UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE ESTRATEGIAS DE AUTO-OPTIMIZACIÓN Y AUTO-ADAPTACIÓN PARA SISTEMAS DISTRIBUIDOS A GRAN ESCALA.

Francisco Penedo Álvarez
Tutor: Rodolfo Haber
Ponente: Pablo Varona

JULIO 2014

Abstract

Large scale distributed software systems are complex systems that need to be able to adapt to a highly dynamic environment and changing user needs. In this context, the main objective of this project is the development of new self-adaptive strategies, along with the methodologies and tools required for their analysis and design. In this work, we design and implement a self-adaptive architecture inspired by IBM's Monitor-Analyze-Plan-Act over a Knowledge base architecture, and we develop new self-adaptive strategies specific for wireless sensor networks following a methodology borrowed from control engineering.

More in detail, we start this work by designing in UML a software library for the development of self-adaptive capabilities, which will be implemented as a Java package. After that, we model two distributed software systems using an actor oriented approach in Ptolemy II. Next, we develop the self-adaptive strategies based on fuzzy inference systems and introduce them in the models as new actors. Finally, we are able to execute a simulation of the system, which allows us to perform an automatic optimization of the parameters of the sytem with the cross-entropy method and to test the suitability of the designed strategies.

Based on the simulation results, we have assessed the good results yielded by the strategies and the potential of the modeling tool for the design and simulation of distributed software systems. But more importantly, this work demonstrates the usefulness of a control engineering approach to solve problems related to the dynamic behavior of software systems.

Keywords: *self-adaptative software, feedback loops, actor model, wireless sensor network, optimization, control engineering, software engineering.*

Resumen

Los sistemas distribuidos a gran escala son sistemas complejos que necesitan adaptarse a un entorno altamente dinámico y a las distintas necesidades del usuario. En este contexto, el objetivo principal de este proyecto es el desarrollo de nuevas estrategias de auto-adaptación, a la vez que las metodologías y herramientas necesarias para su análisis y diseño. En este trabajo, diseñamos e implementamos una arquitectura para capacidades auto-adaptativas en sistemas software insipirada en la arquitectura *Monitor-Analyze-Plan-Act over a Knowledge base* de IBM, y desarrollamos nuevas estrategias de auto-adaptación específicas para redes de sensores inhalámbricas siguiendo una metodología tomada de la ingeniería de control.

Más concretamente, comenzamos este trabajo diseñando en UML una librería software para el desarrollo de capacidades auto-adaptativas, que luego implementamos como un paquete Java. A continuación, modelamos dos sistemas distribuidos usando un enfoque orientado a actores en Ptolemy II. Posteriormente, desarrollamos estrategias auto-adaptativas basadas en sistemas de inferencia difusa y las insertamos en los modelos como nuevos actores. Finalmente, ejecutamos varias simulaciones del sistema, lo cual nos permite realizar una optimización automática de los parámetros del sistema mediante el uso del método de entropía cruzada y, además, probar el desempeño de las estrategias diseñadas.

Basándonos en los resultados de estas simulaciones, hemos podido comprobar los buenos resultados que ofrecen las estrategias de auto-adaptación implementadas y el potencial de la herramienta de modelado para el diseño y la simulación de sistemas distribuidos. Pero lo más importante es que este trabajo demuestra la utilidad de enfocar desde la ingeniería de control la resolución de problemas relacionados con el comportamiento dinámico de sistemas software.

Palabras clave: *software auto-adaptativo, bucles de realimentación, modelo de actores, redes de sensores inhalámbricas, optimización, ingeniería de control, ingeniería de software.*

Acknowledgements

I would like to express my deep gratitude to Dr. Rodolfo Haber for giving me the chance of working with him for the past four years, and to Dr. Raúl del Toro for his guidance and supervision of this work. I would also like to extend my thanks to the researchers of the C4LIFE group for their help and support, as well as the previous work that has made this project possible. Special thanks should be given to Carmelo Juanes for his contributions in the early stages of this work and his valuable suggestions.

Table of Contents

Index of tables

Index of figures

Glossary

CE	Cross-Entropy
DE	Discrete Event
GUI	Graphical User Interface
IDE	Integrated Developing Environment
JVM	Java Virtual Machine
MAPE-K	Monitor-Analyze-Plan-Act over a Knowledge base
OLSR	Optimized Link State Routing
OSPF	Open Shortest Path First
SSE	Sum of Squared Errors
UML	Unified Modeling Language
WSN	Wireless Sensor Network

1. Introduction

1.1. Motivation

Since the middle of the 20th century, the economic growth experienced by the European region has led to an unprecedented increase in the quality of life of the citizens. However, this growth has had some negative consequences on aspects of the society such as the quality of life in urban environments, public security, environment protection and emergency management.

In order to address these challenges, one of the most promising technological advances has been the development of large scale monitoring and control systems in the form of networked adaptive software systems. Such a system is usually composed of embedded computing devices, defined by a low computing power but a high portability and availability, connected in a communication network while executing some kind of task. Their use have been promoted by the production of cheaper and more powerful hardware, as well as the development of more efficient communication systems and data mining techniques capable of dealing with the large amount of data produced by the monitoring tasks.

However, the widespread introduction of such large scale adaptive software systems has been slowed down by the lack of methodologies and techniques to be used for their design, control and operation. The highly dynamic behavior of such large scale distributed architectures, increased even further by the interaction with the environment and ever changing human needs, renders the knowledge gained from small scale or centralized systems insufficient. To overcome this, new self-adaptive capabilities such as self-optimization, self-organization and self-configuration need to be developed. The “self” prefix indicates that such operations are performed without (or with minimal) human intervention.

Therefore, the new techniques developed for the engineering of adaptive software systems should have these self-adaptive capabilities as their primary target. Given the multidisciplinary nature of the considered systems, a combined approach from different related research fields seems sensible. In this project, we present the result of combining the knowledge about dynamic systems, and the methodologies and algorithms used in their design and analysis, borrowed from the field of control engineering, with the tools for the design and implementation of software systems provided by the discipline of software engineering.

1.2. Objectives

The general objectives of this work are the following:

1. To design and implement a general architecture for self-adaptive strategies in software systems.
2. To design and implement new self-adaptive strategies based on the developed architecture.
3. To design and model large scale multi-agent architectures suitable for the application of self-adaptive strategies.
4. To evaluate the application of the self-adaptive strategies in the context of these large scale multi-agent architectures.
5. To acquire new knowledge and developing skills in topics outside the classical software engineering field, such as actor-oriented design and programming or control theory,

We highlight some specific goals derived from the main objectives:

1. To design and implement a general feedback control loop architecture which will serve as a basis for the implementation of the self-adaptive strategies.
2. To design and implement a self-adaptive strategy based on a fuzzy control system.
3. To design and model a WSN based on an actor-oriented approach.
4. To design and evaluate a methodology for the application of a self-adaptive strategy to a software system. This includes an optimization of the strategy for the specific instance of the system, which will be performed using the cross entropy method.
5. To acquire knowledge on the use of the actor-oriented modeling software Ptolemy II.

1.3. Structure of the document

The remainder of this document is structured as follows. Section 2 summarizes some of the related work in the area of self-adaptive software systems, as well as actor-oriented modeling. In section 3 we present the most important tools and algorithms we have used in this project. A detailed description of the design and implementation of the feedback control loop architecture, the WSN models and the self-adaptive strategies can be found in section 4. Section 5 contains the experimental evaluation of the work. Finally, we conclude in section 6, outlining future work not covered in this project.

