technologies. Indeed, much attention has been focused in recent years on the declining state of the aging civil infrastructure in the U.S., as well as on limiting the associated structural responses during extreme events (such as wind and earthquakes). These concerns apply not only to civil engineering structures, such as the nation's bridges, highways, and buildings, but also to other types of structures, such as the aging fleet of aircraft currently in use by domestic and foreign airlines.

The ability to continuously monitor the integrity and control the responses of structures in real time can provide for increased safety to the public, particularly with regard to the aging structures in widespread use today. The capability to mitigate structural dynamic response and prevent structures from reaching their limit states, in addition to the ability to detect damage at an early stage, can reduce the costs and down-time associated with repair of critical damage. Observing, controlling, and/or predicting the onset of dangerous structural behavior, such as “flutter” in bridges, can provide advanced warning to allow for
repair or removal of the structure before human lives are endangered. In addition to controlling and monitoring long-term degradation, assessment of structural integrity after catastrophic events, such as earthquakes, hurricanes, tornados, or fires, is vital. This assessment can be a significant expense (both in time and money); for example after the 1994 Northridge earthquake, large numbers of buildings needed to have their moment-resisting connections inspected (see Fig 1.1). Additionally, structures that are internally, but not obviously, damaged in an earthquake may be in great danger of collapse during aftershocks; structural integrity assessment can help to identify such structures to enable evacuation of building occupants and contents prior to aftershocks. Furthermore, after natural disasters, it is imperative that emergency facilities and evacuation routes, including bridges and highways, be assessed for safety. The need for effective SHM/C is clear, with the primary goals of such systems being to enhance safety and reliability and to reduce maintenance and inspection costs.

To effectively investigate both local and global damage criteria, a dense array of sensors is anticipated as a requirement for large civil engineering structures. The sheer number of accompanying wires, fiber-optic cables, or other physical transmission medium may be prohibitive (Fig. 1.2), particularly for structures such as long-span bridges. Consequently, global wireless communication that will facilitate low-cost, densely distributed sensing should be investigated.

To assist with the large amount of data that is generated by a monitoring system, onboard processing at the sensor allows a portion of the computation to be done locally on the sensor’s embedded microprocessor. Such an approach provides for an adaptable, smart sensor, thus reducing that amount of information that must be transmitted over the net-
Figure 1.1: Damage in moment-resisting connections due to the Northridge earthquake.

Figure 1.2: Traditional SHM system using centralized data acquisition.
work. Kiremidjian et al. (2001) pointed out that pushing data acquisition and computation forward toward the sensor is fundamental to smart sensing and monitoring systems such as that illustrated in Fig. 1.3, but represents a radical departure from the conventional instrumentation design and computational strategies for monitoring civil structures.

1.1 Smart sensors

To better understand what is meant by a “smart” sensor, first consider the definition of a standard sensor. In general, a sensor is a device designed to acquire information from an object and transform it into an electrical signal. As shown in Fig. 1.4, a classical integrated
sensor can be broken down into three components: (i) the sensing element (for example, resistors, capacitor, transistor, piezoelectric materials, photodiode, etc.), (ii) signal conditioning and processing (such as amplification, linearization, compensation, and filtering), and (iii) a sensor interface (e.g., the wires, plugs and sockets to communicate with other electronic components) (Kirianaki, et al. 2002).

As illustrated in Fig. 1.5, the essential difference between a smart sensor and a standard integrated sensor is its intelligence capabilities, i.e., the on-board microprocessor. A microprocessor is typically included for digital signal processing, analog-to-digital or frequency-to-code conversions, calculations, and interfacing functions, all of which can facilitate self-diagnostics, self-identification, or self-adaptation (decision-making) functions (Kirianaki, et al. 2002). The microprocessor can also decide when to dump or store the data and control when and for how long it will be fully awake in order to maintain a low power consumption.

A smart sensor typically has four important features: (i) an on-board Central Processing Unit (CPU), (ii) small size, (iii) wireless capability, and (iv) potential low cost. The first of these attributes has already been discussed. The other three features are explained in subsequent paragraphs.

The size of smart sensors has decreased with time. The use of MEMS has made possible the dream of having ubiquitous sensing and in particular small “smart” sensing.

Figure 1.5: Smart sensor.
MEMS devices are manufactured using very large scale integration (VLSI) technologies and can embody both mechanical and electrical functions. The main advantage this technology and its design paradigm bring to applications is miniaturization. MEMS features are typically on the scale of microns \( (10^{-6} \text{m}) \). MEMS can be used in an environment to both sense and actuate. Sensing requires that a physical or chemical phenomenon be converted to an electrical signal for display, processing, transmission, and/or recording. Actuation reverses this flow and converts an electrical signal to a physical or chemical change in the environment. MEMS devices can be found in a wide range of applications, from accelerometers for airbag deployment to electronic particle detectors that facilitate nuclear, biological, and chemical inspection.

The cost of smart sensors is also decreasing. Mass production of MEMS and microprocessors for a variety of applications has reduced their cost to a fraction of a dollar, and with their increasing popularity, their cost will eventually tend toward mere cents. Improvements in technologies for other important components, such as memory, radio transmitters, and batteries, will allow more capable and long-lasting operating devices, reducing their maintenance cost.

Finally, all smart sensors to date are wireless. Wireless data transmissions are based on radio frequency (RF) communications. There exist some protocols (or set of preestablished rules) for data transmission. One of the most popular is Bluetooth (BT), a short-range radio technology aimed at simplifying communication both among Net devices and between these devices and the internet. Most of these sensors use low radiated power to avoid the heavy costs associated with certifying the sensor with the FCC.
1.2 Smart sensor software

Implementing smart sensors for civil engineering SHM/C applications presents a number of challenges. Relatively complex algorithms for monitoring and control of structures have been developed and implemented in the laboratory. Because these algorithms assume that data is centrally processed (see Fig. 1.2), they cannot be implemented readily in the distributed computing environment employed by smart sensors.

Autonomous agents and Multi-Agent System (MAS) technologies have the potential to play a critical role in developing effective and efficient problem-solving strategies and methods in large scale smart sensor networks. MAS technologies provide a framework for building and analyzing such systems, and offer specific system mechanisms for distributed decision making and coordination (Weiss 2000). The agent-based view offers a powerful repertoire of tools, techniques, and metaphors that have the potential to improve considerably the way in which people conceptualize and implement many types of software. An application structured as a MAS will provide a number of advantages. First, the speed may be considerably increased as a result of concurrent processing. Second, there will be fewer communication bandwidth requirements because processing is located nearer the source of information. Third, response will be improved because processing, sensing, and effecting are co-located. Finally, system development will be easier because of modularity coming from the decomposition of roles into semiautonomous agents.

Autonomous agents are being used in an increasingly wide variety of applications, including structural control (Hogg and Huberman, 1998), air traffic control (Steeb et al., 1988), patient care (Huang et al., 1995), job shop scheduling (Morley and Schelberg, 1993), and transportation management (Fisher et al. 1999). An agent-based architecture
provides an important paradigm on which to lay the foundation for smart sensing for SHM/C. Indeed, Liu and Tomizuka (2003) indicated that agent-based sensing is part of the strategic research required to advance sensors and smart structures technology. Focus should be placed on selecting the best architecture, hierarchical interaction, communication, and negotiation methods for the development of a SHM algorithm. Use can be made of the various agent communication languages that have been designed (Mayfield et al. 1996; Smith and Cohen, 1996).

An effective agent-based computational framework should allow for robust SHM/C algorithm development within the intrinsic constraints imposed by the environment of network smart sensors.

1.3 Overview of the dissertation

The research detailed in this dissertation explores the use of the smart sensor technology for the SHM/C of civil infrastructure. Focus is placed on developing appropriate sensors and computational frameworks.

Chapter 2 provides a literature review of wireless sensors, followed by a review of the smart sensors developed to date.

Chapter 3 provides a detailed description of the software and hardware characteristics of the Berkeley-Mote platform, which has been chosen for this research. In addition to the small physical size, low cost, modest power consumption, and diversity in design and usage, the main advantages of using the Berkeley-Mote platform is that both the software source code and hardware design for the smart sensors are open source.

In Chapter 4, description of the development of a new sensor board which implements a high sensitivity accelerometer is presented. Experimental results of the perfor-
mance of the accelerometer on the existing sensor boards of the Berkeley-Mote platform show that it has both poor low-frequency sensitivity and high noise density; therefore, the suitability for civil engineering applications is unclear. The new sensor board (named Tadeo) overcomes many of the deficiencies of the existing sensor. Finally, to address difficulties with aliasing of measured acceleration signals, the design of a modular four-pole Butterworth filter is presented.

In Chapter 5, an agent-based framework for SHM/C is presented which is based on the Gaia methodology proposed by Wooldridge et al. (2000). Smart sensors can be viewed as independent, intelligent entities, and therefore may be considered agents. The Gaia methodology was chosen because its approach is both general (i.e., is applicable to a wide range of MAS), and comprehensive (i.e. deals with both societal and agent levels of the system). Additionally, the Gaia methodology has been specifically tailored to the analysis and design of agent-based frameworks. An overview of the agent paradigm, including a brief historical background and description of its characteristics, classifications, and architectures, is addressed. Finally, a detailed description of the Gaia methodology is presented.

To illustrate the agent-based framework developed in Chapter 5, a SHM algorithm based on the statistical pattern recognition technique proposed by Sohn et al. (2001) is considered in Chapter 6. This approach is selected as the SHM algorithm for this study, because it does not require centralized collection of data, making it suitable for the agent framework. Following a brief summary of the diverse SHM algorithms used to date, the statistical pattern recognition technique is presented in detail. Finally, results from a simulation model are presented and compared with the corresponding centralized version of the SHM algorithm.
Finally, Chapter 7 summarizes the results drawn from the research presented in this dissertation. Possible future studies for more advanced networks are also presented.

The studies reported in this dissertation demonstrate the applicability of smart sensor technology for SHM/C systems based on an agent paradigm. This work provides an initial step toward the implementation of "smart" sensors in SHM/C systems for civil infrastructure.
CHAPTER 2

SMART SENSORS: LITERATURE REVIEW

While not a defining feature, all smart sensors to date have been wireless. Indeed, the deployment of densely-distributed networks using wired sensors does not represent a scalable approach. This chapter summarizes both prior research on wireless sensors and development of smart sensors for civil engineering applications.

2.1 Wireless sensors

Wireless global communication is important for facilitating low-cost, densely distributed sensing systems. Wireless radio links have been around for several years. Radio Frequency (RF) links have been utilized in embedded systems for numerous applications, including but not limited to cellular phones, home automation, digital audio players and wireless internet. Recently-developed inexpensive hardware has made it feasible to replace of the cabling in current vibration-based systems with RF links.

Westermo and Thompson (1997) presented a technology using peak strain sensors, which can be used to assess structural health. Their network consisted of three gauges, which, along with a digital junction, were installed on a three-story, wood-frame building. The system was powered by a 12-VDC battery pack; it was intended to routinely interrogate all sensors and store pertinent data or changes on each cycle. To transmit the informa
tion, the wireless system was connected to a cellular modem that was set to receive incoming calls from a PC for data downloading or reprogramming.

Pines and Lovell (1998, 1999) discussed an approach using sensors and wireless communication technology to monitor the health of large civil structures remotely using spread-spectrum wireless modems, data communication software, and conventional strain sensors. Their work described examples of condition-based health-monitoring systems that use cellular and through wire for data retrieval. A simple yet inexpensive device was realized and validated on a laboratory test structure at a range of up to approximately 1 mile without loss of communication signal.

Williams et al. (1998) presented a novel idea in which self-sufficient (i.e., generates its own power) wireless sensors were achieved. In their approach, the vibrational energy of the structure was used to power an accelerometer. The feasibility studies on reinforced concrete bridges indicated that the resonant frequency of the electric generator should match the fundamental frequency of the bridge so as to maximize the power generation.

Subramanian (1997) and Varadan et al. (1997, 1998, 1999, 2001) showed the wireless integration of MEMS and surface acoustic wave (SAW) devices employing interdigital transducers (IDT). These devices have a unique advantage in that they do not require an on-board power supply at the sensor location. The acceleration is measured when a wave (produced by a wave generator localized at the base station) is reflected by the sensor; the phase change in the reflected wave is proportional to the acceleration. This sensor has a wide dynamic range. The fabrication of the accelerometer is discussed. The wireless accelerometer provides an attractive opportunity to study the response of a “dummy” in
automobile crash tests and may be potentially useful in the deployment of “smart skins” (intelligent fuselage) for aircraft.

Krantz, et al. (1999) presented the Remotely-Queried Embedded Microsensor (RQEM). The objective of this research was to develop a microsensor that could retrieve data from embedded strain gauges. This system consisted of two main parts: the sensor package and the reader. The sensor package consisted of a microsensor (conventional strain gauge), signal conditioner, receiver/transmitter, data encoder, and power supply. The reader consisted of an external antenna coil attached to a Trovan RFID Tag Reader. The measurement occurred when the reader antenna was placed 3 - 12 inches from the embedded sensor.

Lemke (2000) described a remote vibration monitoring system integrated with the internet in order to acquire field data, which was then uploaded to a web server using a wireless connection. The selection of the ground motion transducer with respect to the desired frequency response was discussed. The network was wired, but the transmission from the field site was performed by cellular telephony. Battery power considerations were also studied and the results showed that the system could be dialed just over 5000 times. With a peak transmission every thirty minutes, the system could last for over 200 days.

Oshima et al. (2000) also presented a monitoring system that could be interrogated via a mobile telephone. This system consisted of a photocell, an accelerometer, and a displacement sensor. The sampling frequency was 200 Hz. Experimental results for the structural frequencies and mode shapes were presented that closely agreed with the analytical results. A comparison between a fiber-optic strain sensor and a standard strain gage for
crack propagation was presented. A difference of 5–10% in strain measurements between theses sensors was found.

Wang and Liao (2001) presented a wireless signal retrieval system. The difference between this application and the traditional one used is the way that the signal acquisition and transmitting subsystems were tuned to different resonant frequencies through frequency modulation (FM) technology before transmitting. Specifically, the wireless transmitter subsystem composed mainly of the following units: the sensor signal processor, the voltage/frequency (V/F) converter, and the transmitter. The wireless receiver subsystem was mainly composed of the following units: the receiver, the signal processor, the F/V converter, and the low pass filter. An example of a transmission of a sinusoidal wave was presented. The received signal yielded the same frequency content, but the amplitude was increased. The authors claimed that for certain algorithms, this increase in the amplitude of the received signal was not a factor. Additionally, they compared the power spectra density of a white noise random signal transmitted through a conventional TX2/RX2 with that of their wireless system, suggesting that reception of a signal in the range 0–5Hz suffered a sharp decrease in reception and showing that the conventional paradigm was not likely to be effective for monitoring of civil infrastructure.

Evans (2001) provides a very good compendium of the various alternatives that can be used for wireless transmission of data, including free bandwidth frequencies, such as 915 MHz and 2.45 GHz, cellular phone lines, two-way paging, and satellites services. A description of the available sensors, such as the micromachined and force balance accelerometers, is also provided. The author indicates the performance and cost of each one of the wireless devices. Finally, two examples of wireless networks are presented: an applica-
tion to a highway bridge used to determine damage, and free-field measurements to produce a real-time seismicity map for Oakland, California.

Mita and Takahira (2001) presented a wireless peak strain and displacement sensor. This sensor consisted of a variable capacitor made of an outer cylinder and an inner cylinder, in which the capacitance depended on the overlapping length. In order to retrieve data, an inductor was added to the variable capacitor, creating a resonant circuit. This circuit was excited by a dip meter and a frequency was read (a dip meter is the equipment which measures the frequency of the resonance circuit). A comparison of measurement results between a laser sensor and the peak strain sensor was presented. The agreement of these measurements assured the feasibility and accuracy of the system.

2.2 Smart sensors

Some of the first efforts in developing a smart sensor for civil engineering applications were presented by Straser and Kiremidjian (1996, 1998), Straser et al. (1998), and Kiremidjian et al. (1997). This research sought to develop a near real-time damage diagnostic and monitoring system which evaluated both extreme and long-term structural health. Two types of monitoring systems were identified: (i) extreme event, and (ii) long-term monitoring systems. Damage detection methods were further categorized into global and local methods. Several damage detection methods were discussed, as well as strategies for optimal sensor placement. The authors noted that one complicating aspect of long-term monitoring indicated that the characteristics of a structure could vary significantly due to environmental changes such as loading, boundary conditions, temperature, and humidity. The hardware was designed to acquire and manage such data, while the software was designed to facilitate damage detection diagnosis. The proposed network provided
ease of installation, low per unit cost, portability, and broad functionality. The sensor unit consisted of a microprocessor, radio modem, data storage, and batteries. One of the problems in having many sensors trying to communicate simultaneously with the base is time synchronization of the signals. To solve this problem, a second AM type radio was implemented to perform synchronization. The authors found that the time delay for this approach was 0.05 millisecond, which for the frequency spectrum of interest, namely 1-50Hz, represented a maximum phase delay of less than one degree. To save battery life, the sensor unit remained in sleep mode most of the time, periodically checking its hardware interrupt to determine if there were external events that require attention. The analysis software determined the maximum inter-story drift ratio over the entire time history, as well as the cumulative normalized Arias Intensity to measure of the total kinetic energy of each floor. A damage detection algorithm called DIAMOND was developed in MATLAB.

Maser et al. (1997) proposed the Wireless Global Bridge Evaluation and Monitoring System (WGBEMS) to remotely monitor the condition and performance of bridges. WGBEMS used small, self-contained, battery operated transducers, each containing a sensor, a small radio transponder, and a battery. The complete system consisted of a local controller placed off a bridge with several transducers distributed throughout the bridge. The data collection at the transducer involves signal conditioning, filtering, sampling, quantization, and digital signal processing. The radio link used a wide band in the 902 to 928 MHz range.

Agre et al. (1999) presented a prototype wireless sensor node called “AWAIRS I” (see Fig. 2.1). This smart sensor could support bidirectional, peer-to-peer communications with a small number of neighbors. The current device consists of a processor, radio, power sup-
ply and sensors (seismic, magnetic and acoustic). Multiple portals for transporting information in or out of the sensor network can be established. The authors discussed some of the networking problems in a wireless sensor network, which include limited battery energy, size of the overhead of the messages communication protocols, and non-real-time delivery, among others. This prototype will run approximately 15 hours continuously on two 9V batteries. The time-division multiple access (TDMA) scheme used allows nodes to turn off their receiver and/or transmitter when they are not scheduled to communicate. This research is still in a development phase.

Brooks (1999) delineates four generation of sensors. The first-generation devices of the 1960’s consisted of unamplified sensors. The second generation was comprised of internally amplified designs. The third generation were devices that used mixed-mode analogue and digital transmission through on-board electronics, and the fourth generation migrates some of the processing to the sensor board. The fourth generation is characterized by a number of attributes: bi-directional command and data communication, all digi-
tal transmission, local digital processing, pre-programmed decision algorithms, user-defined algorithms, internal self-verification/diagnosis, compensation algorithms, onboard storage, and extensible sensor object models. An important issue in this generation is the battery life of the sensors. In order to conserve power it is necessary to put the sensor in an ultra low-power sleep mode, only communicating with the base station when the internal log file is full. With this type of procedure the battery life of the remote sensor can be extended up to two years.