2. State of the art

In this section we present a review of the state of the art in self-adaptive software, focusing on already proposed techniques for its development. We also discuss the approach we will take from the software engineering point of view, comparing different available tools, and conclude highlighting some of the specific tools available for the development of the concrete class of systems we are going focus on.

2.1. Self-adaptive software systems

Self-adaptive capabilities in software systems have been under research in several areas of the discipline of software engineering, such as requisite engineering [1], [2], software architecture [3], middleware architectures [4], component based development [5] and objective based models [6]. However, most of these works have been isolated efforts that have failed to address the issue at its core. Other research areas such as control engineering, artificial intelligence, robotics, distributed systems and machine learning have also studied self-adaptive topics and feedback from their own point of view.

The literature provides some techniques for the development of self-adaptive systems, some already tested while others have been just proposed. For example, Garlan et al. [7] have used external control mechanisms for the implementation of self-adaptive features, proposing an architecture based reusable framework for some specific systems.

The authors Bencomo et al. [8] have been taking a software engineering approach based on components developed in the object-oriented programming language Genie, which supports modeling, automatic generation and running of reconfigurable component based systems. A model-driven approach, based on the abstract definition of architecture models and corresponding tools for their translation to platform specific implementations, have been taken by Vogel et al. [9] for the development of self-adaptive systems with self-monitoring architectures. This approach can lead to incremental synchronization between the run-time system and models for different self-management activities. More examples can be found in the extensive survey conducted by Macías-Escrivá et al. [10].

In spite of the innovative value of the proposed initiatives, they constitute an isolated effort and do not suggest a methodology for the design of self-adaptive systems that integrates the analysis of the effects the introduction of feedback mechanisms could produce. The software engineering discipline has been mainly focused on static applications, often overlooking the dynamic side. On the other hand, control engineering emphasizes the use feedback loops, treating them as first class entities [11].

This is the starting point of this project, which will be focused on the combination of control engineering techniques with software engineering tools and methodologies with the goal of building intelligent self-adaptive systems able to adjust to dynamic environments.

2.2. Actor oriented modeling

Since the beginnings of software engineering, one of the basic research lines have been the development of different paradigms of computation, such as imperative, object-oriented, functional, feature-oriented or data-driven among others, each one conceived to provide a suitable framework for the engineering of different kinds of software solutions. In the case of distributed systems, concurrency is one of the key aspects that need to be addressed by the paradigm of choice, as well as the representation of different autonomous elements. The actor-oriented approach was specifically designed to allow the engineer to naturally build the system around these topics.

Actor orientation [12] is a way of structuring and conceptualizing a system that complements the object-oriented approach. Viewing a system as a structure of actors emphasizes its causal relationships and its concurrent activities, along with their communication and data dependencies. An object-oriented perspective sees the state of a system as a hierarchical collection of objects that are related by references to each other, communicating by calling each other's methods. However, an actor-oriented view of a system decouples the transmission of data from the transfer of control, which allows for different communication models to be used.

Being a similar paradigm as object orientation, some object-oriented tools can also be used for actor-oriented programming. One of the most widely used modeling languages, UML [13], and its derivative, SysML [14], provide a standard visual syntax that represents the actor hierarchy and their relations. However, they leave the semantics of the diagrams open, so that different tools compliant with the standard may give a different behavior to each component. Other derivatives of UML like MARTE (Modeling and Analysis of Real Time Embedded systems) [15] put more emphasis on the semantics of models, but avoids constraining them excessively in an effort to capture already existing design practices.

On the other hand, Ptolemy II [16] focuses on providing precise and well-defined models of system behavior. It also features the possibility of freely combining different semantics in the same model, allowing the modeler to describe completely different parts of the system, such as queues, communication networks and physical processes in the same model.

2.3. Wireless sensor networks

Wireless sensor networks have attracted special interest in the self-adaptive software research community, mostly due to the increased computing power and availability of the necessary devices and the development of cheap and efficient wireless technology. A WSN is a network composed of several autonomous nodes equipped with sensors that cooperate towards the completion of some assigned task by processing and passing their data over a wireless channel. They are usually employed as a solution for physical related problems over a large area that require limited computational capabilities, such as structural health monitoring [17], environmental monitoring [18], electrical power system enhancement [19] or gas sensing [20].

One of the basic challenges the development of these systems pose is the difficulty of testing the engineered solution, given the high cost of deploying the network and the usual impossibility of running an experimental system in a production environment. To address this issue, several simulating tools have been developed. NS-3 [21] is a free discrete-event network simulator for Internet systems that features an extensive library of hardware models and protocols, as well as performance analysis and graphical visualization tools. However, it is often criticized by the lack of a graphical user interface, which makes the modeling process complex and time-consuming.

Other simulators constrain their domain of application to some selected devices, such as TOSSIM [22], which is aimed towards the simulation of TinyOS wireless sensor networks. TOSSIM features a full communication stack simulation, allowing the experimentation of both low- and top-level protocols and applications. It is, however, limited to homogeneous networks where every node executes the same code.

Finally, besides its actor-oriented capabilities, Ptolemy II [16] features a limited extension tool, Visualsense, focused on the design and simulation of wireless networks. It provides a graphical user interface for the design of the system, as well as several premade actors related to wireless communications, such as collision detectors. On the other hand, it lacks low level models necessary for some applications and every protocol needs to be implemented by the user.

3. Tools and technology employed

We describe in this section the main tools used for the development of this project, as well as the most important technologies this project is based on. We start by describing the general architecture which our feedback loop framework will be based on, the IBM's MAPE-K. Next, two of the main algorithms used in this project are presented: the cross-entropy optimization method and the fuzzy inference system, which will be used as the core element of the self-adaptive strategies we will be developing. The design, modeling and simulation of the adaptive software systems will be carried out in the modeling tool Ptolemy II. Finally, we briefly comment on other supporting tools, such as the modeling language UML we will be using for the design of the library, the programming language of our choice, Java, and other complementary software.

3.1. MAPE-K feedback loops

In order to cope with uncertainty and complexity when managing dynamic processes, a great number of mathematical models, tools, and techniques for the analysis of system performance, stability, sensitivity or correctness have been developed under the field of control theory. A key design element in this kind of control systems are feedback loops such as the one depicted in Figure 1. Feedback allows the system to self-adapt by being aware of the effect its actions have on the process.

In an effort to apply the control theory approach to software engineering, IBM released a blueprint for building autonomic systems using MAPE-K (Monitor-Analyze-Plan-Execute over a Knowledge base) feedback loops [23]. This systems are build from autonomic element blocks, depicted in Figure 2, which consist of a managed element and an autonomic manager.

A managed element is a software or hardware component that can be managed through a series of touchpoints. The touchpoints can be used to gather, aggregate and transmit data from the managed element to the autonomic manager (sensor) or to provide a mechanism to change the behavior of the managed element (effector).

An autonomic manager, which may also be a managed element, implements intelligent control loops that can perform different self-adapting tasks, such as self-configuring, self-healing, self-optimizing or self-protecting tasks. It will usually use policies in the form of goals or objectives to govern the behavior of the control loops, making it possible for another autonomic manager to change them via its touchpoints. This allows for the change of policies to be automated and designed in the same way as any other change in the system.