Mitchell et al. (1999) presents a wireless data acquisition system for health monitoring of smart structures. The authors developed a microsensor that uses an analog multiplexer to allow data from multiple sensors to be communicated over a single communication channel. The data is converted to a digital format before transmission, using an 80C515CO microcontroller. A 900 MHz spread spectrum transceiver system, capable of transmitting serial data at the rate of 50Kbps, is used to perform the wireless transmission, over a range of approximately 0.25 miles. Damage can be detected via variations in the natural frequencies of the structure. The system employs the Numerical Algorithms for Subspace State Space System Identification (NASID) method. The main advantage of this algorithm is that it is non-iterative and does not involve nonlinear optimization. This health monitoring system has been applied to a cantilever beam in which a loss of mass represents the damage of the structure. Mitchell et al. (2001) have continued this work to extend the cellular communication between the central cluster and the web server, allowing web-control of the network. The proposed Web-Controlled Wireless Network Sensors (WCWNS) consist of two main parts: the wireless network sensors and the web interface.
Building on the work of Kiremidjian et al. (1997), Lynch et al. (2001) demonstrated a proof-of-concept wireless sensor that used a standard integrated circuit component. This unit consists of an 8-bit Motorola 68HC11E1 microcontroller with a 3MHz CPU that can accommodate a wide range of analog sensors. The systems communicate via direct sequence spread spectrum radio multiplied by a pseudo noise spreading sequence (also known as a chirp code). This approach allows multiple users to access the same bandwidth simultaneously without interference. For the spread-spectrum modems to operate properly, both the sending and receiving modems must be self-synchronized and follow a prescribed sequence of frequencies. Some units use the ADXL210 accelerometer along with a duty cycle modulator that provides a 14-bit output with an anti-aliased digital signal. In other units, a high performance planar accelerometer is used along with a 16-bit A/D converter. This accelerometer has a resolution of 20 µg at a bandwidth of 650 Hz. The whole system can be accommodated within a sealed package with a roughly size of 5” by 4” by 1” (see Fig. 2.2). The sensor unit was validated through various controlled experiments in the laboratory. Kiremidjian et al. (2001) indicates that pushing data acquisition and computation forward is fundamental to the smart sensing and monitoring systems but represents a radical departure from the conventional instrumentation design and computational strategies for monitoring civil structures.

Liu et al. (2001) presents a wireless sensor system that includes 5 monitoring stations, each of them with a 3-axis accelerometer (ADXL05). These stations use an 80C251 microprocessor with a 16-bit A/D converter. Because this network is sensing continuously, transmission of data to the base station could present collisions, caused by two or more sensors sending information to the base station at the same time. To avoid this problem, a
direct sequence spread spectrum radio with long pseudo-noise code has been used to distinguish each substation. The radio signal operates at 900 MHz with a maximum data rate of 57.6 Kbps in half duplex mode. Because of the half duplex operation, to obtain data from each station, the master station needs to switch from transmit mode to receiver mode, and each substation will switch from receive mode to transmit mode once. Each transition needs about 10-20 ms; the authors were able to reduce this time in half by implementing an advanced protocol. Experimental verification was provided.

The objective of the recently-created European project of Energy Efficient Sensor Networks (EYES 2002) is to develop the architecture and technology that will enable the creation of a new generation of self-organizing and collaborative sensors. These sensors will be capable of effectively networking together, in order to provide a flexible platform to support a large variety of mobile-sensor network applications.
This 3-year project has the support of Alcatel Center Information and Technology (Alcatel 2004), one of the most important communication solution providers in Europe, with experience in end-to-end networks that will boost reliable communication between sensors.

The architecture of EYES is supported by structure on two levels. The first level deals with the sensors and the network, i.e., internal sensor architecture, distributed wireless access, routing protocols, reliable end-to-end transport, synchronization and localization of nodes. The second level provides distributed services to the application, deals with information collection, lookup, discovery and security. Figure 2.3 shows a sensor prototype of the EYES project.

EYES will make use of the effort invested in the DataGrid project (DataGrid Project, 2004). The objective of the DataGrid project is to build the next generation of computing infrastructure, providing intensive computation and analysis of shared large-scale data-

![Prototype smart sensor (EYES project 2002).](image)
bases. This project includes more than 12 WorkPackages (WP) that deal with middleware, applications and management.

Specifically, EYES will use WP1, WP2, WP3 and WP4. WP1, system architecture (WorkPackage 1, 2004), aims to produce an open framework for flexible development of new applications. WP2, data management (WorkPackage 2, 2004), has been designed to manage and share Petabyte-scale ($2^{50}$ bytes) information volumes. WP3, distributed services (WorkPackage 3, 2004), deals with the service layer, which supports mobile sensor applications. Finally, WP4, proof-of-concept (WorkPackage 4, 2004), whose deliverables will be a proof of concept network that uses more than 100 nodes.

While limited technical information has been provided to the public, EYES is definitely something to watch for in the near future.

2.3 Summary

This chapter has presented prior research on and development of wireless and smart sensors, which has followed a variety of approaches. Most of the devices were designed using commercial-off-the-shelf (COTS) components. Advantages and disadvantages can be identified for each unit developed; an important drawback, however, is that all these devices are proprietary. There is a recognizable need for smart sensors with an open software/hardware platform. As described in the next chapter, the recently developed Berkeley-Mote platform provides such an open environment.
CHAPTER 3

BERKELEY-MOTE PLATFORM

This research employs the Berkeley-Mote platform because it is an open hardware/software, smart-sensing platform with a large user community (see http://webs.cs.berkeley.edu/tos/). By adopting the Berkeley Mote platform, we can leverage the substantial efforts of other researchers to develop workable smart sensor networks that focus on the application at hand, i.e., monitoring and control of civil infrastructure systems. This chapter provides a detailed description of the software and hardware characteristics of the Berkeley-Mote platform.

3.1 Berkeley-mote platform

The Berkeley-Mote platform is an open hardware/software platform for smart sensor technologies that has been developed with substantial funding from the Defense Advanced Research Projects Agency (DARPA) of the US Government. The goal of this research is to develop Smart Dust, or Motes, a low-cost, fully autonomous system within a cubic millimeter volume, that can allow for the realization of dense sensor arrays. The Mote paradigm consists of four basic components: power, computation, sensors, and communication. It is capable of autonomy and interconnection with other Motes. The main advantages are their small physical size, low cost, modest power consumption, and diversity in design and usage.
Although the final goal of the Smart Dust project is to realize a smart sensor within a cubic millimeter, the project was initiated with the use of the commercial-off-the-shelf components (COTS). COTS Dust has all of the basic functionality of Smart Dust, but because the devices were built in a tenth of the time, instead of being a cubic millimeter in size, they were a few cubic inches in size. COTS Dust could serve as a platform to run a variety of algorithms to test various behaviors that Smart Dust would exhibit, such as network formation, environmental sensing and data transmission among others.

The first devices (Hollar 2000) were designed at the University of California at Berkeley by Prof. Kris Pister. The second generation of Motes, called Rene, implemented a modular construction, allowing the use of one unique base with the possibility of various interchangeable sensors. The next generation, called Mica, has a larger memory capacity. The latest version, called Mica2, and smaller size processing board, called Mica2Dot, are described in the following section.

3.2 Hardware features

The sensor design is comprised of two main parts: the processor/radio board (Mica2, or Mic2Dot), and the sensor board (comprised mainly of MEMS). As compared to the Mica, the Mica2 and Mica2dot, shown in Fig. 3.1, improved the radio communication (with a tunable frequency radio), and the microprocessor unit (7.3728 MHz). A summary of their characteristics is presented in Table 3.1.

The Mica2’s processor is the ATmega128. The microprocessor can be configured for three different sleep modes: (i) idle, which just shuts off the processor; (ii) power down, which shuts off everything but the watchdog and asynchronous interrupt logic necessary
### TABLE 3.1

CHARACTERISTICS OF THE MICA2 AND MICA2DOT PROCESSOR BOARDS

<table>
<thead>
<tr>
<th>Performance</th>
<th>Mica2</th>
<th>Mica2dot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash memory</td>
<td>128K bytes</td>
<td>128 K bytes</td>
</tr>
<tr>
<td>Measurement memory</td>
<td>512K bytes</td>
<td>512K bytes</td>
</tr>
<tr>
<td>EEPROM</td>
<td>4K bytes</td>
<td>4K bytes</td>
</tr>
<tr>
<td>A/D (Channels)</td>
<td>10 bits (8)</td>
<td>10 bits (6)</td>
</tr>
<tr>
<td>Center Frequency</td>
<td>433 868/916MHz</td>
<td>433 868/916MHz</td>
</tr>
<tr>
<td>Num. of channels of RF</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Data rate</td>
<td>38.4 K baud</td>
<td>38.4 K baud</td>
</tr>
<tr>
<td>Outdoor range</td>
<td>300 m.</td>
<td>300 m.</td>
</tr>
<tr>
<td>Size</td>
<td>6x3x1 cm</td>
<td>2.5x0.6 cm</td>
</tr>
</tbody>
</table>

Figure 3.1: Berkeley-mote (Mica2 and Mica2dot) processor boards.
to wake up; and (iii) power save, which is similar to the power down mode but leaves an asynchronous timer running. At peak load, the current system can run about 30 hours on two AA batteries. In the idle mode, one set of batteries can last for up to a year. The radio consists of a True single chip UHF RF transceiver (CC1000) with frequency range of 300-1000 MHz that can operate at speeds up to 76.8 K baud. This design allows, through an internal universal asynchronous receiver-transmitter (UART), the versatility to connect different integrated circuits; i.e., the modularity to support different types of sensors.

The Mica2 processes the information obtained by the sensors. Some of its tasks include analog/digital conversion and sampling. The microprocessor also permits on-board programming. After processing the data, the information is sent through a radio frequency device that can use a 433/926 MHz unlicensed broadcasting frequency.

The current sensor board designs for the Mica2 platform include MTS101CA, MTS300CA and MTS310 (see Figs. 3.2, 3.3 and 3.4 respectively). Table 3.3 provides a

Figure 3.2: MTS101CA sensor board.

Figure 3.3: MTS300CA sensor board.
### TABLE 3.2

**ELECTRONIC CHARACTERISTICS OF THE SENSORS USED BY THE SENSOR BOARDS**

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Number</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Photo resistor</td>
<td>Clairex CL9P4L</td>
<td>Simple voltage divider design</td>
</tr>
<tr>
<td>Thermistor</td>
<td>ERT-J1VR103J</td>
<td>10kohm, 1°C accuracy, -40 to 125°C</td>
</tr>
<tr>
<td>Bi-axial accelerometer</td>
<td>ADXL202</td>
<td>2 Axis, +/- 2g, at 2mg resolution, Noise level 200 µg/√Hz</td>
</tr>
<tr>
<td>Acoustic sensor</td>
<td>National Semiconductor LMC567CM</td>
<td>Phase lock loop and adjustable amplitude threshold support to detect tone. Adjustable tracking frequency</td>
</tr>
<tr>
<td>Acoustic actuator</td>
<td>Sirius PS14T40A</td>
<td>Piezoelectric single tone buzzer. Resonant at 4kHz +/- 0.5kHz</td>
</tr>
<tr>
<td>Magnetometer</td>
<td>Axis Honeywell HMC1002</td>
<td>Range +/-6 gauss (earth’s field +/-0.5 gauss), 27µgauss at 10Hz</td>
</tr>
</tbody>
</table>
description of the sensors that are implemented on the sensor boards. Table 3.2 describes
the electronic characteristics of the sensors used by the sensor boards described above.

Ultimately, the user could design and manufacture a tailored sensor board according
to the specific application; however, certain electronic constraints should be observed.
Researchers can obtain the designs for the current sensor board and the Mica2 and
Mica2Dot from Crossbow Technology, Inc. (www.xbow.com).

Finally, it is worth mentioning the existence of a sensor board, part of the CotsBots
project (Bergbreiter & Pister 2003), that is capable of controlling DC motors used to drive
a mobile robot (see Figs. 3.5 and 3.6). The motorboard employs a pulse with modulation
(PWM) technique to control analog devices using digital signals. This control technique is
very efficient, producing very low levels of noise, since the signal is kept digital.

![Figure 3.5: Motorboard (Bergbreiter & Pister 2003)](image)
This motorboard was tested to control a magnetorheological (MR) friction damper successfully. Together with the Cotsbots project, the Mica2 platform can then be considered as a tool for controlling actuators.

### 3.3 Software description

The Mote software is middleware designed to eliminate the gap between raw hardware capabilities and a useful system, achieving network communication based on the concept of reconfigurable networks which use active messages (AM). The work presented by Hill (2000) and Hill, *et al.* (2000) describes the design of a tiny event-driven operating system that provides support for efficiency, modularity, and concurrency-intensive operation. This operating system, called TinyOS, fits in 178 bytes of memory. TinyOS is used in a networked sensor with the following characteristics: small physical size and low power consumption, concurrency-intensive operation, limited physical parallelism and controller hierarchy, diversity in design and usage, modularity, and a robust operation. TinyOS is designed to scale with current technological trends, supporting smaller, more tightly integrated designs, as well as the substitution of software components into hardware.
A complete system configuration for TinyOS consists of a small scheduler that links components. A component has four interrelated parts: a set of command handlers, a set of event handlers, an encapsulated fixed-size frame, and a bundle of simple threads. Threads, commands, and handlers execute in a context of the frame and operate on its state. To facilitate modularity, each component also declares the commands it uses and the events it signals. These declarations are used to compose the modular components in a per-application configuration. The composition process creates layers of components where higher level components issue commands to lower level components and lower level components signal events to the higher-level components. Physical hardware represents the lowest level of components. The entire system, including libraries and applications, is written in nesC, a language for programming structured, component-based applications, with a syntax similar to the C programming language. The latest operating system software can be downloaded from webs.cs.berkeley.edu/tos/.

3.4 Summary

This chapter provided a detailed description of the software/hardware characteristics of the Berkeley-Mote platform. Smart sensors based on this platform provide the impetus for the development of the next generation of SHM/C systems.

Although the Berkeley-Mote platform offers a variety of sensor boards with a diversity of sensors, these boards are not adequate for all applications. Preliminary results, shown in Chapter 4, demonstrate that the standard accelerometer for the Mote platform, ADXL202E, does not perform well in the low-frequency range, where most civil structures respond.
CHAPTER 4

HIGH SENSITIVITY ACCELEROMETER

In this chapter, the development of a new sensor board that implements a high sensitivity accelerometer is presented. Experimental results for the performance of the accelerometer currently available for the Berkeley-Mote platform show both poor low-frequency sensitivity and high noise density; therefore, their suitability for civil engineering applications is unclear. The new sensor board (named Tadeo) overcomes many of the deficiencies of the existing sensor. To address concerns with aliasing of measured acceleration signals, the design of a modular four-pole Butterworth filter is also presented.

4.1 ADXL202 accelerometer

The ADXL202E accelerometer (Fig. 4.1) is a low-cost, low-power, complete 2-axis capacitive accelerometer with analog and digital output, all on a single monolithic integrated circuit. The ADXL202E measures accelerations with a full-scale range of $\pm 2$ g. These measurements can be both dynamic acceleration (e.g., vibration), and static acceleration (e.g., gravity). The outputs are analog.
voltage or digital signals whose duty cycles (ratio of pulsewidth to period) are proportionate to acceleration. Table 4.1 presents the main ADXL202E’s electronic characteristics. The ADXL202E accelerometer is implemented on the sensor board MTS310CA presented in previous chapter (see Fig. 3.4).

**TABLE 4.1**

CHARACTERISTICS OF THE ADXL202E ACCELEROMETER

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input range</td>
<td>±2</td>
<td>±2</td>
<td>±2</td>
<td>g</td>
</tr>
<tr>
<td>Frequency Response</td>
<td>0-50</td>
<td>0-50</td>
<td>0-50</td>
<td>Hz</td>
</tr>
<tr>
<td>(nominal, 3dB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensitivity</td>
<td>140</td>
<td>167</td>
<td>195</td>
<td>mVolts/g</td>
</tr>
<tr>
<td>Output Noise (RMS)</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>µg/√Hz</td>
</tr>
<tr>
<td>Operating Voltage</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>Volts</td>
</tr>
<tr>
<td>Operating Current</td>
<td>0.6</td>
<td>1.0</td>
<td>1.0</td>
<td>mA</td>
</tr>
</tbody>
</table>

To assess the efficacy of the ADXL202E accelerometer for civil engineering applications, its performance was compared with a PCB model 393B04 (PCB Piezotronics, Inc. 2004) high-sensitivity piezoelectric seismic accelerometer (see Fig. 4.2 and Table 4.2). The 393B04 reference accelerometer was recently calibrated by the manufacturer, so its performance is well known.

### 4.2 Experimental setup

Both the ADXL202E and the 393B04 were mounted on a UCIST Shaking Table (Quanser Consulting Inc.) (Fig. 4.4), which is capable of providing a peak acceleration of ±2.5g with a maximum stroke of ±3 in. To control the shaking table, as well as the data
acquisition systems for the 393B04 accelerometer, a DSPT SigLab 20-42 box was used (Spectral Dynamics, Inc.). This data acquisition system has a 20-bit sigma-delta analog to digital (A/D) converter with a 90 dB antialiasing filter. The data acquisition system of the Berkeley-Mote platform employs a 10-bit A/D converter with no antialiasing filter; however, the MTS310CA has a one-pole antialiasing filter with a 50 Hz cutoff frequency.

The Mica2 was placed on the programming board connected to the serial port. The oscilloscope application (TinyOS version 0.6) was downloaded to the Mica2. For the data
acquisition, the Java interface SerialForward and Oscilloscope were employed. Both
acquisition systems, (i.e., SigLab and TinyOS) were set to sample at 256 Hz.

Figs. 4.4 and 4.5 show a comparison in the time and frequency domain between the
accelerometers when subjected to relatively high amplitude random acceleration of 41.6
mg rms.

As shown in Figure , the response in the time domain of the ADXL202E accelerometer
with respect to the 393B04 is reasonably good. In the frequency domain (see Fig. ), the
responses between the two accelerometers are in good agreement above 1.5 Hz. However,
the noise floor for the ADXL202E is quite high below 1.5 Hz. This effect may be due in
part to limitations imposed by the Mica2's 10-bit A/D converter.

The performance of the ADXL202E at low amplitudes was also investigated. Figures
4.6 and 4.7 show a comparison in the time and frequency domain between the accelerometers when subjected to an amplitude 4.9 mg rms excitation.
Figure 4.4: Time domain plot acceleration response for a random excitation.

Figure 4.5: Power spectra density of the acceleration for a random excitation.
As shown in Fig. 4.6, the response in the time domain of the ADXL202E accelerometer compared with the reference accelerometer is quite poor. This limitation is primarily due to the low sensitivity of the ADXL202E accelerometer, as well as the lower resolution of the Mica2S's 10-bit A/D converter. In the frequency domain (see Fig. 4.7), the responses between the two accelerometers do not correlate well. The noise floor for the ADXL202E is high below 2 Hz.