The architecture of an autonomic manager is dissected in four different parts:

- Monitor: The monitor function collects details from the managed resources, via the sensors, and resumes them into symptoms that can be passed to the analyze function. The data collected can include sampled environment information, configuration settings, internal status and so on. The monitor function aggregates and filters these details until it finds a combination worth analyzing.

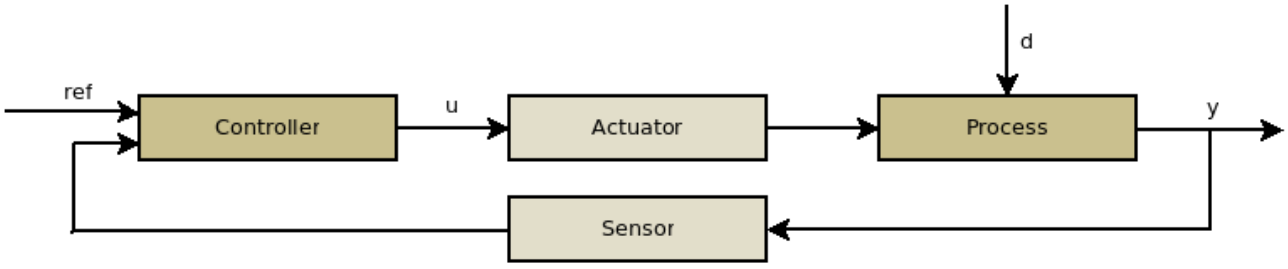

Figure 1: Feedback control loop

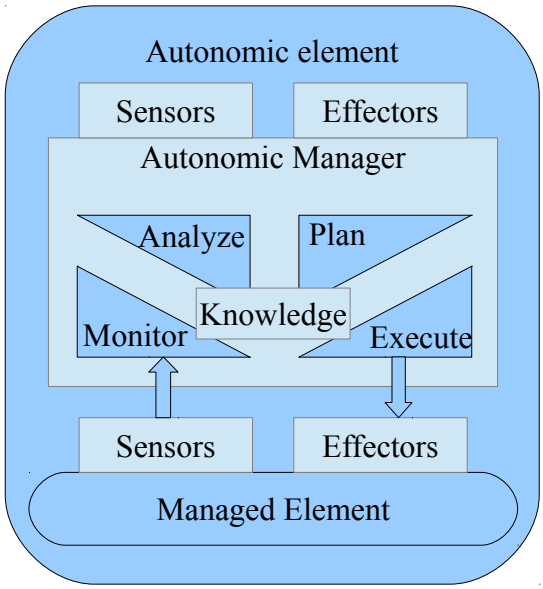

Figure 2: IBM's MAPE-K feedback control loop model

- Analyze: The analyze function provides the mechanisms to examine the symptoms detected by the monitor function in order to determine if some change needs to be made. Usually, this function models complex behavior so it can employ prediction techniques that help in determining if the autonomic manager can abide by the established policy.

  If the analyze function concludes that a change needs to be made, a change request is sent to the plan function containing the modifications this function deems desirable. We will also call this function the reasoner.

- Plan: The plan function selects a procedure to apply the changes requested by the analyze function. The change plan is then sent to the execute function.

- Execute: The execute function provides the mechanism to schedule and perform the change plan created by the plan function to the system, using the effector interface. In some cases, this function is too coupled with the plan function to be worth the effort of describing them separately. This happens when the change procedure is so simple that it contains by itself the means for the execution. This will be the case in some of the feedback loops designed in this project and we will call the joint function the actuator.

The original architecture defines a common knowledge base shared among the different parts of the autonomic manager. The knowledge data can include information about the system, active policies or problem determination knowledge, such as monitored data or symptoms and the internal state of the manager. However, for the purpose of this project, the knowledge base will not need to be of such complexity and we will restrict the access of a part of the autonomic manager to the knowledge of some other part.

This architecture will be the basis for the design of the self-adaptive strategies we will be developing. The adoption of this specific architecture comes from the control theory approach to the software engineering process that it represents, which is one of the main objectives of this project. Several implementations of the MAPE-K loop exist, such as IBM's autonomic toolkit [24] and ABLE [25]. However, the projects are not in active development and are not distributed under an open source license, which would make it difficult to solve possible limitations like lack of non-local communications or incompatibility with specific embedded systems.

3.2. Cross entropy

The cross entropy (CE) method [26] is a general Monte Carlo approach to multi extremal optimization. It was first introduced as an estimation algorithm for the probability of rare events and it was adapted later as an optimization method as a function can be thought of a system where sampling around the optimum is a rare event. We present here a detailed description of the method.

Let X be a random variable defined on the space $\mathcal{X}$ and $f:\mathcal{X}\to\mathbb{R}$ a score function. Then the CE method tries to find x' such that

$$\gamma' = f(x') = \min_{x\in\mathcal{X}} f(x) \tag{1}$$

The algorithm transforms this problem into an associated stochastic problem by defining a family of random variables with density function $g(x,v), v\in\Gamma$ and solving it as the simulation of a rare event, where the event is sampling around the optimum of *f*. The algorithm can be summarized as follows:

1. Initialize v_0.
2. Generate a sample of size *N*, $\{x_i^t\}_{1\le i\le N}$, from the density function $g(x,v_t)$. Let $f_1\ge f_2\ge\ldots\ge f_N, f_i\in\{f(x_i^t)\}, 1\le i\le N$ be the corresponding ordered score values and $\gamma_t = f_{[\rho N]}$.
3. Update v_t to

$$v_{t+1} = \underset{v}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} I_{\{y\le\gamma_t\}}(f(x_i^t))\cdot\ln g(x_i^t, v_t) \tag{2}$$

4. Repeat from step 2 until convergence or ending criterion.
5. Assuming that convergence has been reached at $t=t'$, the random variable defined by the density function $g(x,v_{t'})$ should have all of its mass concentrated on x'.

Step 3 is performed using the best $[\rho N]$ samples, also called elite samples. The sampling density function needed in (2) is usually unknown, but in most cases it can be assumed to be normal. In this case, *v* is the mean μ and the standard deviation σ of the normal distribution and the solution of the equation is simply the sample mean $\bar{\mu}_t$ and sample deviation $\bar{\sigma}_t$ of the elite samples. It also follows that the mean should converge to x' and the deviation should converge to zero. A smoothing parameter α for the mean vector and dynamic smoothing β_t for the deviation are applied in order to prevent the occurrences of 0s and 1s in the parameter vectors.

$$\begin{aligned}
\bar{\mu}_{t+1} &= \alpha\bar{\mu}_{t+1} + (1-\alpha)\bar{\mu}_t \\
\bar{\sigma}_{t+1} &= \beta_t\bar{\sigma}_{t+1} + (1-\beta_t)\bar{\sigma}_t \\
\beta_t &= \beta - \beta\left(1-\frac{1}{t}\right)^q
\end{aligned} \tag{3}$$

Where $0.4\le\alpha\le 0.9, 0.6\le\beta\le 0.9, 2\le q\le 7$.