In the next section, selection of a high sensitivity accelerometer and its implementation for the Mica2 platform are presented.

4.3 High sensitivity accelerometer

A high sensitivity accelerometer, with good low-frequency response as well as low noise density was sought. The SD-1221 accelerometer produced by Silicon Designs, Inc. (Silicon Designs, Inc.) was identified as having performance specifications that mesh well with the needs of civil engineering applications. The main characteristic of this accelerom-
eter is its low noise floor, $2.0 \, \mu g/\sqrt{Hz}$, which is 100 times lower than that of the ADXL202E. In addition, the sensitivity of the SD-1221 is $1000\,mV/g$, which is around 6 times greater than the sensitivity of the ADXL202E (see Table 4.3).

However, the power requirements (5 volts) for the SD-1221 were beyond the capabilities of the Mica2 platform (3 volts). The approach taken to overcome this difficulty was to boost the power from the Mica2 to the required level. The component MAX-1682 (see Fig. 4.8), manufactured by Maxim Integrated Products, Inc. (2004), possesses the necessary features. The MAX-1682 is a switched-capacitor voltage doubler that increases the power from 3.0V to 6.0V. In addition to this component, a regulator was sought to reduce voltage to 5.0V and eliminate the noise of the voltage doubler. The LT1761ES5-5 low-noise, low-dropout (LDO), linear regulator (see Fig. 4.9), manufactured by Linear Technology Corporation (2004), regulates any input over 5.5V providing an output ranging from 4.935 to 5.065 V. Finally, the output of the linear regulator was provided to the SD-1221 high sensitivity accelerometer. A first prototype was manufactured at the Electronics

Figure 4.7: Power spectra density of the acceleration for a random excitation.
### TABLE 4.3
CHARACTERISTICS OF THE SD-1221 ACCELEROMETER

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input range</td>
<td>±2</td>
<td>±2</td>
<td>±2</td>
<td>g</td>
</tr>
<tr>
<td>Frequency Response (nominal, 3dB)</td>
<td>0-200</td>
<td>0-400</td>
<td>0-400</td>
<td>Hz</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>mVolts/g</td>
</tr>
<tr>
<td>Output Noise (RMS)</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>µg/√Hz</td>
</tr>
<tr>
<td>Operating Voltage</td>
<td>4.75</td>
<td>5.0</td>
<td>5.25</td>
<td>Volts</td>
</tr>
<tr>
<td>Operating current</td>
<td>10</td>
<td>14</td>
<td>14</td>
<td>mA</td>
</tr>
</tbody>
</table>

Figure 4.8: MAX1682 Voltage doubler, Maxim Integrated Products, Inc. (2004)

C1 = C2 = 10 µF

Figure 4.9: LT1761ES5-5 Linear regulator (Linear Technology Corporation, 2004)
After demonstration of the proof of concepts using the first prototype, a final sensor board was designed. The sensor board layout was based on the MTS310CA (see Figure 3.4). From this design, light, temperature, and acoustic sensors were kept, and the SD-1221 high sensitivity accelerometer was implemented. The printed circuit board was designed using the Protel DXP program (2003), and it was manufactured at Fineline Circuits & Technology (2004). The sensor board was named “Tadeo” (see Figure 4.11). Connections diagrams of its top and bottom layers are shown in Figures 4.12 and 4.13 respectively.

To assess the accuracy of the measurements, the SD-1221, ADXL202E, and 393B04 accelerometers were mounted on the Quanser shaking table and subjected to random excitation with amplitude of 2.6 mg rms. Figures 4.14 and 4.15 show a comparison of the response of the three accelerometers in both the time and frequency domains.

As shown in Figure 4.14, the response of the SD-1221 follows the response of the 393B04 accelerometer. The high frequency component in the response of the SD-1221 is
Figure 4.11: Tadeo sensor board.

Figure 4.12: Tadeo’s top layer connection diagram.
Figure 4.13: Tadeo’s bottom layer connection diagram.

Figure 4.14: Time domain plot of acceleration response for a random excitation.
due to the lack of an antialiasing filter on this accelerometer. In the frequency domain (see Fig. 4.15), the SD-1221 matches the response of the 393B04 accelerometer above 1.5 Hz. While the performance of the SD-1221 below 1.5 Hz is substantially better than the ADXL202E, we have observed some differences which likely result from the sensor's lack of an antialiasing filter and the limitations of the lower 10-bit A/D converter of the Mica2.

4.4 Four-pole Butterworth anti-alias filter

To address concerns with aliasing of measured acceleration signals, the design of a modular four-pole Butterworth filter is presented. This filter will also help to suppress the high-frequency content of the signal, as shown in Figure 4.14. The design and test of the filter was possible thanks to the support of Mr. Tomonori Nagayama.

The Sallen & Key topology was used for the design of the Butterworth filter. This topology utilizes an active approach, since it makes use of operational amplifiers. Active
filters can have virtually any arbitrary gain and are easier to design. Figure 4.16 shows the basic second-order Sallen & Key active filter topology. The final design of the filter consisted of cascading two second-order filters. Also, a unity gain was selected, since the output of the SD-1221 accelerometer does not required amplification. Figure 4.17 shows the final schematic connection diagram.
The Butterworth filter board has been designed with a cutoff frequency of 50Hz. (Note that, depending on application requirements, filters with other cutoff frequencies can be easily constructed.) The final printed circuit board was manufactured at the Electronics Services Shop (ESS, 2004), and it is shown in Figure 4.18.

To test the performance of the filter, the high-performance SD-1221 accelerometer board is compared with the 393B04 reference sensor, with and without the filter. Figures 4.19 and 4.20 show the measured acceleration and the power spectral density representation of the measured acceleration for a broadband random input excitation. The effectiveness of the antialiasing filter is clearly demonstrated: in the time domain the high frequency content of the record is removed; whereas in the frequency domain, it shows agreement with respect of the 393B04 reference sensor. However, divergence can be appreciated from 0 - 1Hz, which may be the result of the limitation of the A/D of the Mica2.
Figure 4.19: Time domain plot of acceleration response for a random excitation.

Figure 4.20: Power spectra density of the acceleration for a random excitation.
4.5 Summary

In this chapter we have presented the development of a sensor board that implements a high sensitivity accelerometer. The need for the development of a new sensor board was based on the inadequate performance of the ADXL202E accelerometer, available on the Berkeley-Mote Mica2’s MTS310CA sensor board. Results showed that the ADXL202E has poor low-frequency sensitivity and high noise density, making its suitability for civil engineering applications unclear. The Tadeo sensor board overcomes these deficiencies.

Experimental results show that the SD-1221 accelerometer performs suitably for civil engineering applications. Although the SD-1221 accurately measures the low frequency/amplitude signals, the limitations of the Mica2’s 10-bit A/D converter, as well as the lack of an antialiasing filter on the sensor, hindered its performance.

A four-pole Butterworth antialiasing filter has been designed to address this problem. Experimental results demonstrate an improved recorder signal; however, the effect of the 10-bit A/D converter still limited its performance. The need for a better A/D converter for the Mica2 platform has been identified and will be proposed for development in futures studies.
In this chapter, the development of an agent-based framework for SHM is presented. The lack of an appropriate framework for smart sensors hinders the potential of their use. Agents are proposed as the basic element of such a framework. To understand the ideas behind this paradigm, a brief summary regarding Artificial Intelligence (AI), the precursor of agents, is presented. Later, a definition of agent is introduced. Finally, based on the Gaia methodology, an agent framework to support SHM algorithms, is proposed.

5.1 Introduction

A smart sensor, for all intents and purposes, consists of two components: a computation/radio transmission component and a sensing component. Another way of looking at these components is that one is intelligent and the other mechanical. In this type of system (i.e., one containing an intelligent part), the mechanical processes are often slow compared with the speed of computation. Typical algorithms used to interconnect these elements have limited success. Nevertheless, agent-based approaches take advantage of these differences in speed and use highly efficient algorithms that fulfill the proposed goals of the system or systems.
The history of agents can be traced to research on artificial intelligence (AI), object-oriented programming, and concurrent object-based systems, as well as human-computer interface design (Jennings, et al. 1998).

5.2 Artificial intelligence

The term “artificial intelligence” was first coined in 1955 by John McCarthy (John McCarthy et al. 1955), who proposed a Summer Research Project on Artificial Intelligence that took place at Dartmouth College in Hanover, New Hampshire, in 1956. From there, the term AI has changed over time. The American Association for Artificial Intelligence define AI as: “the scientific understanding of the mechanisms underlying thought and intelligent behavior and their embodiment in machines”.

Tecuci (1998) explains that in the beginning of AI, researchers expect that most of the characteristics associated with intelligence in human behavior would be replicated by AI, including the perception of the environment through artificial eyes and ears, communication in a natural language, reasoning, solving of problems, etc. The development and realization of this “complete” intelligent system was far more difficult than anticipated. Researchers historically tended to focus separately on the various components of intelligent behavior, resulting in significant specialization in the field. This new problem division led to several parallel researches that included natural language processing, theorem proving, planning and problem solving, learning, and vision. Jennings et. al (1998) indicates that these specializations were more likely to succeed in the goals of AI, and that the creation of an intelligent artifact would be straightforward.

McCarthy (2003) (who was the first to coin the term AI), presents a number of the branches of AI, with the understanding that some of them may be not included, and also
that some may be regarded just as concepts or topics rather than full branches. These in-
clude:

- Logical AI: Represents the knowledge of the world, goals, and current situations by sentences in logic. The program decides the best course of action by inferring that a certain action or course of action is appropriate to achieve the goals. (McCarthy, 1959).

- Search: Given that many AI programs need to examine a large number of possibilities in order to solve a problem, this concept focuses on the develop of efficient ways to accomplish it.

- Pattern recognition: This branch tries to look for patterns of observations made. These observations can be optical, verbal, etc. The objective is to research methods that can recognize these patterns.

- Representation: The study of mathematical languages that can represent the facts of the world.

- Inference: Given some facts, others can be inferred. Inference can be accomplished by logical deduction or non-monotonic inference. Non-monotonic inference occurs when a conclusion is obtained by default and later can be withdrawn if there is evidence to the contrary.

- Common sense knowledge and reasoning: This area tries to develop human skills related to common sense and reasoning. Although it has been actively researched since 1950, advances are far from human levels.

- Learning from experience: Based on previous experience, programs can learn. Approaches, such as connectionism and neural networks specialize in this process. However, programs can only base their learning on facts and behaviors that their formalisms can represent.

- Epistemology: Epistemology is the branch of philosophy that studies knowledge. In the context of AI: given a problem, the science of epistemology involves acquiring the knowledge needed to solve it.

- Ontology: Ontology is the study of the diverse set of categories (with their relationships) that the program is able to use in order to classify and predict its environment. It specifies, from the program point of view, what there is in the world, what the properties and relations of things are, what kind of processes and events are involved, what the causal connections are, what the laws are, etc. (Gruber 2004).
• Heuristics: heuristics is a method based on empirical information that has no explicit rationalization. Heuristic AI is a way of trying to discover something or an idea embedded in a program. For example, a heuristic function might be one that estimates how far a node in a search tree seems to be from a given goal.

• Genetic programming: A technique in which genetic algorithms are used to solve a task. Generic programming provides a method for automatically creating a working computer program from a high-level problem statement of the problem.

• Planning: Given general facts about the world and a stated goal, this research seeks to generate strategies for achievement that goal, that in most cases, is just a sequence of actions.

Probably, the first activity more closely related to agents was AI planning. AI planning research addressed the question of knowing what to do (i.e. what action to have a machine perform). The research done in this field during the 1970s and early 1980s was primarily focused on the representations required for actions and the planning algorithms themselves. These algorithms represent the environment through a series of symbolic representations and reasonings. Some of the first research in this area, known as symbolic AI, involved simulating small worlds and the algorithms that act and react on them. However, these scenarios were not scalable, and the reaction time, or calculative rationality, was identified as a big problem (calculative rationality is the time that elapses from the moment that the algorithm receives the input to the moment that it generates an output action). Delays are attributed to the fact that the inference machines of the planning algorithms encounter a situation that was not foreseen when they were designed; therefore, the search for the solution will correspond to a delay that depends on the complexity of the task to be solved. Consequently, time-constrained problems could not be solved using these techniques, hence a new set of reasoning approaches for planning were sought.
Liu, et al. (2001) summarized the shortcomings of traditional (symbolic) AI in this way: “They rely on human beings to plan the exact steps for transforming and solving the problem, and for carefully distributing the task to individual AI systems”.

According to Jennings et al. (1998), the best known critic of the symbolic AI models used in planning algorithms was Rodney Brooks. Brooks presented a sequence of papers pointing out objections to symbolic AI, and proposed various alternative research programs known as behavioral AI, reactive AI, or situated AI (Brooks 1986, 1991a and 1991b). The research program of Brooks laid out the subsumption architecture that employed no symbolic representations or reasoning, but rather was based on a collection of task accomplishing behaviors. Brooks’ theories were used to develop a reactive paradigm in which actions are taken based on the local information available from the environment. The advantage of this approach is that no calculative rationality is required; therefore, time constraint problems may be more readily solved. Nevertheless, purely reactive paradigms are not suitable for all the problems. A combination of symbolic reasoning with reactive paradigm seems to be the most appropriate, as it takes advantage of the virtues of both approaches. This paradigm is termed a hybrid architecture.

In the next section, the basic aspects of agents, including the different types and how they interact, is presented. Finally, the definition of multiagent systems is presented.

### 5.3 Agents

As indicated in the previous section, AI planning, which uses individual entities to solve a common goal, each with different levels of specialization and responsibilities, was the first step towards the development of agents. To date, general consensus on the defini-
tion of an agent does not exist; however, most of the authors have agreed that agents should have certain characteristics.

The most basic definition of an agent is one who acts. However, this definition does not give a clear context of where an agent acts, and what the motive of its actions is. Russell and Norving (1995) provide a more extended definition of an agent: “anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors”. This definition, although more complete, leads us to believe that any computational program can be treated as an agent, since it is only required to have an input and output. A definition that seems to cover more aspects of what an agent is (and will be considered as definition for this study) was proposed by Jennings et al. (1998): “a computer system, situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives.” The interested reader can find additional definitions of agents in Franklin and Graesser (1996).

The definition by Jennigs et al. (1998) emphasizes the capability of flexible autonomous action as a necessary characteristic for a computational program to be considered an agent. Many other authors concurred with this assumption, although this subject is still controversial (Wooldridge 1996, Petrie 1996 & Castelfranchi 1996).

For completeness, a list of the additional agent characteristics given by several authors is presented:

- Autonomy: the ability of a system to make independent decisions, without the guidance of other agents or people. (Castelfranchi, 1995)

- Reactivity: the agent receives information of the environment through sensors. This information is used to pursue actions that change the agent’s environment in a timely fashion.
• Social ability: the ability of agents to interact with other agents in order to achieve their design objectives (Genesereth and Ketchpel, 1994)

• Pro-activity: the ability of exhibit goal-directed behavior by taking the initiative, so an agent can satisfy its design objectives.

• Mobility: the ability to move around an electronic network (White 1994).

• Veracity: the assumption that an agent will not knowingly communicate false information (Galliers, 1988).

• Benevolence: the assumption that agents do not have conflicts goals, and that every agent will therefore always try to do what is asked of it. (Rosenschein and Genesereth, 1985)

• Rationality: assumption that an agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals from being achieved (Galliers, 1988).

To help us better understand the concept of agents, Figure 5.1 presents an abstract view of how an agent functions in relation to its environment. Wooldridge (1999), pointed out that although the agent affects its environment, most agents will be embedded in complex systems; therefore they will not have complete control over their environments. However, an agent can be assumed to have partial control in that it can influence its environment.

Figure 5.1: Agent in its environment (Wooldrige, 1999).
Wooldridge and Jennings (1995), discuss three key issues that yield important insights into the properties of agents: agent theories, agent architectures, and agent languages.

- **Agent theories** deal with specifications such as, how agents are conceptualized and how they formally represent and reason about their properties.

- **Agent architectures** deal with the transition from specification to implementation and address the question of how to implement computer systems that satisfy properties laid out by the theory.

- **Agent languages** deal with programming languages used to implement the agent and answer questions such as how to program agents and how to effectively compile and execute agent programs.

Presenting the different agent theories proposed by diverse authors is beyond the scope of this study. Readers interested in this topic are referred to Wooldridge and Jennings (1995). For the purposes of this study, a brief compendium of agent architectures are presented that represents the implementation of agents in computer systems.

### 5.3.1 Agent architectures

Wooldridge and Jennings (1995) divided agents architectures into three types: *classical, alternative, and hybrid*. The *classical* approach architecture was based on symbolic AI and is also known as *deliberative* architecture. This architecture represents the world as symbolic models in which decisions are made via logical reasoning, based on pattern matching and symbolic manipulation. The problem with this architecture, as described before for symbolic AI, is the amount of time that the calculative rationality needs to achieve the goal of the agents.

The *alternative* architecture, also known as *reactive* architecture, does not include any kind of central symbolic world model and does not use complex symbolic reasoning.
Although reactive agents may have better time response, they only possess local information; therefore, non-local information is not taken into account.

*Hybrid* architectures combines classical and alternative approaches. In general these architectures are comprehended as a number of software layers. The arrange of the layers can be vertical or horizontal. Layered architectures are capable of reactive (alternative approach) and pro-active (deliverative approach) behavior. In general only two layers are required to represent both of the capabilities described. The principle of horizontal layering is that each independent layer behaves as an agent, suggesting which action to perform based on the input, analogous as in a parallel system. For vertical layering only one agent can perceive, and other can perform the action, analogous as a serial system (see Fig. 5.2).

As seen in Figure 5.2a, in horizontal layering, each agent competes with others to produce the output action, which could produce non-coherent behavior; therefore, an additional agent that coordinates or moderates the output of the others agents is commonly

![Figure 5.2: Layered agent architectures (Wooldridge, 1999).](image-url)
used. This moderator has to consider all the possible interactions that the $n$-layer agents produce, which can be considered as a possible problematic point of design.

Vertical layering addresses this problem, because the decisions of each agent is passed through the system, and therefore there is no need of an agent coordinator. Specifically, for the two pass control (see Figure 5.2 (b)), first the information is transmitted up to the $n$-layer; then the control is transmitted back down to the first layer. The interactions between agents are also reduced compared with horizontal layering. However, one of the concerns about this layering is that the information will need to pass each one of the agents before a decision can be made; therefore, failure of an agent can possibly have serious consequences for total performance.

There exist other approaches which combine the horizontal and vertical architectures called hybrid architectures. Here, each layer produces suggestions of what action to perform. For those cases, there should be mediation between these layers. The Touring machine (Ferguson, 1992) is an approach that uses this scheme (see Figure 5.3). The Touring machines are comprised of three concurrently-operating layers: a modeling layer, a planning layer, and a reactive layer. These three layers cover a range of different behaviors, such as time constraint situations (reactive), rational and goal-directed behavior (planning), and the reasoning of events taking place around its environment (modeling). Mediation is achieved through a control subsystem that determines which layer should have overall control of the agent.
5.3.2 Agent languages

Agent languages are defined as the methods that will allow programming of hardware or software computer systems in terms of agent theory. To date, there exist several agent-oriented programming approaches; however, not all of them consider all of the characteristics that an agent, as defined by Wooldridge and Jennings (1995), should have.