Finally, as we will be dealing with constrained optimization problems, we will need to use a bounded distribution for the sample drawing, so that the samples lie within the acceptable region. The most straightforward to use is the truncated gaussian distribution, as the updating rules remain the same (see [27]). The density function of the truncated gaussian distribution with mean μ, standard deviation σ and bounded between *a* and *b,* for $a\le x\le b$ is defined as follows:

$$f(x;\mu,\sigma,a,b)=\frac{\frac{1}{\sigma}\varphi\left(\frac{x-\mu}{\sigma}\right)}{\Phi\left(\frac{b-\mu}{\sigma}\right)-\Phi\left(\frac{a-\mu}{\sigma}\right)} \tag{4}$$

And by $f = 0$ otherwise. Here, φ is the density function of a standard normal distribution and Φ is its the cumulative distribution function.

3.3. Fuzzy control systems

Fuzzy Logic [28] is a powerful modeling tool in situations of essential uncertainty in models, information, objectives, restrictions and control actions, that tries to emulate the decision making process of humans. It uses vague logical statements to derive vague inferences from imprecise data.

One of the most important concepts in Fuzzy Logic is that of a linguistic variable. A linguistic variable is characterized by a tuple (X, T_e (X), D, M) where X represents the name of the variable, T_e (X) is the set of linguistic values (attributes, adjectives) of X, D is the universe of discourse and M is the semantic rule associating each value with its meaning or membership function. The membership function for each linguistic value is a function in [0, 1] from the universe of discourse D and defines a fuzzy set.

A fuzzy set represents a vague classification. For example, the variable temperature may include the linguistic values “hot” and “very hot”, with a fuzzy set associated to each of them. Then, we are allowed to classify a deterministic value of the temperature as “between hot and very hot” with a precise meaning in terms of membership functions. Usually, these functions are built from piece-wise linear functions, gaussian distribution functions, sigmoid curves and quadratic and cubic polynomials.

A brief description of the fuzzy inference process [29] is now presented. For the sake of simplicity, we describe the process for a single output variable. A visual overview of the system is shown in Figure 3.

1. Fuzzyfication. For each input variable, the membership functions associated are computed.
2. Decision-making. In this step, the fuzzy inference process is performed.
    1. Antecedent composition. For each rule h, the antecedent is a logical formula composed of several terms of the form “X is V”, where V is a linguistic value, connected by the logical operators AND, OR and NOT. The fuzzy value of the terms have been obtained in step 1. Each logical operator is interpreted as a fuzzy set operation (intersection, union and complementation, respectively) and the result of these operations is the value of the antecedent, A_h. Any number of operations can fill the role of the fuzzy set operations, so long the intersection is a T-norm, the union is a complementary T-conorm and the complementation maps x to $1 - x$ [30]. The most widely used pairs are (min, max) and (product, probabilistic sum).

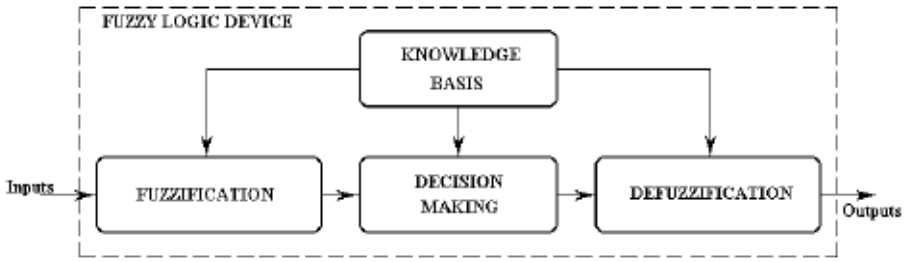

Figure 3: Block diagram of a Fuzzy Logic Inference System

2. Implication. For each rule h, the consequence fuzzy set, B_h, is reshaped by an inference operation $\otimes$, resulting in the following fuzzy set:

$$R_h = A_h \otimes B_h \tag{5}$$

3. Aggregation. The fuzzy sets representing the output of each rule are combined into a single fuzzy set, possibly taking into account different rule weights, which we will call W_h. The aggregation function, $\coprod$, should be a T-conorm. The resulting output fuzzy set is:

$$R = \coprod_h W_h R_h \tag{6}$$

3. Defuzzyfication. The output fuzzy set resulting from step 2 is not yet suitable for use. A single deterministic value should be computed that represents the fuzzy set. Some of the most popular defuzzyfication methods are the mean value of maximum and the center of area.

3.4. Ptolemy II

Ptolemy II [31] is an open-source software framework supporting experimentation with actor-oriented design. Actors are software components that execute concurrently and communicate through messages sent via interconnected ports. A hierarchical interconnection of actors form a model, depicted in Figure 4. In Ptolemy II, the semantics of a model are not determined by the framework, but rather by a software component in the model called a director, which implements a model of computation. Ptolemy II supports dataflow, process networks, synchronous-reactive, finite state machines, discrete event and continuous time models.

Although this ability to design a full system from the composition of different models is one of the key features of the Ptolemy II software environment, we will mainly be making use of one of the models of computation it supports, the Discrete Event (DE) model. In this model, two actors interact by interchanging messages placed on a time line, called events. Each event has a value and a time stamp, and actors process events in chronological order. The output events produced by an actor are required to be after or at the same time as the input events consumed.

The execution of this model uses a global event queue. When an actor generates an output event, the event is slotted into the queue according to its time stamp. During each iteration of a DE model, the events with the smallest time stamp are removed from the global event queue, and their destination actor is fired. This differs from other usual time models such as continuous time or statistical simulation where each iteration makes an approximation of the system evolution during an associated time step.

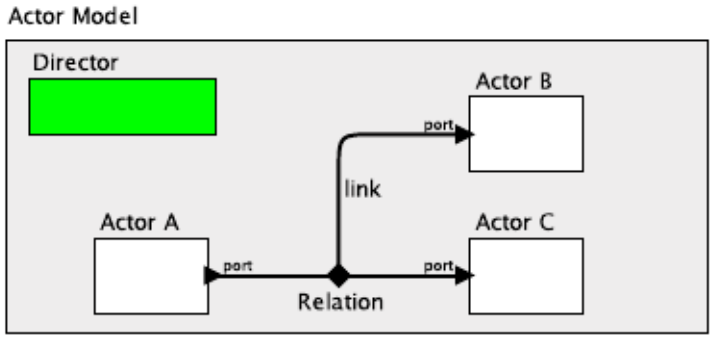

Figure 4: Visual representation of an actor model. Borrowed from Ptolemaeus [53]

This is the model that best suits the description of the communication aspects of WSN, which will be the focus of this work, as it directly maps to the message passing processes of a communication network. Ptolemy II includes a set of tools, such as a graphical editor and several actors, called Visualsense which is intended to be used specifically for the design of wireless networks. One of its key features is the channel actor, which models some characteristics of a wireless channel, such as power loss or random losses, and implements the dynamic connection of actors transmitting and listening to the channel using their positions.

This is the main reason supporting our choice of Ptolemy II and Visualsense as the design and simulation tool that we will be using to model our different WSN scenarios. We have also valued the lack of low level specification that needs to be done in comparison to other network simulators such as NS-3.

3.5. Java

Java [32] is an object-oriented compiled computer language specifically designed to be cross-platform. Java applications are compiled to bytecode that can run on any Java Virtual Machine (JVM) regardless of the underlying platform, allowing the programmer to "write once, run anywhere". This feature, combined with a familiar C++ like syntax and a big ecosystem of tools, frameworks and libraries has made it one of the most popular programming languages in use.

For the development of this project we will be using the version 1.6 of the Java language. Besides the highlighted characteristics above, the main reason supporting our choice is the straightforward integration with Ptolemy, which is completely written in Java and allows an easy extension or modification of its components.