It is worth mentioning a brief historic perspective of the evolution of the programming paradigms that lead us to Agent Oriented Programming (AOP). Lind (2000) reports that the first view of a program was as monolithic block without any inherent structure. Later, programs came to be viewed as the union of several smaller structural units named subroutines. Subroutines by themselves were not powerful enough in that they emphasized the control flow aspect of programming, and that they did not acknowledge the data involved. A new view of programming that groups data and computation was developed. The basic unit of this new paradigm was the “object”. Finally, a third change in the pro-

Figure 5.3: Touring machine framework (Ferguson, 1992).
programming paradigm, which uses not merely passive objects but rather active entities are the “agents”. Figure 5.4 presents the historic development described above.

The transition from objects to agents was not direct. Wooldridge (1997), indicates that when objects were considered more like processes they became concurrent programming models, and later named active-objects or “actors”.

Agha (1986) uses a framework employing a concurrent object language based on actors. Actors were proposed as universal units for concurrent computation, and they are defined as self-contained, interactive, autonomous, components of a computing system that communicate by asynchronous message passing. It is claimed that actors have certain advantages as elements to be used in complex systems. Actors allow the designer to manage concurrency explicitly. Their use in parallel distributed architectures facilitates performance gains allowing larger problems to be solved (Agha and Jamali, 1999).

Shoham (1993) presented a new language called agent-oriented programming (AOP) that can be viewed as a specialization of object-oriented programming. This considers the agent as an entity whose states consists of mental components such as beliefs, capabilities,

Figure 5.4: Historic development of agents (Lind, 2000).
choices, and commitments. AGENT-0 was the first prototype program that implemented the AOP. It was not intended for building large-scale production systems; however, it gives insight into the capabilities of this framework.

Thomas (1993) improved AGENT-0 and created the Planning Communicating Agents (PLACA) language. Its objective was to address the inability of agents in AGENT-0 to plan and communicate requests for action via high-level goals.

Although PLACA was an improvement, it still did not provide with clear definition of the relationship between the logic involved and the interpreter programming language. Fisher (1994) developed the Concurrent MetateM language, which tries to solve this drawback. This language illustrates how agents based on pure logic can work.

5.4 Multiagent systems

In the previous section, a brief background for agents was presented; however, real world problems require groups of agents to solve complex problems with a diversity of goals. Such systems are called Multi-Agent Systems (MAS). MAS can be defined as a computational system consisting of a collection of agents interacting concurrently within a given context.

Jennings et al. (1998) points out these characteristics of MAS:

- MAS have incomplete information, and therefore limited viewpoints.
- MAS do not have global system control.
- Data is decentralized.
- Computation is asynchronous.

MAS technologies are playing a critical role in developing effective and efficient problem-solving strategies and methods in large-scale smart-sensor networks. MAS tech-
nologies provide a framework for building and analyzing such systems and offer specific mechanisms for distributed decision making and coordination of systems (Weiss 2000). The agent-based view offers a powerful repertoire of tools, techniques, and metaphors that have the potential to considerably improve the way in which people conceptualize and implement many types of software. By structuring such applications as a MAS, the system will have the following advantages: speed-up due to concurrent processing; fewer communication bandwidth requirements because processing is located nearer the source of information; more reliability because of the lack of a single point of failure; improved responsiveness because processing, sensing, and effecting are co-located; and finally, easier system development due to modularity coming from decomposition into semi-autonomous agents.

MAS are being used in an increasingly wide variety of applications, including structural control (Hogg and Huberman, 1998), air traffic control (Steeb et al., 1988), patient care (Huang et al., 1995), job shop scheduling (Morley and Schelberg, 1993), and transportation management (Fisher et al. 1999).

5.5 Agent framework for smart sensors

The final part of this chapter will be to describe the development an agent framework for SHM algorithms. An agent-based architecture provides an important paradigm on which to lay the foundation for smart sensing for SHM. Indeed, Liu and Tomizuka (2003) indicated that agent-based sensing is part of the strategic research required to advance sensors and smart structures technology. Focus should be placed on selecting the best architecture, hierarchical interaction, communication, and negotiation methods for the development of SHM algorithms. Use can be made of the various agent communication
languages that have been designed (Mayfield et al. 1996; Smith and Cohen, 1996). However, all of them are tailored to specific applications, or to be used under a particular program. In this research, an agent-oriented design will be implemented in Simulink, using the StateFlow toolbox (The MathWorks, Inc., 2004).

An effective agent-based computational framework should allow for robust SHM algorithm development on a network of smart sensors that can operate within the intrinsic constraints imposed by this environment.

Many Agent-Oriented (AO) methodologies are available. Iglesias et al. (1999) present a survey of agent-oriented methodologies, in which many researchers, rather than starting from zero, have extended existing methodologies and include relevant aspects of the agents. The authors divided these extensions mainly into two areas: Object Oriented (OO) methodologies, and Knowledge Engineering (KE) methodologies.

The advantages of using extensions of the OO methodologies are that the OO and AO methodologies have important similarities: agents can be considered as active objects with mental states (defined as beliefs, desires, and intentions). Also, OO is popular, and its methods are mature. However, agents are not simply objects. While objects communicate via message-passing, agents can analyze these messages and decide whether to execute the requested action. In addition, agents are characterized in a social dimension and with a mental state. Burmeister (1996), Kinny et al. (1997), Moulin and Cloutier (1994), and Moulin and Brassard (1996) are some the authors who use the OO paradigm to create an agent framework.

The KE is the technique applied by Knowledge engineers to build intelligent systems. The advantage of KE methodologies is that they have to interact with the development of
knowledge base systems, providing a good basis for MAS. The down side is that they do
not address the social aspects of the agents or their reflective and goal oriented attitudes.
Glasser (1996) and Iglesias et al. (1997) present some of these KE extended methodolo-
gies.

Wooldrige, et al. (2000) cited that the methodologies can be also divided into top-
down approaches, based on progressive decomposition of the behavior based on some
notion of role, and bottom-up approaches by identifying elementary agent behaviors.

Finally, Petrie (2000) suggested that the common characteristics of agent-based sys-
tems is that some agent theories are interpreted for some applications to produce an agent
model used for application development. Figure 5.5 shows this concept.

5.5.1 Gaia methodology

The framework proposed herein for SHM is based on the Gaia methodology (Woold-
rige, et al. 2000). Gaia is a conceptual framework intended to allow a developer to go sys-
tematically from the system requirements to a design that is sufficiently detailed to be
implemented directly in software. This methodology was chosen because its approach is
both general (i.e. is applicable to a wide range of MAS), and comprehensive (i.e. deals

![](image.png)

Figure 5.5: Agent-based modeling (Petrie, 2000).
with both societal and agent levels of the system); and it has been specifically tailored to the analysis and design of agent-based systems. Furthermore, Gaia encourages the developer to think of building agent-based systems as a process of organizational design. The developer has the freedom to choose the most suitable platform, because Gaia does not make any assumptions about the delivery platform. A detailed summary of the concepts and procedures involved in the application of the Gaia methodology is presented.

Gaia is suitable to the development of the systems with the following domain characteristics: agents that can make use of significant computational resources, that the organization of the structure of the system is static, and whose abilities do not change over time. Additionally, it is assumed that the goal is to obtain a system that maximizes some global quality measure and consists of fewer than 100 agents types.

The Gaia methodology takes the designer from a statement of requirements to a design with enough detail to be implemented easily in a computational framework. The analysis and the design stages are the processes for developing an increasingly detailed model. The relationships between Gaia’s models and stages are shown in Figure 5.6 (Wooldrige et al. 2000).

**Analysis stage**

Understanding the system and its structure is the objective of the analysis stage. In other words, the analysis stage captures the system’s organization. Gaia’s view of the organization is based on a collection of roles. These roles stand in certain relationship to one another, and they take part in systematic, institutionalized patterns of interactions with other roles.
The concepts in the analysis stages include:

- System (related to agent-based system): artificial society or organization.
- Role: the next level in hierarchy in a system with four attributes.
  - Responsibilities: determine the functionality (divided in two types).
    - Liveness properties: describe those states of affairs that an agent must bring about, given certain environmental conditions.
    - Safety properties: describe an acceptable state of affairs that should be maintained across all states of executions.
  - Permissions: the rights associated with the role, which identify the resources that are available to that role in order to realize its responsibilities.
  - Activities: computations associated with the role that may be carried out by the agent without interacting with other agents. (Activities are private actions).
  - Protocols: define the way in which the role in question can interact with other roles.

Figure 5.7 shows the relationship between the analysis concepts described above.

The Gaia’s analysis stage is comprised of two further models: the roles model and the interaction model (see Fig. 5.6).
The roles model identifies the key roles in the system. In simple terms, a role is more or less identical to the notion of an office. These roles are characterized by two types of attributes:

- Permissions/rights associated with the role, and
- Responsibilities of the role

As defined before, the permissions associated with a role have two aspects.

Identify the resources that can legitimately be used to carry out the role.

State the resource limits which the role executor must operate.

In the same way, the responsibilities of the role define their functionality. The responsibilities can be divided into two categories:

- Liveness responsibilities: something will be done. This has two basic components:
• activities (unit of action that the agent, may perform and does not involve interaction with other agents).

• protocols, (activities that do require interaction with other agents)

• Safety responsibilities: invariants of the system or safe conditions

Wooldrige et al. (2000) suggested a visual aid to differentiate liveness activities from liveness protocols. Activities will be written in a underline format (i.e., $\underline{yyyy}$ is referred to an activity).

Finally, liveness expressions outline the possible execution trajectories through the various activities and protocols associated with the role. The general form of a liveness expression is:

$$\text{ROLENAME} = \text{expression}$$

where ROLENAME is the name of the role whose liveness properties are being defined, and expression is the liveness expression defining the liveness properties of ROLENAME. The liveness expressions use the operators shown in Table 5.1.

### TABLE 5.1

**OPERATORS FOR LIVENESS EXPRESSIONS (WOOLDRIDGE ET AL. 2000)**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \cdot y$</td>
<td>$x$ followed by $y$</td>
</tr>
<tr>
<td>$x</td>
<td>y$</td>
</tr>
<tr>
<td>$x^*$</td>
<td>$x$ occurs 0 or more times</td>
</tr>
<tr>
<td>$x^+$</td>
<td>$x$ occurs 1 or more times</td>
</tr>
<tr>
<td>$x^m$</td>
<td>$x$ occurs infinitely often</td>
</tr>
<tr>
<td>$[x]$</td>
<td>$x$ is optional</td>
</tr>
<tr>
<td>$x</td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.8 presents a template of the role schemata that puts together all the information discussed above in reference to the role model. This template will be used in this study.

The *Interaction model* represents the links between roles, and consists of a set of protocol definitions, one of each type of inter-role interaction. A protocol definition consists of the following attributes:

- *purpose*: a description of the interaction;
- *initiator*: the role(s) responsible for starting the interaction;
- *responder*: the role(s) with which the initiator interacts;
- *inputs*: information used by the role initiator while enacting the protocol;
- *outputs*: information supplied by/to the protocol responder during the course of the interaction
- *processing*: a description of any processing the protocol initiator performs during the course of the interaction.

Figure 5.9 shows the graphical representation of the protocol described above.

In summary, the Gaia’s *analysis stage* can be laid out as follows:

<table>
<thead>
<tr>
<th>Role Schema:</th>
<th>name of the role</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Short description of the role</td>
</tr>
<tr>
<td><strong>Protocol and Activities</strong></td>
<td>protocols and activities in which the role plays a part</td>
</tr>
<tr>
<td><strong>Permissions</strong></td>
<td>“rights” associated with the role</td>
</tr>
<tr>
<td><strong>Responsibilities</strong></td>
<td></td>
</tr>
<tr>
<td>Liveness</td>
<td>liveness responsibilities</td>
</tr>
<tr>
<td>Safety</td>
<td>safety responsibilities</td>
</tr>
</tbody>
</table>

Figure 5.8: Template of role schemata (Wooldrige et al. 2000).
1. Identify the roles in the system.

2. For each role, identify and document the associated protocols.

3. Using the protocol model as a basis, elaborate the roles model.

1. Iterate stages (1)- (3).

To better understand the analysis stage, consider the example of a CONCRETEMAKER’s role (the goal of this role is to ensure that a construction site is kept supplied with concrete). The permissions associated with the CONCRETEMAKER role will be:

reads supplied concreteMachine //name of concrete machine

supplied concreteType //type of concrete

concreteStatus //full, refill

changes concreteStock //stock level of concrete

The definition of these four permissions of the CONCRETEFILLER indicates that, the agent in charge of this role has parameterized the specific machine to be used, and the type of concrete, has the permission to access the value concreteStatus, and has the permission to both read and modify the value concreteStock.

The responsibilities associated to the CONCRETEMAKER role are:

- whenever the concrete is at refill level, make more.
- whenever new concrete is ready, make sure to inform workers about it.
The liveness expression representative of the responsibilities defined before for the CONCRETEMAKER role is:

\[ \text{CONCRETEMAKER} = (\text{Make}, \text{InformWorkers}, \text{CheckStock}, \text{AwaitRefill})^{\omega} \]

This expression translates into the CONCRETEMAKER will execute the protocol Make, followed by the protocol InformWorkers, followed by the activity CheckStock, and the protocol AwaitRefill. Finally, it indicates that this sequential execution is repeated infinitely often.

A list of predicates are typically used to represent the safety requirements. Usually, these predicates involve variables listed in the role’s permissions attributes. Regarding the CONCRETEMAKER role, an agent carrying out this role will be required to ensure that the concrete stock is never empty, and is represented by the following safety expression:

- \( \text{concreteStock} > 0 \)

The schema for role CONCRETEFILLER is presented in Figure 5.10.

Finally, the illustration of the Make protocol is shown (see Figure 5.11). This states that the protocol is initiated by the role CONCRETEMAKER and involves the role CONCRETEMACHINE. Make protocol consist of mixing the concrete components for a specified type of concrete, in the machine named concreteMachine, and results in CONCRETEMACHINE being informed about the value of concreteStock.

**Design stage**

The design stage of the Gaia methodology aims to transform the abstract models derived during the analysis stage into models of sufficiently low level of abstraction that they can be easily implemented. This process involves generating three models (see Fig. 5.6).
**Role Schema:**  
\[ \textit{CONCRETE MAKER} \]

**Description:**
This role ensures that a construction site is kept supplied with concrete, and informing the worker when new concrete is made.

**Protocol and Activities**
Make. InformWorkers. CheckStock. AwaitEmpty

**Permissions:**
- reads supplied \( \text{concreteMachine} \) //name of concrete machine
- supplied \( \text{concreteType} \) //type of concrete
- changes \( \text{concreteStatus} \) //full or refill
- changes \( \text{concreteStock} \) //stock level of concrete mixing components

**Responsibilities**

**Liveness:**
\[ \text{CONCRETE FILLER} = (\text{Make. InformWorkers. CheckStock. AwaitEmpty})^o \]

**Safety:**
- \( \text{concreteStock} > 0 \)

Figure 5.10: Schema for role \textit{CONCRETE MAKER}

---

**Make**

<table>
<thead>
<tr>
<th>Make</th>
<th>concreteMachine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>concreteType</td>
</tr>
<tr>
<td>ConcreteMaker</td>
<td>ConcreteMachine</td>
</tr>
</tbody>
</table>

Mixing concrete components in concrete machine

| ConcreteMachine | concreteStock |

Figure 5.11: Definition of Make protocol
• *The agent model* identifies the agents types and agent instances.

• *The service model* identifies the main services required to realize the agent’s role.

• *The acquaintance model* documents the lines of communication between the different agents.

In detail, the *agent model* documents the various agents types used in the system, and the agent instances realized by these agents types. The agent types can be seen as a set of agent roles. The *agent model* can be defined using a simple agent type tree. In this tree, leaf nodes correspond to roles, and other nodes correspond to agent types. In this context, an agent can be composed of two or more different roles. As seen in Figure 5.12, the agent type $t_1$ has children $t_2$ and $t_3$; therefore $t_1$ is composed of the roles of $t_2$ and $t_3$. Finally, the agent instances are documented by annotating agent types in the agent model. Table 5.2 presents the instance qualifiers used for annotating agent types.

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>there will be exactly $n$ instances</td>
</tr>
<tr>
<td>$m..n$</td>
<td>there will be between $m$ and $n$ instances</td>
</tr>
<tr>
<td>*</td>
<td>there will be 0 or more instances</td>
</tr>
<tr>
<td>+</td>
<td>there will be 1 or more instances</td>
</tr>
</tbody>
</table>

The *services model* identifies the services associated with each agent role, and specifies the main properties of these services (or functions of the agent). If a service is performed by an agent, it will be necessary to document its properties.
The acquaintance model simply defines the communication links between agent types. However, it does not define what messages are sent or when messages are sent; it simply indicates that communication pathways exist.

Summarizing, the Gaia’s design stage can be laid out as follows:

1. Create an agent model.
2. Develop a services model, by examining activities, protocols, and safety and liveness properties of roles.
3. Develop an acquaintance model from the interaction model and agent model.

To better understand the design stage, consider again the CONCRETEMAKER role. The first model, the agent model, will consider one-to-one correspondence between the agent CONCRETEMAKER and the CONCRETEMAKER role (see Figure 5.13). With respect to the service model, there are four protocols and activities associated with this role: Make, InformWorkers, CheckStock, and AwaitEmpty. In general, there will be at least one ser-

---

Figure 5.12: Agent type tree.

Figure 5.13: The CONCRETEMAKER agent model.
vice associated with each protocol. In the case of the activity, the service will take as input the stock level and some threshold value and will simply compare the two (see Table 5.3). Finally, the acquaintance model indicated that the agent ConcreteMaker is in communication with the agent ConcreteMachine and with the agent Workers, as shown in Figure 5.14.

5.5.2 SHM agent-framework

An agent framework for SHM based on the Gaia methodology has been developed. Specifically, the SHM algorithm is considered to be decentralized, i.e., each sensor is capable of detection of damage at its location.

The steps proposed in the Gaia methodology are: requirement statement, analysis stage, and design stage. In the following sections, the development of each stage is presented.

**TABLE 5.3**

**SERVICE MODEL OF CONCRETEMAKER**

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>make concrete</td>
<td>concreteMachine, concreteType</td>
<td>concreteStock</td>
<td>true</td>
<td>concreteType ∈{200 kg/cm², 300kg/cm²}</td>
</tr>
<tr>
<td>inform workers</td>
<td>concreteStatus</td>
<td></td>
<td>true</td>
<td>Workers knows status</td>
</tr>
<tr>
<td>check stock</td>
<td>stock level, threshold</td>
<td>compare stock level vs. threshold</td>
<td>concreteStock &gt; 0</td>
<td>concreteStock &gt; 0</td>
</tr>
<tr>
<td>Await Refill</td>
<td>concreteMachine</td>
<td>concreteStatus</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Requirement statement

Based on the information acquired through sensors, the SHM application will seek to find the presence of damage at a certain location in the structure. The parts involved in this process include user, network formation and synchronization, sensing, SHM algorithm, and data aggregation.

Description of the network

To gain a better understanding of the application, a brief description of the network is required. In general, a typical network is a series of points or nodes interconnected by communication paths. Networks can interconnect with other networks and contain subnetworks. A node is a connection point, either a redistribution point or an end point for data transmissions. Overall, a node has programmed or engineered capability to recognize and process or forward transmissions to other nodes. For this study, the network is formed by two types of nodes, gateway and sensor nodes. Gateway nodes (or base stations) are the only ones that output information to the user from the network. Sensor nodes allocate the sensing device, whose information is used for damage detection. Sensor nodes process, share and transmit information that finally can be obtained at the gateway node.

In principle, each sensor node is laid out at each floor of the structure, and the gateway node is preferably located at the base of the structure. The gateway node also will
function as the director for topology formation and synchronization of the network. Figure 5.15 shows a typical network array for a simple structure.

The gateway node initiates the topology formation by sending a radio signal. Sensor nodes within the gateway node’s range will reply, forming a radio link. Later these sensor nodes will cascade this effect. The only restriction imposed is that a sensor node cannot have more than 1 parent (precedent node). The motives behind this restriction is to avoid duplicated messages. Figure 5.16 shows the topology formation for the structure in Figure 5.15. This idea of network formation was originally developed by Mechitov et al. (2004).

To better understand the network formation in cases more complex than the network shown in Figure 5.15, an additional example is presented. Suppose an arrangement of base station and sensor nodes as shown in Figure 5.17 (a). The base station (ID number 0) ini-

![Figure 5.15: Typical network array.](image)

75
Figure 5.16: Topology formation.
iates the topology formation. Only sensor nodes 1 and 2 are reached, and they will acknowledge the base station as their parent, forming a radio link (see Figure 5.17 (b)). Additionally, using this radio link, nodes 1 and 2 will indicate in their future messages that they are 1 hop away from the gateway sensor. In Figure 5.17 (c), nodes 1 and 2 emit a radio signal. Nodes 3 and 6 acknowledge nodes 1 and 2 as parents respectively, and will indicate that they are 2 hops away from the base station. Meanwhile, nodes 4 and 5 are in the radio range of nodes 1 and 2, therefore with the possibility of connection for both of them. Because a rule has been imposed that each node can have only one parent, they need to decide which one it will be. A diverse number of criteria can be considered for the parent election. In this study, the following rules will be applied in the importance order pro-

Figure 5.17: Topology rules for tree formation.
posed: (i) select the parent closer to the root, (ii) select the parent with the strongest radio signal, (iii) select the parent with fewer children. For this example, the first rule cannot define the parent, since nodes 1 and 2 have the same distance from the root (1 hop away). The second rule applies because node 4 is closer to node 1, and node 5 is closer to node 2, therefore having respectively stronger radio signal. Figure 5.17 (d) shows the final topology obtained.

The topology information is updated periodically, and when a sensor node cannot communicate with his parent, it will send a message to find another possible parent. If no parent can be found, the sensor will increase its radio power range and try to reach a new parent again. This increased radio power information is sent embedded in the message, so the new possible parent will also increase its power radio to reach its new child. After finding a new parent, a message will be sent to the base station to inform of the faulty sensor node. Figure 5.18 shows the course of action described.

Description of the network’s synchronization

At the same time that the network topology is formed, the nodes are synchronized. The Flooding Time Synchronization Protocol (FTSP), proposed by Kusy and Maroti (2004), will be used as the synchronization algorithm in this study. This protocol utilizes a single broadcasted message to obtain time synchronization reference points between the sender and its neighbors. It assumes that the sender of the time sync message is already synchronized; then it synchronizes the local clock of the receiver to the global time provided by the sender. Nodes broadcast time synchronization messages periodically and synchronize their clocks to that of an elected leader. Kusy and Maroti (2004) have shown that the algorithm is robust and that it handles topology changes and node failures well.
Figure 5.18: Topology update.

Node failure

Child increase radio power transmission

New parent replies with increased radio power transmission

New radio link is formed

Child sends a message to base station about faulty node

Base station receives information

Node failure
The final steps towards the SHM framework is the implementation of the algorithm. This chapter will only focus in the overall procedure, i.e. without making reference to a particular algorithm. In the subsequent chapter, a detailed description of the algorithm chosen as the reference implementation will be presented.

**SHM framework**

The purpose of the SHM algorithm is to determine the most likely localization of the damage based on the information obtained by the sensor nodes. Whenever a node identifies possible damage at its position, it will inquire as to the damage index of its neighbors. With this information, it will be feasible to assess possible location of the damage.

To illustrate the sequence of events described above, Figure 5.19 shows a damage scenario at the 3rd floor of the structure shown in Fig. 5.15. The first event is the detection of damage at the sensor node on the fourth floor. This node will query his neighboring nodes about their status; in this specific case neighboring nodes are nodes at the 3rd and 5th floors. After the node fourth receives the information, it will inform the base station of this situation.

**Analysis stage**

After describing the process of the SHM application, the methodology moves forward to an organizational view. Figure 5.20 shows the flow-diagram of the process-oriented system’s operation for the base station smart sensor. Figure 5.21 shows the flow-diagram for a node smart sensor.

After the physical installation of the sensors on the structure, the overall process is initiated by the user, by commanding the base station to form and synchronize the sensor
Figure 5.19: Sequence of events after SHM encounters a damaged position.

1. Damage in level 3
2. Node 4 queries neighbors with respect to their damage index
3. Node 4 receives “No Damage” indicator from bottom node and “Damage” from the top node
4. Node 4 informs the gateway node of this situation.
5. Child sends a message to base station about faulty node

Possible damage in floor 3.
Figure 5.20: Flow-diagram of the base station smart sensor.
Figure 5.21: Flow-diagram of a node smart sensor.
network. At periodic intervals, the network will update itself. Also, at different intervals, the SHM application will be applied. If damage is encountered, a notification will be sent to the user. Finally, in case of node failure, the network will attempt to automatically make repairs, later informing the user of the situation.

From the flow diagrams and process description, the following roles can be identified:

• **User:**
  • Responsible for initiating the process

• **User manager:**
  • Responsible for managing user’s request
  • Responsible for overseeing the process and ensuring that appropriate information is returned

• **Base station:**
  • Generally responsible (manager) for the base station smart sensor.
  • Responsible for radio communication
  • Responsible for network formation/synchronization
  • Responsible for the internal clock
  • Responsible for schedule the update of the network

• **Node:**
  • Generally responsible (manager) for the Node smart sensor
  • Responsible for radio communication
  • Responsible for managing the network formation/synchronization requests from other sensors
  • Responsible for the sensing device
• Responsible for the SHM algorithm
• Responsible for neighbor inquiring
• Responsible for scheduling the SHM algorithm

Figures A.1 through A.15 in the Appendix present the role schema for the identified responsibilities. Figures A.16 through A.30 present the protocol definitions associated with the roles.

Design stage

After the completion of the analysis stage, the methodology moves forward to the design stage, with the definition of the three models: the agent, service, and acquaintance models.

As stated before, the purpose of the Gaia agent model is to document the various agent types. In this study each smart sensor (base station and node) will be considered to be an agent, composed of a number of the roles as described previously. Along with the base station and node agents, an agent that represents the user is required, and an agent mediator between the user and the base station is proposed. Figure 5.22 shows the proposed agent model.

In the case of the service model, the identification of the services associated with each agent role are described in Table A.1 through A.15 of the Appendix.

Finally, Figure 5.23 shows the acquaintance model. This model is rather simple and indicates that interaction between the user and the network is done through the base station; although this schema shows two Node Agents, the agent model in Figure 5.22 shows that there could be one or more instances.
5.6 Summary

In this chapter, we have presented the development of an agent-based framework for SHM based on the Gaia methodology. The development of the SHM algorithm takes into consideration that each sensor will be capable of damage detection at their location; therefore, it is ideal for decentralized algorithms. However, the framework can be modified to fit other SHM algorithms, such as hierarchic or centralized algorithms.
Because the Gaia conceptual framework is sufficiently detailed, it can be implemented directly. However, the selection of a decentralized SHM algorithm to be implemented is required. In the next chapter, this research and the selection of the SHM is presented.
CHAPTER 6

REFERENCE IMPLEMENTATION OF THE AGENT-BASED SHM SYSTEM

To illustrate the agent-framework described in the previous chapter, a reference implementation of a SHM system using the Auto-Regression – Auto-Regression with eXogenous input (AR–ARX) algorithm method proposed by Sohn et al. (2001). Following a brief overview of the many available approaches for damage detection, a detailed explanation of pattern recognition techniques and, in particular, the AR–ARX is presented. The implementation of the agent-based framework using the AR–ARX technique as the SHM algorithm is then described and an example for damage detection in a shear building is presented.

6.1 Current SHM algorithms.

A great number of algorithms for damage detection have been developed, nearly all of which focus on the estimation of the change in their structural dynamic characteristics. A review by Doebling et al. (1996) cites more than 600 references to these types of methods. In this paper, several different approaches are identified, including methods based on frequency changes, mode shape changes, mode shape curvatures/strain mode shape changes, dynamically measured flexibility, mass/stiffness/damping matrix updates, detection of structural nonlinearity, neural network-based methods, etc. Most of these algorithms require a numerical model representing the structure; however, there are a few
Doebling et al. (1996) suggested that frequency change methods for damage detection can be subdivided into forward and inverse problems. The first consists of calculating frequency shifts from a known type of damage and comparing the measured frequencies to the computed frequencies to determine the damage. The second approach calculates the damage parameters from the frequency shifts.

Mode shapes and curvatures-change techniques use both experimental and analytical methodologies. The advantage of these techniques, as compared to using pure frequency changes, is that mode shapes and curvatures are more sensitive to damage. However, calculation of mode shapes requires measurement of the responses distributed throughout the structure; therefore a centralized processing system must be employed.

Algorithms based on flexibility matrices have been shown to be insensitive to changes in the more difficult to measure high-frequency modes of the structure. These techniques rely heavily on having an accurate model of the structure to determine damage locations. However, studies using the Damage Locating Vectors (DLV) method proposed by Bernal (2002) and subsequently extended by Gao et al. (2002) show significant promise for decentralized implementation.

Matrix update methods aim to calculate changes in the mass, stiffness, and damping matrices, to reproduce as closely as possible the measured static or dynamic response from the data. Non-linear methods mainly seek to identify the opening and closing of cracks. These methods typically rely on information obtained from Finite Element Models (FEM), as well as the collected records of the sensors in the structure.
Neural network-based methods represent an alternative computational paradigm in which the solution to a problem is learned from a set of examples. Many of these methods use a Multi-Layer-Perceptron (MLP) trained through backpropagation. Finally, other methods are presented by Doebling et al. (1996) that use diverse schemes, such as decrements techniques, damping loss factors, and propagation of share waves, among others.

There exist other types of classification for SHM techniques, such as time-history and spectral pattern methods, time domain methods, frequency domain methods and time-frequency methods (Farrar et al., 1999). The application of SHM algorithms is broad, ranging from bridges, building, and trusses to rotating machinery equipment and aircraft.

6.2 Pattern recognition

The great majority of the SHM methods presented above rely on an analytical model of the structure, inheriting the uncertainties in the modeling and tuning of the parameters. Pattern recognition is used in a wide number of fields, including wave form analysis, biology, and psychology. The four leading approaches for pattern recognition include template matching, neural networks, syntactic or structural matching, and statistical classification (Jain et al. 2000).

Template matching is one of the first approaches to pattern recognition. Based on matching, that is the determination of similarities between entities of the same type, template matching employs an available template or prototype of the pattern to be recognized. Patterns to be recognized are matched against the template. A diversity of all allowable positions and scale changes are taken into account. Template matching, although effective for some applications, has the disadvantage of not recognizing patterns that have been distorted during imaging process or viewpoint change.
Neural networks are computing systems which mimic the brain through a network of highly interconnected, processing elements. Neural networks have learning capabilities which enable them to recognize complex patterns. The main characteristic of the neural networks is their ability to learn complex nonlinear input-output relationships and adapt themselves to the data.

The syntactic approach is a technique that divides the pattern into subpatterns. The most basic element of a pattern is called a primitive. Primitives observe a set of rules that interconnect them. The syntactic approach can be thought as a language. The primitives then are part of the alphabet, and the rules that interconnect them can be viewed as the grammar. Complex patterns can be described by a small number of primitives and grammatical rules. Syntactic approaches have been used where patterns have a definite structure which can be captured as a set of rules. Many difficulties are encountered in the search of primitives and the inference of the grammar.

Statistical pattern recognition views each pattern as represented by a set of $n$ features or measurements. The goal is to choose those features that allow pattern vectors belonging to different categories to be identified. The effectiveness of this method is determined by how well patterns from different classes can be separated. Statistical pattern recognition techniques are not model-based and have the potential to be deployed in a decentralized manner; therefore they are attractive for SHM based on smart sensors and will be discussed in detail in the next section.

6.2.1 \textbf{Statistical pattern recognition technique.}

One of the early studies on this topic was by Fukunaga (1972), who proposed a two step process for statistical pattern recognition: feature selection and classifier design.
Given a set of measurements, these can be classified by means of a discriminant function or classifier (see Figure 6.1). Furthermore, if the number of measurements to be classified is considerable, their management can become cumbersome. Consequently, the extraction of the most important features of the information becomes an important task. This process is called feature selection. Figure 6.2 presents the block diagram of the statistical pattern recognition proposed by Fukunaga.

Jain et al. (2000) proposed a model for statistical pattern recognition. In this model there exist two operational modes: training (learning) and classification (testing) (see Figure 6.3). The objective of the preprocessing module (in both operational modes) is to segment the pattern of interest from the background, i.e., removing of noise and...
normalization of the pattern. The feature extraction/selection module (in the training mode) has the objective of finding the appropriate features for representing the input patterns. Additionally the classifier is trained to partition the feature space. This training process allows for optimization of the preprocessing and feature extraction/selection strategies. In the classification mode (testing), the trained classifier assigns the input pattern to one of the pattern classes under consideration based on the previously measured features.

Farrar et al. (1999) presents a comprehensive four step approach for using statistical pattern recognition in SHM:

1. Operation evaluation
2. Data acquisition and cleansing
3. Feature extraction and data reduction
4. Statistical model development

Operational evaluation

This stage begins to determine the limitations on what can be monitored and how it will be achieved. To help determine how to implement the SHM systems, three questions should be answered:
1. How is damage defined for the system being investigated and, for multiple damage possibilities, which are the greatest concern?
2. What are the conditions, both operational and environmental, under which the system to be monitored functions?
3. What are the limitations on acquiring data in the operational environment?

Data acquisition and cleansing

The second stage involves the selection of sensors, their number and placement throughout the structure, how often the data should be collected, and during what operational conditions. The normalization of the data is an important issue, because it is collected during different operational and environmental conditions; therefore, the focus should be placed on collecting the data during similar operational environments.

After the data has been collected, data to be used in the feature selection process is chosen, which is the objective of the cleansing process. Farrar et al. (1999) claims that this task is done using the experience of the individuals directly involved with data acquisition.

Feature selection

As also mentioned by Fukunaga (1972) the feature selection determines the elements of the set of measurements that do not follow a specific pattern and which in this case represents a damaged structure. Therefore it is necessary to recognize which are the best attributes to extract from the data, i.e., that will be sensitive to the presence of damage. Large amounts of data may accumulate and their management can become difficult; consequently, data condensation should be considered.

Statistical model development

The objective of the last stage of this paradigm is to quantify the damage state of the structure. The algorithms proposed to assess this quantification fall into three categories: group classification, analysis of outliers, and regression analysis. If data from both the
damage and undamaged conditions is available it is referred as *supervised learning*. Two of the categories can be considered supervised learning: group classification and regression analysis. If no information on the damaged condition of the structure is available, then it is referred as *unsupervised analysis*.

Group classification puts the features into respective categories: damage or undamaged. In an informal manner, experienced users can deduce the presence, type, and level of damage. In a formal sense, there exist methodologies designed to perform this classification. Such methodologies include Bayesian classifiers (Fig. 6.4), $K$th-nearest neighbor rules (Fig. 6.5), and neural network classifiers (Fig. 6.6).

In regression analysis, the goal is to determine the values of parameters for a function that cause the function to best fit a set of observations. In this case the function is designed to determine the locations and extent of damage (see Figure 6.7). To utilize this function, features from the undamaged structure and from the structure at varying damage levels are required.

![Diagrams](image)

$P(A| P_1, P_3)$ is maximal. $P(A| P_1)$ is maximal.

Figure 6.4: Example of group classification using Bayesian classifiers
The analysis of outliers identifies, from observations in a distribution of data, elements that deviate so much from the other observations as to arouse suspicion that it was generated by a different mechanism and therefore suspected to represent damage conditions (see Figure 6.8). The most utilized tool for the estimation of outliers is multivariate probability density function.

Figure 6.5: Example of group classification using $k^{th}$ nearest neighbors

Figure 6.6: Example of group classification using neural networks
Figure 6.9, proposed by Farrar et al. (1999) and Sohn and Farrar (2001), summarizes the structural health monitoring paradigm described. Because the focus here is to illustrate implementation of the agent paradigm using the AR–ARX, this dissertation will center on the last step of the SHM method proposed, i.e., statistical model development.

### 6.3 The AR–ARX method

Sohn et al. (2001) present a damage diagnosis method that uses time series analysis of vibration signals. This method is based on a two-stage approach that employs the auto-regressive (AR) analysis and the Auto-Regressive with eXogenous (AR–ARX).
Figure 6.9: Flow chart for implementing a SHM program (Sohn and Farrar, 2001).
The first step of this methodology is the acquiring of acceleration records from unknown operational and environmental conditions, but with the certainty of the health of the structure. These records will be used as a reference database for comparison with future unknown structural conditions.

One of the first challenges to overcome in order to determine the damage to a structure is to take into account the variability of the operating conditions, as well as the environmental effects to which a structure is subject. For that purpose a normalization scheme is needed. Since the recorded time series are considered stationary, the proposed normalization is expressed in equation (6.1)

\[
\hat{x} = \frac{x - \mu_x}{\sigma_x}
\]  

(6.1)

where \(\mu_x\) and \(\sigma_x\) are, respectively, the mean and standard deviation of \(x\).

After the signals are normalized, an AR model with \(p\) auto-regressive terms is constructed for each of the recorded signals; this information will be considered the reference database. In general, an AR model uses historical data to predict future data. It is also known as an infinite impulse response filter (IIR) or an all-pole filter. Equation (6.2) shows an AR\((p)\) model.

\[
\hat{x}(t) = \sum_{j=1}^{p} \phi_{xj} \hat{x}(t-j) + e_x(t)
\]  

(6.2)

where \(\hat{x}(t)\) is a normalized element at discrete time index \(t\), an \(e_x(t)\) is an unobservable random error. As in a linear filter, \(\sum_{j=1}^{p} \phi_{xj} \hat{x}(t-j)\) represents the weighted sum of the previous observations.
The Yule-Walker method is used to calculate the AR coefficients $\phi_{ij}$. There exist other methods, such as the Burg, covariance, and modified covariance; however, the Yule-Walker method performs as well as the other methods, but because it employs the autocorrelation function, it does not produce singular conditions regardless of the order, and it always produces a stable model. Additionally, the structure of the autocorrelation matrix has a Toeplitz configuration (i.e., constants entries along their diagonals), making it suitable to use well known algorithm solutions.