3.6. UML

The Unified Modeling Language (UML) [13] is a well-known general-purpose modeling language designed to provide a standard way to visualize the design of a system. We will be using it as the main design tool for the development of this project.

3.7. Other tools

The coding phase of the development will be carried out with the aid of the IDE Eclipse [33], which is specially designed to speed up the process of coding in Java, has special plug-ins for UML design and automatic code generation, and automatizes the compiling process of the project, removing the need for manually writing compiling scripts.

In order to maintain a control on the revisions of the project, the software Git [34] has been used. As the development process will be done for the most part by a single person, Git provides a very fast and easy to setup local repository, requiring almost no time to perform any operation on it. We will be backing up the repository with a synchronized copy uploaded to the service Dropbox [35].

Most of the data analysis and graphic representation will be carried out in the software environment R [36], one of the most popular tools for statistical computing.

Finally, this document has been written in the office suite LibreOffice [37], with the help of the bibliographic database manager Zotero [38] and its LibreOffice plug-in.

4. Design and development

In this section, we describe the design of the different scenarios for which we will develop self-adaptive strategies and the framework we will use for the implementation of those strategies. In subsection 4.1, we develop a software implementation of the conceptual MAPE-K architecture, which we will use as a framework for the design of self-adaptive strategies. We will follow the standard software engineering practice of explicitly stating and analyzing the functional and non functional requirements of the software, and then proceeding with the design phase, which we will carry out from an object-oriented point of view. In subsection 4.2, we model two WSN scenarios: a sensor network aimed to the detection and measurement of a contaminant and a generic multi-hop communication network. Finally, in subsection 4.3, we design and model, following the MAPE-K architecture, a self-adaptive strategy for each modeled system based on fuzzy inference systems and we propose an optimization methodology on design time using the cross-entropy method.

4.1. Self-adaptive architecture

The purpose of a software architecture for self-adaptive systems is to provide a common framework for the development of the different self-capabilities of the system, giving them an individual entity and putting them on the same level as other elements such as data bases, communication protocols and interfaces during the analysis and design phase. We present here the requirements we want to fulfill in the scope of this project and the subsequent design and implementation phase.

4.1.1. Requirement analysis

We start by presenting a series of requirements for a self-adaptive architecture design library based on the MAPE-K architecture. These requirements have been established taking into account what purpose it serves and how the architecture accomplishes it.

Functional requirements

(FR1) Control loop design: The library must focus on the design and implementation of control loops. The library must produce a fully functional control loop based on the implementation of single steps (such as reasoning or planning steps) provided by the user, taking care of the necessary bookkeeping, such as chaining the steps together.

(FR2) Modules: The library must provide the user with the ability to design a control loop with any number of single steps (modules). The library must not impose any restriction on the number of the modules nor the expected functionality of any of them. The library must manage the input/output relations between the modules according to the specification of the user.

(FR3) Managed controllers: The library must allow a control loop to act on control loops. The library must treat control loops as the same kind of entity as controlled elements, providing an homogeneous interface to both. It must allow the user to specify the actions allowed on a control loop, as well as what data can be sampled from it.

(FR4) Knowledge base: The library must provide a database for each control loop. The control loop modules must be able to store and retrieve data from the knowledge base.

(FR5) Data types: The library must be able to handle any kind of data types as inputs, outputs and knowledge of a control loop. This includes but it is not limited to: numeric values, nominal values, collections of values and arbitrary objects

describing complex actions or sample data (such as a complete process scheduler or a topology description of a network). The library must be able to ensure that the declared data type of an attribute is equal to the data type of the value it is storing at runtime.

(FR6) Data flow modes: The library must allow the data flow to be by demand or by observation. The user must be able to design a control loop or a module to receive inputs by either requesting data or listening to broadcasted data from its predecessor in the data flow.

Non functional requirements

(NFR1) Adding data types: The user must be able to easily extend the library to manage more data types. As the user may need the control loop to use complex custom data types, the design of the library must allow the introduction of such types by implementing a simple wrapper with little functionality over the user data type.

(NFR2) Using middleware: The library must be designed and implemented in such a way that it allows an easy integration with middleware. As the library is intended to be used in systems that usually need a layer of middleware, it should be easy to adapt the library to use the middleware layer for the communication between its units (modules, control loops, managed elements and knowledge bases). It should be noted that it is not required for the library to be already integrated with a middleware, as different applications would use different ones, but it is required that the user only needs to implement simple wrappers for the components of the library in order to use it over any middleware.

4.1.2. Design

In order to fulfill the requirements above mentioned, an object oriented library has been designed. The functional requirements related to data management have been addressed in the data package, while the requirements related to the self-adaptive architecture itself have been addressed in the autoelem package.

The data package

The data package contains the classes needed to store and send typed data such as process sensor data or algorithm parameters. For an overview of all classes contained in the package, see Table 1. For a UML class diagram of the package, see Figure 5.

There are three classes and an interface related to the representation of data, namely Instance, Attribute, Instances and DataType. All of the other classes and interfaces are related to the transmission of data, as well as reading and writing files.

Actual data is stored in an Instance object. Each single piece of data is represented by a double precision floating point number and an Instance object holds a group of heterogeneous data. An Instance object represents the row of a data table, while the table itself is represented by an Instances object. The Attribute class holds information of each column of the table, such as name or type. Finally, the DataType interface is used to handle new custom data types.

Representing data as a Double for non-numerical data types works by first storing the real values into an array (held by an Attribute object), then storing the map from the value into its index in the array and finally storing the index as the value in the Instance object. This approach is memory efficient, but makes it more difficult to use it in a distributed environment. This issue is partially addressed in the autoelem package implementation.

Class	Description
DataSink	Interface of a data sink
DataSource	Interface of a data source
AbstractDataSource	Base class for data sources
AbstractDataSink	Base class for data sinks
DataType	Interface for custom types
IObservable	Basic interface of an Observable object
AttributeParsers	Parsers and writers for attribute declaration and values of attributes in string format
FileDataSink	Class for writing data files in a modified version of ARFF format
FileDataSource	Class for reading a data set from a stream in a modified version of ARFF format
Instance	Class for storing the values of an instance from a data set
Instances	Class for storing a list of objects of type Instance
ObservableObject	Base class for observable objects

Table 1: Overview of data package

The purpose of the DataSource and DataSink interfaces, as well as the abstract classes directly implementing them, is to provide a common interface for both ends of a link in a data flow* chain. This means that every process, manager, manager module, sensor, effector, etc, will* be either a data sink, a data source or both. Sinks and sources may behave in the following two ways, as required by (FR6):

- By demand: reading and writing are performed when someone asks for it, ie, by using the *read* and *write* methods. This is the preferred mode for file management.
- By observation: reading is performed as the source is able to, for example as a process sensor is able to get its data from the process. Writing is performed as the sink receives data from the sources it's listening. This is the preferred mode for message passing.

The data package already implements a number of basic required data types: real and integer numbers, nominal values, string values and relational values (data matrix). Nominal values should be considered when a new data type representing a fixed set of values is needed (for example, a fixed number of distance functions), translating the nominal value to the actual value upon use.

If the new type doesn't have a fixed set of values (for example, parameterized distance functions), a DataType implementation will suffice, which only requires the implementation of an equals function, as required by (NFR1). The comparison function will make sure that no values of other types are set for a given attribute. This works by comparing with a “representative” value used in the construction of the Attribute, so it is even possible to use the same class for different types (Instances is such a class: different headers correspond to different data types). This meets the requirement (FR5).