When a new time series $y(t)$ is recorded in which the structural condition is unknown, estimation of the $\phi_{ij}$ coefficients of the AR analysis is done. This operation is indicated in equation (6.3):

$$
\hat{y}(t) = \sum_{j=1}^{p} \phi_{ij} \hat{y}(t-j) + e_{y}(t)
$$

(6.3)

where $\hat{y}(t)$, $e_{y}(t)$, and $\sum_{j=1}^{p} \phi_{ij} \hat{y}(t-j)$ have the same meaning as for equation (6.2).

To determine if the new time series represents a damaged condition for the structure, it is compared with the signals recorded in the data base. The closest signal will be defined as the one which minimize the difference of the AR coefficients. Equation (6.4) shows this difference.

$$
\text{Difference} = \sum_{j=1}^{p} (\phi_{ij} - \phi_{ij})^2
$$

(6.4)

Since the data base records were obtained under different operational and/or environmental conditions, if the new signal was obtained under similar conditions and there has not been damage in the structure, the closest signal of the data-base and the new signal will have similar dynamic features. However, if indeed the new recorded signal is a prod-
uct of a damage condition of the structure, there will be no time series capable of representing it, and therefore a considerable difference will be found.

After the minimal difference is encountered, the second part of the two-stage time series analysis is calculated. Here, it is assumed that the unobservable error \( e_x(t) \) represents the unknown external inputs; therefore it is proposed the use of an ARX model to reconstruct the input/output relationship between \( e_x(t) \) and \( \hat{x}(t) \), as shown in equation (6.5)

\[
\hat{x}(t) = \sum_{i=1}^{a} \alpha_i \hat{x}(t-i) + \sum_{j=1}^{b} \beta_j e_x(t-j) + e_{\hat{x}}(t)
\]  

(6.5)

where \( \alpha_i \) and \( \beta_j \) are the coefficients of the ARX model, and \( e_{\hat{x}}(t) \) is the residual error after fitting the ARX\((a, b)\) model to the \( \hat{x}(t) \) and \( e_x(t) \) pair. It has been suggested by Ljung (1987) that the values for \( a \) and \( b \) be kept with the relation expressed in equation (6.6)

\[
a + b \leq p
\]  

(6.6)

thus indicating the sum of the two should be less than the \( p \), the number of coefficients of the AR model.

From the previous calculation, the coefficients \( \alpha_i \) and \( \beta_j \) are used to explore how well this ARX\((a, b)\) model reproduces the input/output relationship of \( \hat{y}(t) \) and \( e_{\hat{y}}(t) \), as shown in equation (6.7)

\[
e_{\hat{y}}(t) = \hat{y}(t) - \sum_{i=1}^{a} \alpha_i y(t-i) - \sum_{j=1}^{b} \beta_j e_{\hat{y}}(t-j)
\]  

(6.7)

where \( e_{\hat{y}}(t) \), \( \alpha_i \), \( \beta_j \), \( \hat{y}(t) \), and \( e_{\hat{y}}(t) \) are defined as before. If the employed pair \( \hat{x}(t) \) and \( e_{\hat{x}}(t) \) used to reproduce the new recorded signal is not representative of \( \hat{y}(t) \), the standard
deviation of the residual error $\varepsilon_y^*(t)$ would have a big change with respect to the standard deviation of the $\varepsilon_x^*(t)$.

The last step in the AR–ARX technique is to calculate the ratio of the standard deviations (see equation (6.8)); this will be considered as the damage index of the structure. If this damage index becomes greater than a proposed threshold $h$, then a structural change is assumed to have taken a place in the system.

$$\frac{\sigma(\varepsilon_y^*(t))}{\sigma(\varepsilon_x^*(t))} > h$$  \hspace{1cm} (6.8)

### 6.3.1 AR–ARX applications

Sohn et al. (2001) use the AR–ARX technique to differentiate strain data records obtained in different structural conditions of a patrol boat. Only three records were available, two of them under the same structural condition and the other obtained when the surface of the patrol boat was damaged. Because of the limited number of records, it was necessary to divide the reference record into overlapping windows. The value of threshold $h$ proposed was 1.85, which had a 95% success rate of discrimination between damage and no-damage conditions.

Sohn and Farrar (2001) localized the damage in an eight degrees-of-freedom mass-spring system when introducing nonlinearity between masses. The systems were excited by a force electro-dynamic shaker at different amplitude levels from ranging from 3 to 7 V root mean square (RMS) value. The reference data based consisted of 45 acceleration records. The residual errors presented a maximum at the sensor record localized near the actual damage locations.
Lynch (2002) proposed the use of the AR–ARX model deploy on arrays of smart sensors. In this approach, the reference database is assumed to have been previously acquired and saved on a centralized data server. Subsequently, the wireless sensors acquire new records when the health of the structure is unknown. These records are then normalized and the coefficients of the AR models are calculated on the sensor’s microprocessors. This information is then sent to the centralized server, which finds the closest AR model in the database and sends it back to the sensor units. Finally, the sensor units calculate the damage index. Figure 6.10 shows this process. While Lynch obtained good results, his approach requires that each sensor communicate directly with the base station and is therefore not scalable.

The AR-ARX method is a promising algorithm for damage detection. The next section discusses its implementation in the agent-based framework.

6.4 Implementation of the AR-ARX technique in the agent-based framework

The agent-based framework is implemented in Simulink using the StateFlow toolbox (The MathWorks, Inc., 2004). Simulink is a software package that enables modeling, simulation, and analysis of dynamical systems. In addition, StateFlow is a graphical design and development tool for control and supervisory logic, that is based on finite state machine theory. A finite state machine is a representation of an event-driven (reactive) system. Event-driven systems simply make a transition from one state (mode) to another prescribed state, provided that the condition defining the change is true (see Figure 6.11).

As presented in Chapter 5, the agent-based framework proposed will follow a prescribed flow, changing from one state to another, depending on the prescribed conditions. Thus, the StateFlow toolbox is a suitable tool for this purpose. Nevertheless, the transition
Acquire measurement data and locally store in the wireless sensing unit

Normalize measurement data

\[ \hat{y} = \frac{y - \mu_y}{\sigma_y} \]

Determine coefficients of an AR(p) process

Transmit AR(p) coefficients to centralized server

Receive the coefficients of an ARX model from the data base with the associated standard deviation of model’s residual error

Process measurements data with the ARX model to determine residual error of measurement data

Normalize measurement data

\[ \hat{e}_k = y_k - \sum_{i=1}^{a} a_i y_{k-i} - \sum_{j=1}^{b} \beta_j r_{k-j} \]

Determine if damage is present in the structure

\[ \frac{\sigma(\hat{e}_k^y)}{\sigma(\hat{e}_k^x)} \geq h \]

Figure 6.10: AR–ARX SHM system implementation (Lynch, 2002)

Acquire the AR process model from the wireless sensing unit

Find the closest AR model in stored database by minimizing the difference of model coefficients

\[ \min \left( \sum_{i=1}^{p} |b_i^x - b_i^y| \right) \]

Send the corresponding ARX model and standard deviation of the ARX model residuals

Figure 6.11: StateFlow driven-event.
from the proposed framework to its implementation in Simulink and StateFlow has to be adapted because of technical limitations of these platforms.

The most significant limitation presented in the implementation of the agent-based framework is the simulation of radio link and radio messages passing throughout the network. Herein, network connections were represented as a series of direct links between the gateway agent, and agent node. Under this scheme, no collisions or data losses are simulated. However, a limitation regarding the number of hops away that a node can communicate through the radio can be imposed. In this example, this limitation is set up to be two, i.e., two floors away in a structure (see Figure 6.12 for the case of a 10 story building).

![Diagram of network connections with nodes labeled 1 to 10, showing various nodes with broken radio and disconnected nodes.](image)

**Figure 6.12:** Radio range imposed limitations.
Additionally, no synchronization algorithm is used, because the system cannot be simulated to run under different clocks. Therefore, all the clocks of the system are synchronized. Consequently, all of the clock and timer roles are already available in Simulink and do not need to be implemented. The StateFlow built-in functions for this purpose are: every(X, sec) and after(X, sec).

The first computational agent is the **USERHANDLER**. This agent is represented by a switch that indicates the StartSystem has been requested by the user. Also, two groups of LEDs are used to represent the information acquired from the **BASESTATIONMANAGER** agent. These LEDs will turn on/off according with the status of the entity that they represent, therefore fulfilling the objective of informing the user about the status of the systems (see Figure 6.13).

The second computational agent is the **BASESTATION**. This agent has control over the scheduling and network formation and informs the user about the status of the network/structure. As proposed before, the **BASESTATION** can communicate up to two hops away,
i.e., it can have direct links between agent nodes 1 and 2. Also shown in Figure 6.14, the communication boxes are used to allow only one of the two inputs to be used as the connection link. (see Figure 6.14)

To implement the role BASESTATIONNETSYNC (part of the BASESTATIONMANAGER agent) which updates the network at a predetermine schedule, a StateFlow state is used (see Figure 6.15). The BASESTATIONNETSYNC state sends a signal to the network every NetSync seconds though the network, via the function BSync. A reply form the nodes of the networks is sent back to the base station informing of any broken radio links. Figure 6.15 also shows the Off and WAIT_100SEC. The BASESTATIONNETSYNC state will become active only when the user requests it; until that happens the Off state is active. The WAIT_100SEC state is a delay introduced to the program prevents it from starting until 100 seconds after the user’s request to start the system. The reason for this delay is to allow the initial transient response of the structure to attenuate.

![Figure 6.14: Radio communication links between agents.](image-url)
Figure 6.15: Base Station Agent.
The last computational agent implemented is the **NODEMANAGER**. This agent has to fulfill the roles of **NODENETSYNC**, **NODESTRUCTURALHEALTHMONITORING** and **NODESENSING**. The **NODEINQUIRER** role is achieved through radio connections outside the agent (because of limitations of Simulink discussed previously). The main structure of the **NODEMANAGER** is composed by two simultaneous states, **NODENETSYNC** and **OPERATIONAL STATES** (see Figure 6.16).

Figure 6.17 shows the two sub-states of the **NODENETSYNC** state, the On and Off states. The Off state will be active if the radio of the node is not working (the On/Off setup of the radio can be change manually by the user in real time). The On state will become

![Figure 6.16: Node agent](image-url)
active only if the radio of the node is working. When the On state senses a parent connected to the node, it can then proceed to connect with a child node.

The Operational_states implements the NodeStructuralHealthMonitoring and NodeSensing roles (see Figure 6.18). The Sensing sub-state groups the database record sensing, and the damage record sensing (see Figure 6.19). The Sen_Database state has the objective to record a predetermined number of events with a specific length defined previously by the user (see Figure 6.20). After the database records have been acquired, the state will proceed to the Sen_Damage state with similar characteristics to the Sen_Database state (see Figure 6.21). Finally, the NodeStructuralHealthMonitoring state contains three sub-states, the AR_ARX_Database, the AR_ARX_Damage, and the

Figure 6.17: NodeNetSync state
Figure 6.18: Node agent: **OPERATIONAL_STATES**.

Figure 6.19: Node agent: **NODESENSING** sub-states.
Figure 6.20: Node agent: SEN_DATABASE state.

Figure 6.21: Node agent: SEN_DAMAGE state
The complete Simulink model is presented in Figure 6.26. An addition to the elements described previously, this model includes a band-limited white noise block that simulates the input excitation. Other external elements are used to build radio connections between agents. Finally, to allow the user to experiment with diverse predetermined parameters, the number of database and damage records, length of the measurements, and time for update network are user specified.
Figure 6.23: Node agent: AR_ARX_DATABASE state.

Figure 6.24: Node agent: AR_ARX_DAMAGE state.
6.5 Numerical validation

In this section, a numerical validation of the AR–ARX technique with and without the use of the agent-framework proposed in the previous chapter is presented. A 10-story shear building will be used as example for this study (see Figure 6.27). The 10-story shear structure is model as a lumped mass system, in which masses are concentrated at each floor level, with the columns represented by equivalent shear springs. The mass and stiffness are uniformly distributed through the structure with values of 50000 kg and 1 kN/mm, respectively. A damping ratio of 2% was assumed. The first three periods of the structure are 2.97, 0.99, 0.61 seconds, respectively.

6.5.1 AR–ARX technique without agent-framework

To have a reference base-line, the AR–ARX analysis will be implemented first without the use of agents. The analytical model is excited with a band-limited white noise at its
Figure 6.26: Complete Simulink model
base. Accelerations are measured at each of the floors. A set of 50 different records were employed to construct the database reference which simulates the different operational and/or environmental conditions to which the systems is subjected. The number of reference records is rather arbitrary; however, with reference of previous studies (Sohn et al. 2001 and Sohn and Farrar 2001), this number seems adequate. Nevertheless, it will always be desirable to have as many records as the system is capable of storing.

To ensure that the response of the structure is stationary, the records are analyzed after $2/\zeta$ seconds, where $\zeta$ is the damping factor (in this example $2/0.02 = 100$ seconds). Simulations show that after this lapse of time, the transients of the systems decay below 1%.

Figure 6.27: Ten degree-of-freedom shear building.
(see Figure 6.28). The length of the analyzed records is 20 seconds, sampled at 0.0112 seconds. To ensure adequate representation of all of the modes, the integration time is calculated as the smallest period (0.225 sec.) of the structure divided by 20, so all the modes can be adequately represented. However, a slower sampling rate is required to eliminate high frequency unobservable random errors $e_x(t)$; therefore, a decimation of 4 was employed, resulting in a sampling rate of 0.0899 seconds. The number of coefficients used for the AR model is 50 ($p = 50$).

Damage is simulated in the structure by modifying the stiffness of a selected floor, but leaving the mass unchanged. A set of 20 records is collected after an element is damaged. For each one of these records, the AR coefficients are calculated and the database is searched for the closest match based on Eq. (6.4).

![Figure 6.28: Response of a single degree-of-freedom system subject to a unit impulse.](image_url)
For the second stage of the analysis, the coefficients employed for the ARX model in Eq. (6.5) are $a = 28$ and $b = 22$. Other combinations of coefficients were used; similar results were obtained. Equation (6.8) is evaluated for each of the 20 records, and damage indices are calculated. Finally, the average of the 20 damage indices is calculated.

Ideally, the damage index should be 1 if the structure is undamaged; however, this is usually not the case. To determine the normalization constant, half of the database records (in this case 25) are used as a damage records.

After the normalization constant has been determined for each of the measured records and normalized. If the damage index is greater than 1.0 possible damage is indicated.

Damage in the column element between the 4th and 5th floors is simulated by reducing the associated stiffness by 20%. The final damage indices are graphically presented in Figure 6.29. This figure shows that the damage indices of the 4th and 5th floor are greater

![Figure 6.29: Damage indices plot when damage 20% is imposed in the column between 4th and 5th floors.](image)
that 1.0 threshold value \((h=1.0)\), indicating damage in the column element between them. The 9th damage index is also greater than one; however, neither node 8 or 10 indicates damage, therefore no damage can be inferred in this position.

An additional example is presented in Figure 6.30 where the stiffness element between 7th and 8th floors is reduced by 20%. Damage indices of floors 1st, 5th, 7th and 8th are above the threshold level. Because only floor 7th and 8th are consecutive, this indicates damage in the column between them.

### 6.5.2 AR–ARX technique using agent-framework

This section presents results based on the agent-framework proposed in the previous chapter, and Simulink implementation described in section 6.4. For comparison purposes, the same cases studied in section 6.5.1 will be considered.

As described in Chapter 5, the network will update itself every \(N\) seconds. A trade-off exist in the selection of the update time. If this time is too large, information can be lost in

![Figure 6.30: Damage indices plot when 20% stiffness damage is imposed in the column between 7th and 8th floors.](image)

Figure 6.30: Damage indices plot when 20% stiffness damage is imposed in the column between 7th and 8th floors.
the case of a broken transmission link, because re-configuration is not done sooner. However, if this number is too small, a significant power will be consumed by sending and receiving messages. For the purpose of this simulation, 10 seconds was selected as the update time. Because the intent of this example was not to consider power consumption, this number does not affect the simulator.

The user has the prerogative to start or stop the simulation at any time. Additionally, for simulation purposes, the user has the ability to decide which node’s radio has failed using a series of switches.

For the data collection, the database is populated by recording 50 events of 20 seconds in duration, sampled every 0.0112 seconds. Later, each sensor records 20 damage condition events of 20 seconds. After each record is collected, the system pauses for 21 seconds to allow for the required processing.

Result of the two damaged cases, in which columns between the 4th and 5th floors and the 7th and 8th floors have their associated stiffness reduced by 20%, are presented in Figures 6.31 and 6.32. Figure 6.31 shows that the damage indices of the 4th and 5th floor are greater than 1.0, indicating damage in the element between them. Figure 6.32 shows that the damage indices of the 7th and 8th floors are greater than the threshold level, identifying a possible damage location between these two floors.

6.5.3 Comparison of results

Results obtained from the centralized approach and the agent-based framework are comparable, following the same trend and detecting the floor with the simulated damage. The differences between the approaches are due to the different excitation input utilized.
Figure 6.31: Damage indices plot when damage 20% is imposed in the column between 4th and 5th floors.

Figure 6.32: Damage indices plot when damage 20% is imposed in the column between 7th and 8th floors.
6.6 Summary

This chapter presented a reference implementation of the agent-based framework for smart sensing developed in Chapter 5. The AR–ARX approach was selected as the SHM algorithm to be employed. Because the AR–ARX technique is based on the analysis of time series, a centralized data acquisition system is not required; therefore each agent can process the information locally informing the base station in case of damage detection. A detailed description of the implementation of the agent-based framework in Simulink with support of the StateFlow toolbox was presented. Some of the limitations encountered in this software platform and the associated solutions were discussed.

To demonstrate the applicability of an agent-based framework, a 10-floor shear building is considered with and without damage. Simulation results indicates that the agent-based framework produces the outcome as the centralized architecture.

The agent-based framework proposed was demonstrated to be robust, i.e., capable of reconfiguring the network of sensors, and exploits the capabilities of the smart sensor by processing acquired data on the sensor’s microprocessor.
CHAPTER 7
CONCLUSIONS AND FUTURE STUDIES

7.1 Conclusions

In this dissertation, the use of the smart sensor technology for SHM/C of civil infrastructure has been explored. Focus was placed on developing appropriate sensors and computational frameworks.

An extensive literature review of smart sensors developed to date was provided. The evolution of smart sensors from simple wireless nodes that can only send information directly through a centralized data acquisition system to nodes that can locally process data to provide an efficient, robust and flexible sensing system was illustrated. Implementation challenges, such as data loss, network formation, and synchronization, were identified. Because of its open software/hardware environment, smart sensors based on the Mote platform developed at the University of California at Berkeley were selected as the focus of this research.

A description of the software and hardware characteristics of the Berkeley-Mote platform was then presented. Features and characteristics of this platform, including both the hardware and the software, were detailed. Smart sensors based on this platform provide the impetus for the development of the next generation of SHM/C systems.

A new sensor board (named “Tadeo”) that implements the SD-1221 high sensitivity accelerometer was developed. The currently available MTS310CA sensor board, which
includes the bi-axial accelerometer ADXL202E, was assessed through a series of experiments. The ADXL202E was shown to be deficient for civil engineering applications, having high noise density and poor sensitivity in the low frequency range. Tests of the Tadeo sensor board show excellent performance. Finally, aliasing of measured acceleration signals was eliminated through the use of a modular four-pole Butterworth filter.

An agent-based framework for SHM/C was developed based on the Gaia methodology. Following an overview of the agent paradigm, including brief historical background and a description of its characteristics, classifications, and architectures, a detailed description of was provided for a new agent-based smart sensing system using the Gaia methodology.

A reference implementation of this agent-based framework for SHM was provided using use of the AR–ARX algorithm. This implementation was accomplished using Simulink with the StateFlow toolbox. A numerical example employing a 10-floor shear building in which damage was simulated by reducing the level of stiffness in a column element was used to illustrate the approach. Results indicate that the agent-based framework is as effective as the centralized architecture. The proposed agent-based framework is shown to be an effective paradigm for implementation of SHM/C algorithms for civil infrastructure applications.

### 7.2 Future studies

While this research demonstrates the potential of the use of smart sensing technology for SHM/C in civil infrastructure systems, many questions still need to be addressed. This section presents several directions for future studies.
7.2.1 Implementation in smart sensor array

The feasibility of the agent-based framework was established in this research; the next step is its experimental implementation of this framework on a smart sensor array. As discussed previously, the Simulink simulation model used in this research for the agent-based framework has some limitations in representing the smart sensor’s communication environment. To fully explore the potential of the agent framework under real conditions, (e.g., packet collisions, data loss, measurement noise, memory constrains, network synchronization, etc.) it must be tested on an array of smart sensors.

7.2.2 Intel mote

Intel® has recently announced development of the Intel-Mote platform (Kling 2003), with a number of improvements over the Berkeley Mote platform (see Fig. 7.1 and table 7.1). This Mote will fully support TinyOS. The ultimate goal (to be accomplished by 2005 —see Fig. 7.2) is to develop the Mote in the form of a single microchip with layered components which will include sensors, RF MEMS, nonvolatile storage, digital/analog silicon, and battery. The present prototype is half the size of the original Berkeley-Mote and provides increased CPU power for tasks such as location detection and digital signal process-

Figure 7.1: Intel® mote prototype (Kling, 2003).
Other enhancements include reliability of radio links using the BT protocol, increased CPU, security features, modular design, reduced cost, competitive battery life, assumed duty cycle <1%, and additional on-board memory.

The Intel-Mote platform differs from the Berkeley-Mote in that it utilizes the BT radio protocol. As discussed in Chapter 1, there exist different types of communication protocols. Basically, although radio frequency commutation using free band frequency
(used by the Berkeley-Mote) allows communication over great distances, there are a number of problems with power consumption, insecure communications, and the need for complicated synchronization schemes. For that reason the Intel-Mote platform now uses the BT protocol, which is claimed to be mature. Silicon on Ceramic (SOC)-integrated devices are available, consisting of single chips with BT/controller/memory at low costs. BT uses spread-spectrum operation, which increases link reliability and ensures precise synchronization within piconets.

As the Intel-Mote platform matures, its use in SHM/C applications should be further investigated.

### 7.2.3 Extension of agent-based paradigm to complex systems.

The 10-story shear building example proposed in this study, although basic, demonstrate the applicability of an agent-based framework for SHM. However, real structures are more complicated. As an example of the types of systems envisioned for real structures, a scheme for determining the localization of joint damage in a structure is outlined in the following paragraphs.

To support a more complicated layout of sensors, the use of a backbone network configuration is suggested. A backbone network is an important architectural element for building reliable networks. It provides a path for the exchange of information between different subnetworks. A backbone can tie together diverse networks in the same building, in different buildings, or over wide areas. Generally, the backbone's capacity is greater than that of the networks connected to it.

Figure 7.3 shows a typical backbone network. Small dots represents sensor nodes and large dots backbone nodes. In this study, the backbone nodes will have a stronger power
transmission radio; sensor nodes will have the ability to switch between low and high power transmission. The objective of this switching mode is to minimize the power consumption and maximize the efficiency of data transmission in the network. In general only one sensor node (the master node at each joint) will switch to high power. Figure 7.4 shows the radio power levels in a joint under this scheme.

The base station will initiate the main sequence of events. The first event will consist of the topology formation. The best way to form the network is to broadcast the relationship table of connections; therefore the base station and nodes are required to know the location and identification number of the sensors through the structure. Table 7.3 shows the connection relationship for node B in Figure 7.3. Additionally, a backup connection
TABLE 7.2

TOPOLOGY FOR NODE B

<table>
<thead>
<tr>
<th>Parent</th>
<th>Children</th>
<th>Backbone Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>4, 5, 6, 16, 17, 18, 19</td>
<td>C</td>
</tr>
</tbody>
</table>
can be established in case of node failure. Table 7.2 shows the backup table for node B if node A fails.

<table>
<thead>
<tr>
<th>Parent</th>
<th>Children</th>
<th>Backbone Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4, 5, 6, 16, 17, 18, 19</td>
<td>C</td>
</tr>
</tbody>
</table>

This process is robust, because the developer can design an equilibrium network connection, and also predict and solve possible network failures scenarios. However, if there are a large number of sensors, this process could be time consuming. An alternative solution is the use of ad-hoc networks.

In the ad-hoc network scheme, the base station initiates the process by sending a signal to backbone nodes. The backbone nodes located in the radio range of the base station will emit an acknowledge signal and establish a radio link. Later, this link will be cascaded to expand the tree (see Figure 7.5).

Once the backbone network is formed, a signal aimed at sensor nodes is sent. Sensor nodes hearing this will start sending a signal, using low-power mode transmission, to locate nodes in their neighborhood. After all the nodes in a joint have linked to one another, a master node is elected. This selection is arbitrary. For example, the master node can be selected by the highest identification number of the joint. The master node will send a signal, using a high power mode, to the backbone network (as shown in Figure 7.4). The leader will have the task of informing the backbone parent about any anomalies.
Figure 7.5: Network formation of backbone nodes.
encountered by the nodes of the joint. This process can be considered a type of data aggregation.

### 7.2.4 Other damage detection algorithms.

As shown in Chapter 6, the AR–ARX technique can determine the localization of the damage; however, it has limitations when trying to localize damage at very low (less or equal to 10%) or very high (greater or equal than 40%) damage. To illustrate this problem, a 10% reduction of stiffness in the column element between 4th and 5th floors, and reduction of 50% in the column between 7th and 8th floors is simulated. Figures 7.6 and 7.7 show the respective damage index plots. Damaged elements cannot be inferred here.

Recent work by Gao et al, (2004) presents a novel idea to identify damage using a distributed computing damage algorithm. This approach uses an extended version of the

![Figure 7.6: Damage indices plot when 10% stiffness damage is imposed in column between 4th and 5th floors.](image)

**Figure 7.6: Damage indices plot when 10% stiffness damage is imposed in column between 4th and 5th floors.**
Damage Locating Vector (Bernal, 2002) method, which considers the input excitation signal as unknown, i.e., ambient vibration. This technique seeks to use a group of sensors to detect damage, organizing them in a hierarchical manner. The information obtained from each hierarchal group is sent back to the base station through a selected leader of the group. A numerical example indicates that the algorithm can detect both single and multiple damaged case scenarios. This algorithm is very promising for implementation on a smart sensor network using the agent-based framework developed herein.

Figure 7.7: Damage indices plot when 50% stiffness damage is imposed in column between 7th and 8th floors.
APPENDIX

ANALYSIS AND DESIGN STAGE OF THE GAIA METHODOLOGY

For the analysis stage and design stages of the Gaia methodology, the role schema, protocols, and associated services are presented in this appendix.

A.1 Schema roles.

In Figures A.1 through A.15 are presented the role schema.

A.2 Protocols.

In Figures A.16 through A.30 are presented the protocols associated with the roles.

A.3 Services.

In Tables A.1 through A.15 are presented the service models of the agent roles proposed.
<table>
<thead>
<tr>
<th>Role Schema:</th>
<th>USER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>Individual in charge of initiating the process.</td>
</tr>
<tr>
<td><strong>Protocol and Activities:</strong></td>
<td>StartSystemRequest</td>
</tr>
<tr>
<td><strong>Permissions:</strong></td>
<td>generates ( StartSystem ) // boolean</td>
</tr>
<tr>
<td><strong>Responsibilities:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Liveness:</strong></td>
<td>( USER = (\text{StartSystemRequest})^+ )</td>
</tr>
<tr>
<td><strong>Safety</strong></td>
<td>• true</td>
</tr>
</tbody>
</table>

Figure A.1: Schema for role USER
### Role Schema: ** USERHANDLER **

**Description:**
Receives request from the user and oversees the process to ensure appropriate information is returned

**Protocol and Activities:**
- AwaitCall
- ProduceStatus
- InformUser

**Permissions:**
- **reads**
- **supplied**
- **StartSystem** // boolean
- **Status** // Informs about the health status of the system

**Responsibilities:**

**Liveness:**
- \( \text{USERHANDLER} = (\text{AwaitCall}.\text{GenerateInformation})^+ \)
- \( \text{GENERATEINFORMATION} = (\text{ProduceStatus}.\text{InformUser}) \)

**Safety:**
- **true**

Figure A.2: Schema for role USERHANDLER.
**Role Schema:** BASESTATIONMANAGER (BSM)

<table>
<thead>
<tr>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsible for initiating the systems and informing the user about the status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol and Activities:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AwaitRequest, DoNetSync, IdentifyType, RequestScheduleStart, InformUser</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Permissions:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>reads</strong></td>
</tr>
<tr>
<td><code>StartRequest</code> // boolean</td>
</tr>
<tr>
<td><code>ConnectionTable</code> // Table of connection of the network</td>
</tr>
<tr>
<td><code>InformOfDataReception</code> // Radio indicates reception of information</td>
</tr>
<tr>
<td><strong>generates</strong></td>
</tr>
<tr>
<td><code>Status</code> // Status of the system</td>
</tr>
<tr>
<td><code>NetSyncRequest</code> // Request network/synchronization</td>
</tr>
<tr>
<td><code>StartScheduleRequest</code> // Request start of schedule</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibilities:</th>
</tr>
</thead>
</table>

**Liveness:**

\[ \text{BASESTATIONMANAGER} = (\text{AwaitRequest.ActionType})^0 \]

\[ \text{ACTIONTYPE} = (\text{NetSyncEvent} \mid \text{RadioDataReception}). \text{InformUser} \]

\[ \text{NETSYNCEVENT} = (\text{DoNetSync.RequestScheduleStart}) \]

\[ \text{RADIODATARECEPTION} = (\text{IdentifyType}) \]

**Safety:**

- true

---

Figure A.3: Schema for role BASESTATIONMANAGER
<table>
<thead>
<tr>
<th>Role Schema:</th>
<th><strong>BASESTATIONRADIO</strong> (BSR)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>Responsible for the transmission and reception of the messages, and for setting the power level of the radio in the event of not receiving acknowledgement.</td>
</tr>
<tr>
<td><strong>Protocol and Activities:</strong></td>
<td>AwaitReception, TransmissionRequest, InformOfDataReception, <strong>Send</strong>, ReceiveAcknowledgment, <strong>SendAcknowledgment</strong>, TransmissionReply</td>
</tr>
<tr>
<td><strong>Permissions:</strong></td>
<td></td>
</tr>
<tr>
<td>reads</td>
<td></td>
</tr>
<tr>
<td>TransmissionInformation</td>
<td>// Data to be sent through the radio</td>
</tr>
<tr>
<td>TransmissionRequest</td>
<td>// boolean</td>
</tr>
<tr>
<td>RadioData</td>
<td>// Radio data received</td>
</tr>
<tr>
<td>generates</td>
<td></td>
</tr>
<tr>
<td>TransmissionReply</td>
<td>// Informs that reply has been received</td>
</tr>
<tr>
<td>InformOfDataReception</td>
<td>// Informs that data has been received</td>
</tr>
<tr>
<td>changes</td>
<td></td>
</tr>
<tr>
<td>PowerLevel</td>
<td>// Power level of the radio</td>
</tr>
<tr>
<td><strong>Responsibilities:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Liveness:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>BASESTATIONRADIO</strong> = (AwaitReception.ActionType)</td>
<td></td>
</tr>
<tr>
<td>ACTIONTYPE = (Transmission</td>
<td>ReceptionOfData)</td>
</tr>
<tr>
<td>TRANSMISSION = (TransmissionRequest, <strong>Send</strong>, ReceiveAcknowledgment, <strong>SendAcknowledgment</strong>, TransmissionReply)</td>
<td></td>
</tr>
<tr>
<td>RECEPTIONOFDATA = (SendAcknowledgment, InformOfDataReception)</td>
<td></td>
</tr>
<tr>
<td><strong>Safety</strong></td>
<td></td>
</tr>
<tr>
<td>• true</td>
<td></td>
</tr>
</tbody>
</table>

Figure A.4: Schema for role **BASESTATIONRADIO**.
Role Schema: BASESTATIONNETSYNC (BSNS)

Description:
Responsible for the network update/formation and their synchronization. Generates a network connection table.

Protocol and Activities:
NetSyncRequest, RequestTransmission, BuildTable, Sync, NetSyncResponse, GetTime, NetSyncDataAssemble

Permissions:
reads
  Time // Time from the clock of the Base Station
  TransmissionReply // Information received by the radio
  NetSyncRequest // boolean

generates
  ConnectionTable // Identifies type of data received
  NetSyncData // Data to be sent through the radio

Responsibilities:

Liveness:
BASESTATIONNETSYNC = (NetSyncRequest, NetSyncData, RequestTransmission, BuildTable, Sync, NetSyncResponse)
NETSYNCDATA = (GetTime, NetSyncDataAssemble)

Safety:
• true

Figure A.5: Schema for role BASESTATIONNETSYNC.
### Role Schema: BASESTATIONCLOCK (BSC)

**Description:**
Responsible for providing the clock upon request.

**Protocol and Activities:**
ClockRequest, Clock, ReturnClock

**Permissions:**
reads

\[
\text{ClockTime} \quad \text{// Clock from the CPU timer}
\]
generates

\[
\text{Time} \quad \text{// Time required for network synchronization}
\]

**Responsibilities:**

**Liveness:**
\[\text{BASESTATIONCLOCK}= (\text{ClockRequest, Clock, ReturnClock})\]

**Safety:**
- true

---

Figure A.6: Schema for role BASESTATIONCLOCK.
<table>
<thead>
<tr>
<th>Role Schema:</th>
<th>BASESTATIONTIMER (BST)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>Responsible for scheduling network updates.</td>
</tr>
<tr>
<td><strong>Protocol and Activities:</strong></td>
<td>ScheduleStart, GetTime, RegressiveTime, RequestNetSyncUpdate</td>
</tr>
<tr>
<td><strong>Permissions:</strong></td>
<td>reads</td>
</tr>
<tr>
<td></td>
<td>supplied ScheduleTime // Defines schedule time for network/synchronization update</td>
</tr>
<tr>
<td></td>
<td>StartRequest // boolean</td>
</tr>
<tr>
<td></td>
<td>generates NetSyncRequest // Request to do the network synchronization.</td>
</tr>
<tr>
<td><strong>Responsibilities:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Liveness:</strong></td>
<td>BASESTATIONTIMER= (ScheduleStart, GetTime, RegressiveTime, NetSyncRequest)°</td>
</tr>
<tr>
<td><strong>Safety:</strong></td>
<td>• true</td>
</tr>
</tbody>
</table>

Figure A.7: Schema for role BASESTATIONTIMER.
<table>
<thead>
<tr>
<th>Role Schema:</th>
<th>NODEMANAGER (NoM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Responsible for initiating the system and for informing its parent about its status.</td>
</tr>
<tr>
<td>Protocol and Activities:</td>
<td>AwaitRequest, GetSHM, InformParent, IdentifyType, DoNetSync, ObtainStatus.</td>
</tr>
<tr>
<td>Permissions:</td>
<td></td>
</tr>
</tbody>
</table>
| reads | RadioData // Information from radio  
| | NetworkTable // Table of connection of the network.  
| | SHMDATA // Damage indication.  
| generates | Status // Status of the Children to be sent to parent  
| | NetSyncRequest // Information from radio  
| Responsibilities: |  
| Liveness: | NODEMANAGER=(AwaitRequest.ActionType)  
| | ACTIONTYPE=(GetSHM | RadioReception).InformParent  
| | RADIORECEPTION=(IdentifyType, (DoNetSync | ObtainStatus)  
| Safety: | • true  

Figure A.8: Schema for role NODEMANAGER.
### Role Schema: NODERADIO (NoR)

#### Description:
Responsible for the transmission and reception of messages and for setting the power level of the radio in the event of not receiving acknowledgement.

#### Protocol and Activities:
AwaitReception, TransmissionRequest, InformOfDataReception, Send, ReceiveAcknowledgment, SendAcknowledgment, TransmissionReply

#### Permissions:
- **reads**
  - TransmissionInformation // Data to be sent through the radio
  - TransmissionRequest // boolean
  - RadioData // Radio data received
- **generates**
  - TransmissionReply // Informs that reply has been received
  - InformOfDataReception // Informs of data has been received
- **changes**
  - PowerLevel // Power level of the radio

#### Responsibilities:

**Liveness:**
- NODERADIO = (AwaitReception.ActionType)
- ACTIONTYPE = (Transmission|ReceptionOfData)
- TRANSMISSION = (TransmissionRequest, Send, ReceiveAcknowledgment, SendAcknowledgment, TransmissionReply)
- RECEPTIONOFDATA = (SendAcknowledgment, InformOfDataReception)

**Safety:**
- true

---

Figure A.9: Schema for role NODERADIO.
### Role Schema: NodeNetSync (NoNS)

**Description:**
Responsible for network update/formation and its synchronization. Generates a network connection table.

**Protocol and Activities:**
NetSyncRequest, RequestTransmission, BuildTable, Sync, NetSyncResponse, GetTime, NetSyncDataAssemble

**Permissions:**
- **reads**
  - Time // Time from the clock of the Base Station
  - TransmissionReply // Information received by the radio
  - NetSyncRequest // boolean
- **generates**
  - ConnectionTable // Identifies type of data received
  - NetSyncData // Data to be sent through the radio

**Responsibilities:**

**Liveness:**
\[
\text{NodeNetSync} = (\text{NetSyncRequest}, \text{NetSyncData}, \text{RequestTransmission}, \text{BuildTable}, \text{Sync}, \text{NetSyncResponse})
\]

\[
\text{NetSyncData} = (\text{GetTime}, \text{NetSyncDataAssemble})
\]

**Safety:**
- true

---

Figure A.10: Schema for role NodeNetSync.
<table>
<thead>
<tr>
<th>Role Schema:</th>
<th>NODECLOCK (NoC)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>Responsible for providing the clock upon request.</td>
</tr>
<tr>
<td><strong>Protocol and Activities:</strong></td>
<td>ClockRequest, Clock, ReturnClock</td>
</tr>
<tr>
<td><strong>Permissions:</strong></td>
<td>reads</td>
</tr>
<tr>
<td></td>
<td>ClockTime // Clock from the CPU timer</td>
</tr>
<tr>
<td></td>
<td>generates</td>
</tr>
<tr>
<td></td>
<td>Time // Time required for network synchronization</td>
</tr>
<tr>
<td><strong>Responsibilities:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Liveness:</strong></td>
<td>BASESTATIONCLOCK= (ClockRequest, Clock, ReturnClock)</td>
</tr>
<tr>
<td><strong>Safety:</strong></td>
<td>• true</td>
</tr>
</tbody>
</table>

Figure A.11: Schema for role NODECLOCK.
**Role Schema:** \texttt{NODETimer (NoT)}

**Description:**
Responsible for scheduling network update.

**Protocol and Activities:**
ScheduleStart, GetTime, \texttt{RegressiveTime}, RequestNetSyncUpdate

**Permissions:**
- \texttt{reads}
  - supplied: \texttt{ScheduleTime} // Defined schedule time for network/synchronization update
  - \texttt{StartRequest} // boolean
  - \texttt{generates}: \texttt{NetSyncRequest} // Request to do the network synchronization.