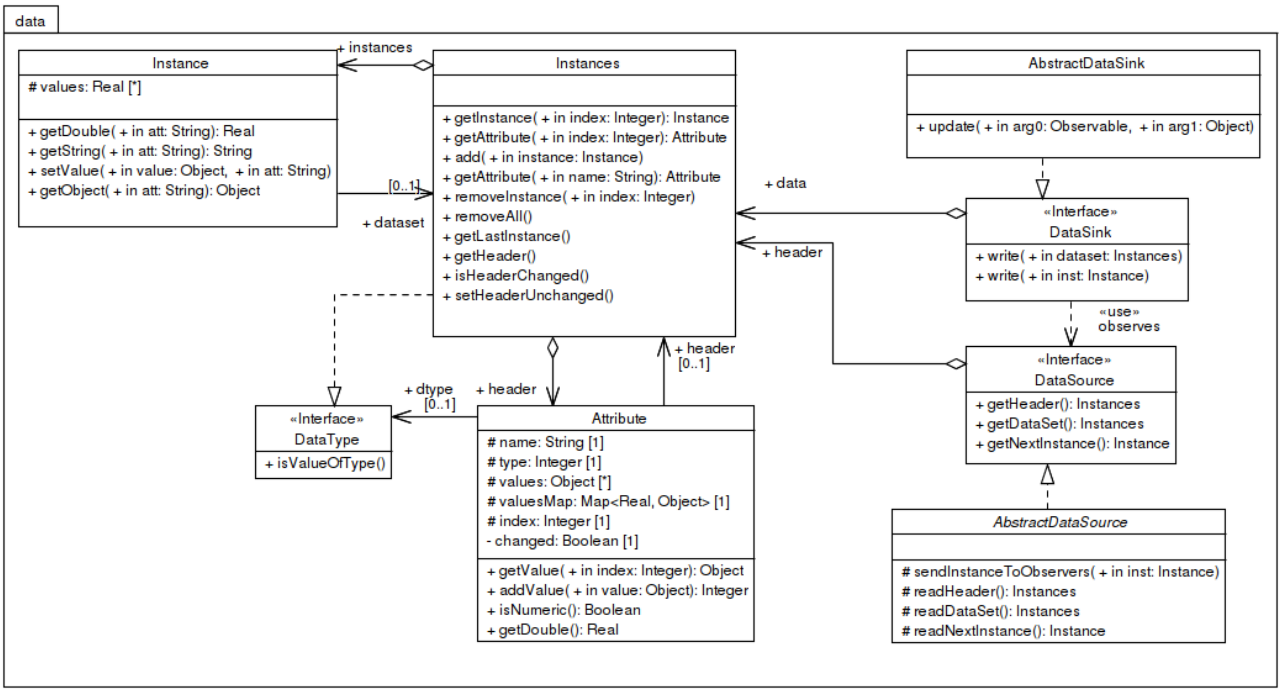

Figure 5: UML class diagram of data package

The autoelem package

This package contains the core classes of the architecture and implements all the communication logic between processes, controllers and controller modules. For an overview of the package, see Table 2. For a complete UML diagram, see Figure 7. In Figure 6, an overview of the data flow between the different components is depicted.

Class	Description
AutonomicManager	Class for running an autonomic manager.
AutonomicManagerEffector	Effector of an AutonomicManager.
AutonomicManagerSensor	Sensor of an AutonomicManager.
Effector	Base class for effectors.
KnowledgeBase	Class for storing all the knowledge from an AutonomicManager.
ManagedElement	Class for managed elements.
ManagerModule	Class for a manager module.
Sensor	Base class for sensors.

Table 2: Overview of autoelem package

A process (an industrial robot, a drone, a lighthouse, a software system...) is represented by a ManagedElement and a Sensor/Effector pair. The job of the Sensor is to send data obtained from the process translated into the representation given in the data package, while the job of the Effector is to send data created by the controller to the process.

The control loop is represented by an AutonomicManager (meeting the requirement (FR1)), which is a ManagedElement by itself, providing a unified interface to act on and sense from a control loop

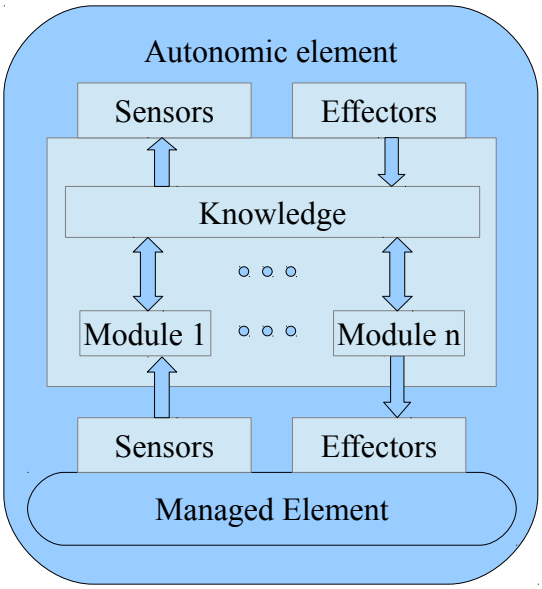

Figure 6: Diagram representing the dataflow between the components of the architecture

as required by (FR3). The controller is composed by several ManagerModule objects, which contain the actual controller logic and are not limited in number or function (required by (FR2)), and a KnowledgeBase to store all of the information needed by the controller (including configuration data). As a ManagedElement, an AutonomicManager have a special Sensor/Effector pair (AutonomicManagerSensor and AutonomicManagerEffector) that is only able to sense or act over the KnowledgeBase data.

Communication between components of the architecture follows an event driven, message passing pattern. A source (data.DataSource) performs some computation (such as a Sensor getting process data or a ManagerModule executing some clustering algorithm), packs some useful data in an Instance object and then sends it to its observers.

The library implements the message passing logic so that it works efficiently even in a distributed environment. As stated previously, the implementation of the data package is memory efficient but not as good if used over a middleware: an Instance is only meaningful if it has access to the header (Instances), so it can translate double codes into actual values, so the full header should be transmitted along with the Instance every time. If headers don't change, there's no need to send them over middleware, which is the default implementation of the communication logic given in the classes AbstractDataSource (method sendInstanceToObservers()) and AbstractDataSink (method write()). This is the main difficulty to overcome in order to meet the requirement (NFR2), as the modular design of the library allows for easily implementing middleware wrappers over the main classes.

The knowledge base design deserves special mention, as it has been simplified from the original MAPE-k specification while still meeting the requirement (FR4). In particular, each module is given two entries on the knowledge base storing its configuration and its shared data. However, the module does not share it with other modules but with (possibly) other elements acting on the controller. This is done to enforce the design of modules in isolation from each other, so new modules can be easily added, removed or swapped. Thus, data sharing between modules must be done by sending it over an input/output relationship.

In order to share the contents of the knowledge base and provide the means to sense from and act on a control loop the library provides a special sensor/effector pair for an AutonomicManager: AutonomicManagerSensor and AutonomicManagerEffector. This classes share certain attributes stored in the knowledge base of the controller and modify them on demand. The user may define what attributes are shared or acted on by constructing a map of data set indexes (assigned to each

knowledge entry of each module) into the desired attribute indexes of each set. With this, the requirement (FR3) is fulfilled by the design of the library.

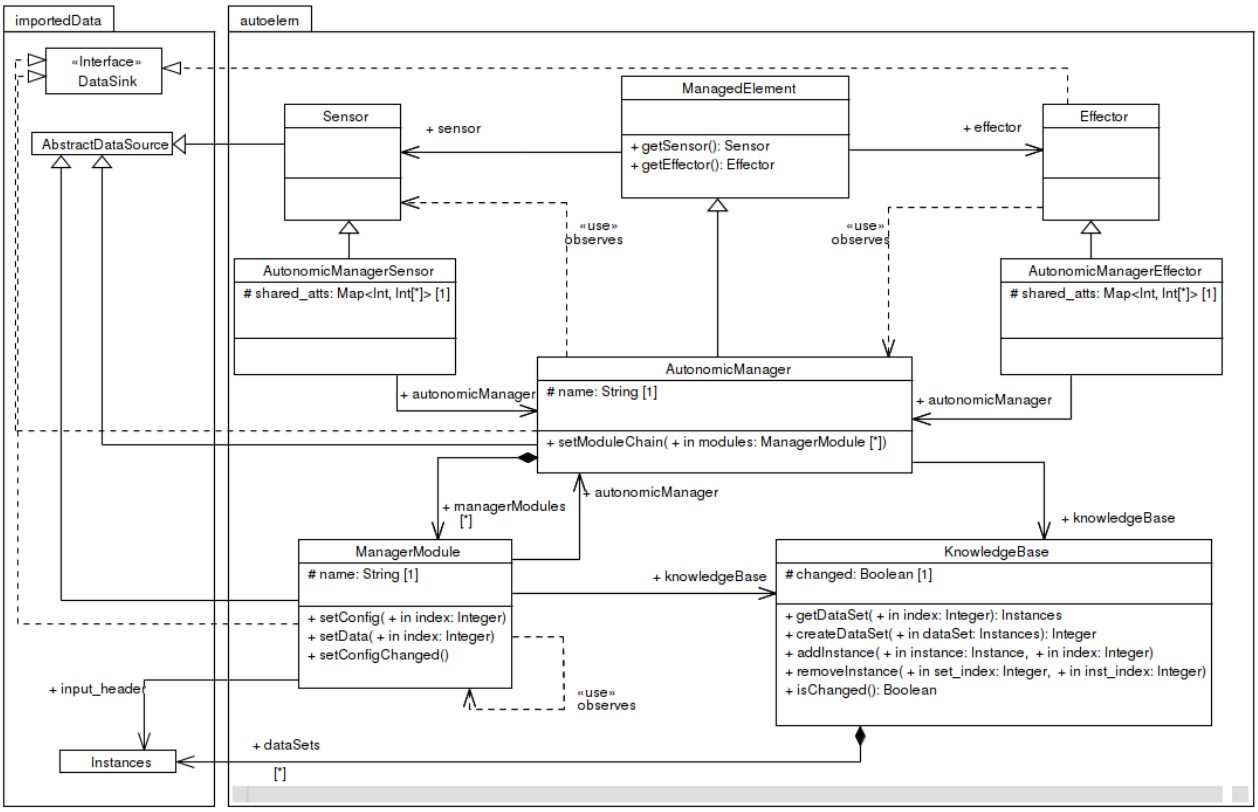

Figure 7: UML class diagram for autoelem package

4.2. Use-case scenarios

We describe here the modeling process of two software systems that will form the basis which we will develop new self-adaptive strategies upon. In the first one, a wireless sensor network is designed for the detection of a gas contaminant. Accuracy and battery saving will be the primary objectives of the adaptive algorithms. In the second one, we model a multi-hop communication network over a wireless channel with power loss. In this case, the main focus of the self-adaptive strategy will be maintaining the connectivity on the network and reducing the battery consumption.

Before discussing the particular details of these scenarios, we present their core elements and some common characteristics of the systems.

Area of operations and environment

The system is restricted to a limited area, which will be considered two dimensional in order to simplify the modeling phase. It can optionally be discretized by considering a grid over the surface.

The environment accounts for any processes that can interact with the rest of the elements. For example, a dynamic physical property of the area of operations, such as temperature or humidity, can be sampled and affected by other elements of the scenario.

Nodes

The system basic working component. Nodes are assigned tasks to perform locally that contribute to the overall goal of the system. It is not required for all nodes to do the same task nor that each node executes the same task all the time. However, we will consider simple homogeneous node systems, as heterogeneous systems require a more elaborate approach (see for example [39] and [40] for a solution addressing the task-device mapping problem). The following properties are key for the chosen scenarios:

- Position and movement: each node has a defined position in the area of operations, which may change over time.
- Battery: each node has a battery supporting its actions. Although there are accurate models proposed in the literature (for example, see [41] for a lithium-ion battery model), they are intended to be used for circuit level simulations. Instead, very simple models that only take into account the most relevant tasks of the node would be used.
- Decision element: in order to improve the execution of its assigned task, as well as to improve the overall behavior of the system, each node must be able to execute some kind of decision mechanism. This element will be able to use the state of the node and the external information sensed in order to alter the operation of the node, both related to the internal operation of the node and the execution of the assigned task.

Base stations

Specialized nodes that serve as data collectors. Its position may be relevant for some scenarios, but they lack a battery model (it is assumed to be an infinite supply) and decision element.

Communications

Communication links will serve as the edges of the network. The main communication on the network will be carried over a wireless lossy channel with power loss and collisions, although secondary communication can be handled by simpler channels if it is out of the scope of the scenario. If needed, a routing protocol suited for wireless networks will be part of the communication protocol.

4.2.1. Gas detection sensor network

In this section we describe a gas detection scenario based on [42]. The purpose of this model is to be used as a proof of concept for both the MAPE-K based architecture design and the Ptolemy modeling framework. Therefore, the scenario will be heavily simplified when compared with a real case and the decision element will be very simple.

The most important parameters of the model are discussed in this section, but a complete description of each parameter is provided in Appendix A.

Description

In this case, a network composed of a number of sensor nodes deployed in a region will try to predict the concentration of moving gas particles in the air, using their own samples and information gathered from the network. A base station will collect and merge the data, producing a single estimation from the network.

The possible objectives of the scenario are the following:

- Maximize the accuracy of the estimation of the gas concentration.
- Maximize the life time of the network. Alternatively, minimize node battery usage.

Area of operations and environment

The region of *AreaWidth* by *AreaLength* will be divided in a squared grid, each cell of *CellSize* side. Each cell will be named by a tuple (*x,y*), being *x* the coordinate in the west-east axis, *y* the coordinate in the north-south axis and (0,0) the most north-west cell. Two elements of the scenario will refer to the grid:

- A node's position (occupying the north-west corner of the cell for all purposes).
- The gas concentration will be simulated and estimated for each cell.