**Responsibilities:**

**Liveness:**
\texttt{NODETimer} = (ScheduleStart. GetTime. RegressiveTime. NetSyncRequest)°

**Safety:**
- true

---

Figure A.12: Schema for role NODETimer.
**Role Schema:** \texttt{NODE\_STRUCTURAL\_HEALTH\_MONITORING} (NoSHM)

<table>
<thead>
<tr>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsible for the Structural Health Monitoring algorithm</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol and Activities:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHMRequest, SensingRequest, \texttt{SHMAlgorithm}, SHMResponse, AnalyzeData, GetInquire</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Permissions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>reads</td>
</tr>
<tr>
<td>supplied \texttt{ScheduleSHMTime} // Defined schedule time for network/synchronization update</td>
</tr>
<tr>
<td>\texttt{SensingData} // Recorded sensing data</td>
</tr>
<tr>
<td>generates</td>
</tr>
<tr>
<td>\texttt{DamageIndex} // Damage index of the measurement (true or false)</td>
</tr>
<tr>
<td>\texttt{DamageIndexInquire} // Define if it is necessary to inquire Damage index of neighbors (true or false)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibilities:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liveness:</td>
</tr>
<tr>
<td>\texttt{SHMAlgorithm}\textsuperscript{\textbackslash N} = (SHMRequest, SensingRequest\textsuperscript{\textbackslash N}, \texttt{SHMAlgorithm}, StatusDamage)</td>
</tr>
<tr>
<td>\texttt{STATUSDAMAGE} = (Damage</td>
</tr>
<tr>
<td>\texttt{DAMAGE} = (GetInquire, \texttt{AnalyzeData}, SHMResponse)</td>
</tr>
<tr>
<td>\texttt{NO_DAMAGE} = (SHMResponse)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Safety:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• if requestnumber &lt; N =&gt; DamageIndex = nil</td>
</tr>
</tbody>
</table>

---

Figure A.13: Schema for role \texttt{STRUCTURAL\_HEALTH\_MONITORING}.
<table>
<thead>
<tr>
<th><strong>Role Schema:</strong> NODESENSING (NoSe)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td>Responsible for acquiring data through the sensing device</td>
</tr>
<tr>
<td><strong>Protocol and Activities:</strong></td>
</tr>
<tr>
<td>SensingRequest, Sensing, SaveSensing, SensingReply</td>
</tr>
<tr>
<td><strong>Permissions:</strong></td>
</tr>
<tr>
<td>reads supplied</td>
</tr>
<tr>
<td>SensingLength // Defines the sensing length of the application</td>
</tr>
<tr>
<td>generates</td>
</tr>
<tr>
<td>SensingRecord // Record of the sensing device</td>
</tr>
<tr>
<td><strong>Responsibilities:</strong></td>
</tr>
<tr>
<td><strong>Liveness:</strong></td>
</tr>
<tr>
<td>SENSING= (SensingRequest, Sensing, SaveSensing, SensingReply)</td>
</tr>
<tr>
<td><strong>Safety:</strong></td>
</tr>
<tr>
<td>• memory &gt; 0</td>
</tr>
</tbody>
</table>

Figure A.14: Schema for role NODESENSING
**Role Schema:** \texttt{NODEINQUIRER (NoI)}

**Description:**
Responsible for inquiring about neighbors’ Damage Index.

**Protocol and Activities:**
\texttt{InquireRequest. GetDamageIndex, CompareDamage, InquireResponse}

**Permissions:**
- **Reads:**
  - \texttt{DamageIndexInquire} // true or false
  - \texttt{CompareDamage} // If top and bottom neighbors have different damage indicators then true, otherwise false
  - \texttt{InformDamage} // Information of possible damage to the Base Station
- **Generates:**

**Responsibilities:**
\texttt{NODEINQUIRER= (InquireRequest, GetDamageIndex, CompareDamage, InquireResponse)}

**Safety:**
- true

---

Figure A.15: Schema for role \texttt{NODEINQUIRER}
<table>
<thead>
<tr>
<th>StartSystemRequest</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
</tr>
</tbody>
</table>

US commands the USH to start the system.

Figure A.16: Definition of protocols associated with User.
Figure A.17: Definition of protocols associated with UserHandler role: (a) AwaitCall, (b) ProduceStatus, and (c) InformUser.
Figure A.18: Definition of protocols associated with BaseStationManager:
(a) AwaitRequest, (b) DoNetSync, (c) RequestScheduleStart, and (d) InformUser
BSM informs the USH about the connection table of the network

---

Figure A.18: (contd)
Figure A.19: Definition of protocols associated with BaseStationRadio: (a) AwaitReception, (b) TransmissionRequest, (c) TransmissionReply, and (d) InformOfDataReception.
Figure A.20: Definition of protocols associated with BaseStationNetSync:
(a) NetSyncRequest, (b) GetTime, (c) NetSyncResponse, and (d) RequestTransmission,
<table>
<thead>
<tr>
<th>(d) TransmissionRequest</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSNS</td>
</tr>
<tr>
<td>BSNS receives the message and tries to send it. If no reply is received, increases the radio power until reaches the maximum. If no reply is received, informs of the situation in the reply message.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>TransmissionReply</td>
</tr>
<tr>
<td>BSR</td>
</tr>
<tr>
<td>BSR informs BSNS about the transmission reply.</td>
</tr>
</tbody>
</table>

Figure A.20: (contd)
Figure A.21: Definition of protocols associated with BaseStationClock: (a) ClockRequest, and (b) ReturnClock.
Figure A.22: Definition of protocols associated with BaseStationTimer: (a) ScheduleStart, (b) GetTime, and (c) RequestNetSyncUpdate.
Figure A.23: Definition of protocols associated with NodeManager: (a) AwaitRequest, (b) DoNetSync, (c) InformParent, (d) GetSHM, and (e) InformParent
(c) TransmissionRequest

<table>
<thead>
<tr>
<th>NoM</th>
<th>NoR</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoM Requests that NoR transmit information to its parent about either NetworkTable or SHMData.</td>
<td></td>
</tr>
</tbody>
</table>

StatusRequest

(d) NetSyncRequest

<table>
<thead>
<tr>
<th>NoR</th>
<th>NoM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoR receives the message and tries to send it. If no reply is received, increases the radio power until reaches the maximum. If no reply is received, informs of the situation in the reply message.</td>
<td></td>
</tr>
</tbody>
</table>

TransmissionReply

SHMRequest

<table>
<thead>
<tr>
<th>NoM</th>
<th>NoSHM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoN requests that NoSHM get the damage index.</td>
<td></td>
</tr>
</tbody>
</table>

SHMResponse

<table>
<thead>
<tr>
<th>NoSHM</th>
<th>NoM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoSHM replies about the damage index to NoM</td>
<td></td>
</tr>
</tbody>
</table>

DamageIndex

Figure A.23: (cont)
Figure A.24: Definition of protocols associated with NodeRadio: (a) AwaitReception, (b) TransmissionRequest, (c) TransmissionReply, and (d) InformOfDataReception.
Figure A.25: Definition of protocols associated with NodeNetSync: (a) NetSyncRequest, (b) GetTime, (c) NetSyncResponse, and (d) RequestTransmission,
Figure A.26: Definition of protocols associated with NodeClock: (a) ClockRequest, and (b) ReturnClock.
Figure A.27: Definition of protocols associated with NodeTimer: (a) ScheduleStart, (b) GetTime, and (c) RequestNetSyncUpdate.
Figure A.28: Definition of protocols associated with NodeSHM: (a) SHMRequest, (b) SensingRequest, (c) SHMResponse, and (d) GetInquire.
NoSHM requests that NoI query the neighbors about the damage index.

NoI inquires about damage index, and delivers this information to the NoSHM.
Figure A.29: Definition of protocols associated with NodeSensing: (a) SensingRequest, and (b) SensingReply.
Figure A.30: Definition of protocols associated with NodeInquirer: (a) InquireRequest, (b) GetDamage, and (c) InquireResponse.
### TABLE A.1: SERVICE MODEL OF THE USER AGENT

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>request of start system</td>
<td>UserRequest</td>
<td>StartSystem</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

### TABLE A.2: SERVICE MODEL OF THE USERHANDLER AGENT

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>await call of the user</td>
<td>UserRequest</td>
<td>StartSystem</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>produce status of the system</td>
<td>StartSystem</td>
<td>Status</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>inform user</td>
<td>Status</td>
<td></td>
<td>true</td>
<td>user knows status</td>
</tr>
</tbody>
</table>

### TABLE A.3: SERVICE MODEL OF THE BASESTATIONMANAGER AGENT

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>await userhandler request</td>
<td>StartRequest</td>
<td>StartSystem</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>check action type</td>
<td>StartSystem</td>
<td>actionType</td>
<td>true</td>
<td>actionType ∈ {NetSync, RadioData}</td>
</tr>
<tr>
<td>produce network/synchronization</td>
<td>NetSyncRequest</td>
<td>NetworkTable</td>
<td>actionType = NetSync</td>
<td>true</td>
</tr>
<tr>
<td>identify signal</td>
<td>RadioData</td>
<td>Status</td>
<td>actionType = RadioData</td>
<td>true</td>
</tr>
<tr>
<td>inform User</td>
<td>Status</td>
<td></td>
<td>true</td>
<td>user knows status</td>
</tr>
</tbody>
</table>
### TABLE A.4: SERVICE MODEL OF THE BASESTATION RADIO

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>await reception</td>
<td>eventOccurs</td>
<td>actionType</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>check action type</td>
<td></td>
<td>actionType</td>
<td>true</td>
<td>actionType ∈ {Transmission-Request,DataReception}</td>
</tr>
<tr>
<td>produce radio transmission</td>
<td>transmission-Information</td>
<td>informDataReception</td>
<td>radio ≠ busy</td>
<td>true</td>
</tr>
<tr>
<td>produce acknowledgment</td>
<td>RadioData</td>
<td>acknowledge-ment</td>
<td>actionType = RadioData</td>
<td>true</td>
</tr>
<tr>
<td>inform base station</td>
<td>informOfDataReception</td>
<td></td>
<td>true</td>
<td>Base Station is informed</td>
</tr>
</tbody>
</table>

### TABLE A.5: SERVICE MODEL OF THE BASESTATION NETSYNC

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>request of network synchronization</td>
<td>Request</td>
<td>StartNetSync</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>produce network/synchronization data</td>
<td>Time</td>
<td>NetSyncData</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>produce radio transmission</td>
<td>NetSyncData</td>
<td>DataReceived</td>
<td>radio ≠ busy</td>
<td>true</td>
</tr>
<tr>
<td>produce synchronization</td>
<td>DataReceived</td>
<td>ConnectionTable</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>inform base station</td>
<td>InformOfConnectionTable</td>
<td></td>
<td>true</td>
<td>Base Station is informed</td>
</tr>
</tbody>
</table>
### TABLE A.6: SERVICE MODEL OF THE BASESTATIONCLOCK

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>request of clock</td>
<td></td>
<td>Request</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>inform of time</td>
<td>Time</td>
<td></td>
<td>true</td>
<td>Requester is informed</td>
</tr>
</tbody>
</table>

### TABLE A.7: SERVICE MODEL OF THE BASESTATIONTIMER

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>start time</td>
<td>ScheduleTime</td>
<td>StartTime</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>get time</td>
<td></td>
<td>Time</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>request NetSync</td>
<td>Request-NetSync</td>
<td></td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

### TABLE A.8: SERVICE MODEL OF THE NODEMANAGER AGENT

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>await request</td>
<td>Request</td>
<td></td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>check action type</td>
<td>Request</td>
<td>actionType</td>
<td>true</td>
<td>actionType ∈ {GetSHM, RadioReception}</td>
</tr>
<tr>
<td>produce SHM</td>
<td>SHMRequest</td>
<td>SHMData</td>
<td>actionType = GetSHM</td>
<td>true</td>
</tr>
<tr>
<td>identify signal</td>
<td>RadioData</td>
<td>signalType</td>
<td>actionType = RadioReception</td>
<td>signalType ∈ {DoNetSync, GiveStatus}</td>
</tr>
<tr>
<td>produce network synchronization</td>
<td>NetSyncRequest</td>
<td>NetworkTable</td>
<td>signalType = DoNetSync</td>
<td>true</td>
</tr>
</tbody>
</table>
### TABLE A.8: SERVICE MODEL OF THE NODEMANAGER AGENT

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>share SHM information</td>
<td>SHMInformationRequest</td>
<td>SHMDATA</td>
<td>signalType = GiveStatus</td>
<td>true</td>
</tr>
<tr>
<td>inform parent</td>
<td>Status ∨ SHMDATA</td>
<td></td>
<td>true</td>
<td>parent knows the status</td>
</tr>
</tbody>
</table>

### TABLE A.9: SERVICE MODEL OF THE NODERADIO

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>await reception</td>
<td>eventOccurs</td>
<td>actionType</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>check action type</td>
<td></td>
<td>actionType</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>produce radio transmission</td>
<td>transmission-Information</td>
<td>informDataReception</td>
<td>radio ≠ busy</td>
<td>true</td>
</tr>
<tr>
<td>produce acknowledgment</td>
<td>RadioData</td>
<td>acknowledge-ment</td>
<td>actionType = RadioData</td>
<td>true</td>
</tr>
<tr>
<td>inform Base Station</td>
<td>informOfDataReception</td>
<td></td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>

### TABLE A.10: SERVICE MODEL OF THE NODENETSSYNC

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>request of network synchronization</td>
<td>Request</td>
<td>StarNetSync</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
### TABLE A.10: SERVICE MODEL OF THE NODENETSYNC

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>produce network/synchronization data</td>
<td>Time</td>
<td>NetSyncData</td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
</tr>
<tr>
<td>produce radio transmission</td>
<td>NetSyncData</td>
<td>DataReceived</td>
<td>radio ≠ busy</td>
<td><strong>true</strong></td>
</tr>
<tr>
<td>produce synchronization</td>
<td>DataReceived</td>
<td>ConnectionTable</td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
</tr>
<tr>
<td>inform base Station</td>
<td>InformOfConnectionTable</td>
<td></td>
<td><strong>true</strong></td>
<td>Base Station is informed</td>
</tr>
</tbody>
</table>

### TABLE A.11: SERVICE MODEL OF THE NODECLOCK

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>request of clock</td>
<td>Request</td>
<td>Request</td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
</tr>
<tr>
<td>inform of time</td>
<td>Time</td>
<td>Time</td>
<td><strong>true</strong></td>
<td>Requester is informed</td>
</tr>
</tbody>
</table>

### TABLE A.12: SERVICE MODEL OF THE NODETIMER

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>start time</td>
<td>ScheduleTime</td>
<td>StartTime</td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
</tr>
<tr>
<td>get time</td>
<td></td>
<td>Time</td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
</tr>
<tr>
<td>request SHM</td>
<td>RequestSHM</td>
<td></td>
<td><strong>true</strong></td>
<td>StartTime = restart</td>
</tr>
</tbody>
</table>

### TABLE A.13: SERVICE MODEL OF THE NODESHM

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHM request</td>
<td>SHMRequest</td>
<td></td>
<td><strong>true</strong></td>
<td><strong>true</strong></td>
</tr>
</tbody>
</table>
### Table A.13: Service Model of the NodeSHM

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensing request</td>
<td>GetSensing</td>
<td>SensingData</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>calculate SHM algorithm</td>
<td>SensingData</td>
<td>DamageIndex</td>
<td>Sensing data &gt; N</td>
<td>N = N+1</td>
</tr>
<tr>
<td>check damage</td>
<td>DamageIndex</td>
<td>damageType</td>
<td>true</td>
<td>damageType ∈ {Damage, NoDamage}</td>
</tr>
<tr>
<td>in case of no damage</td>
<td>RequestResponseSHM</td>
<td></td>
<td>damageType = NoDamage</td>
<td>NoDamage ≠ nil</td>
</tr>
<tr>
<td>in case of damage</td>
<td>DamageIndex</td>
<td>RequestInquire</td>
<td>damageType = Damage</td>
<td>Damage ≠ nil</td>
</tr>
<tr>
<td>request inquire of parent and children</td>
<td>GetInquire</td>
<td>DamageIndexInquire</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>response SHM</td>
<td>DamageIndex</td>
<td></td>
<td>true</td>
<td>Parent is informed</td>
</tr>
</tbody>
</table>

### Table A.14: Service Model of the NodeSensing

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensing request</td>
<td>SensingRequest</td>
<td></td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>obtain sensing record</td>
<td>SensingRequest</td>
<td>SensingData</td>
<td>true</td>
<td>Records = Records + 1</td>
</tr>
<tr>
<td>save record</td>
<td>SensingData</td>
<td>SensingData = nil</td>
<td>(SensingData ≠ nil) ∨ (SensingData = nil ∧ ~ memoryspace)</td>
<td></td>
</tr>
<tr>
<td>SensingReply</td>
<td>Sensing Record</td>
<td></td>
<td>true</td>
<td>SHM is informed</td>
</tr>
</tbody>
</table>
## TABLE A.15: SERVICE MODEL OF THE NODEINQUIRER

<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>inquire Request</td>
<td>InquireRequest</td>
<td></td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>obtain damage index from parent and children</td>
<td>GetInformationMessage</td>
<td>DamageIndex</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>produce information</td>
<td>DamageIndex</td>
<td>InformDamage</td>
<td>(InformDamage = nil)</td>
<td>(InformDamage ≠ nil)</td>
</tr>
<tr>
<td>InquireReply</td>
<td>InformDamage</td>
<td></td>
<td>true</td>
<td>SHM is informed</td>
</tr>
</tbody>
</table>