The gas concentration will be part of the environment of the scenario. It will be represented as a function of time as $\rho^q(t)$ (measured in *μg/cm³*), being *q* a cell from the area grid. Its propagation will follow the model presented in [42] and is given by the following equation:

$$\rho^q(t+\delta t)=e^{\alpha \delta t}\rho^q(t)+\delta t\, B\cdot V^q(t) \qquad (7)$$

where α is a coefficient related to the gas not dispersed by the wind, $B=[\alpha_n,\alpha_s,\alpha_w,\alpha_e,1]$ is a vector of coefficients related to the speed of the wind from each direction, $V^q=[\rho^{q_n},\rho^{q_s},\rho^{q_e},\rho^{q_w},u^q]$ and u^q is the the amount of gas created in cell *q* over a unit of time. Gas concentration updates at intervals of *RefreshPeriod* seconds. Figure 8 represents the monitored area

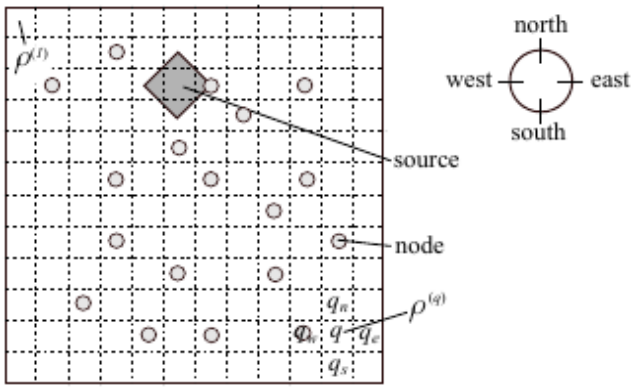

Figure 8: Graphical representation of the monitored area. Borrowed from Sijs and Papp [42]

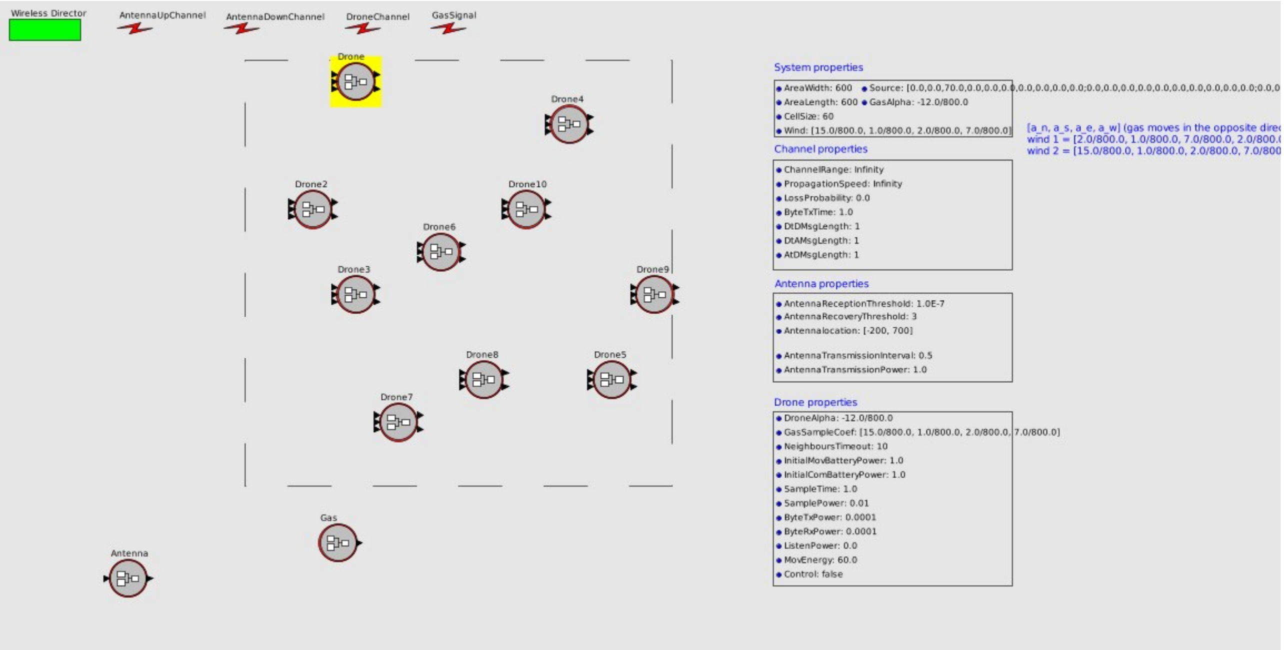

Figure 9: Top level of the actor model of the gas propagation scenario

and clarifies the choice of notation. In Figure 9 we show the final model of the scenario as seen in Ptolemy II / Visualsense.

Nodes

A node, also called *Drone*, is assigned the task of sampling the gas concentration in the cell it is located, computing an estimation of the concentration in neighboring cells based on his sample and data gathered from his neighbors, and sending his estimation to the base station and other nodes. The actor model of the *Drone* is depicted in Figure 10.

Regarding the estimation process, a *Drone* will use the same gas propagation model as the environment (equation (7)) in order to estimate the gas concentration at any given time for each cell. In addition, it will be able to merge in new information about the gas concentration, such as samples made by itself and estimations from other *Drones*. As complex data merging algorithms such as those discussed in [42] are beyond the scope of this project, a simple algorithm based on the confidence level of the estimation will be used. For each cell q and time t, we define the quality of the cell, $Q^q(t)$, such that:

1. $Q^q(t)=0.95$ if the cell q has been sampled by the *Drone* at time t. The loss of quality accounts for a possible measurement error.
2. Let A_q be the set of adjacent (not diagonally) cells of q and $\tilde{Q}^q(t+\delta t)=e^{-\lambda\delta t}Q^q(t)$ a temporal decay applied to the quality of a cell, with $\lambda>0$. Then,

$$Q^q(t+\delta t)=\frac{\tilde{Q}^q(t+\delta t)+\sum_{r\in A_q}\tilde{Q}^r(t+\delta t)}{1+|A_q|} \tag{8}$$

   if the cell has not been sampled at the given time. This is just the arithmetic mean of the quality of the measures used in equation (7) when estimating the gas concentration.

We can now define a simple merging algorithm as the weighted average of the estimations. Thus, let $Q'^q(t)$ be the quality of the estimation received by other *Drone* and $\rho'^q(t)$ the estimation of said *Drone*. Then, the merged estimation and quality is given by:

$$\begin{cases} \bar{\rho}^q(t+\delta t) = \dfrac{\rho^q(t+\delta t) * \tilde{Q}^q(t+\delta t) + \rho'^q(t+\delta t) * \tilde{Q}'^q(t+\delta t)}{\tilde{Q}(t+\delta t)^q + \tilde{Q}'(t+\delta t)^q} & \text{if } \tilde{Q}'(t+\delta t)^q > \epsilon \\ \bar{\rho}^q(t+\delta t) = \rho^q(t+\delta t) & \text{otherwise} \end{cases} \quad (9)$$

$$\begin{cases} \bar{Q}^q(t+\delta t) = \dfrac{\tilde{Q}^q(t+\delta t) + \tilde{Q}'^q(t+\delta t)}{2} & \text{if } Q'^q(t+\delta t) > \epsilon \\ \bar{Q}^q(t+\delta t) = Q(t+\delta t)^q & \text{otherwise} \end{cases} \quad (10)$$

Where ϵ is the minimum required quality. It should be noted that the last update for the receiver and the sender may have happened at different times, so different update intervals should be used. After the merge is done, the *Drone* will update its estimation and quality tables to the merged ones.

Besides the estimation mechanism, the other relevant *Drone* process we will model in this scenario will be battery use. The battery model considered for the nodes will be based on the assumption that the only relevant, battery consuming tasks are sending, receiving and listening for messages, and sampling the gas concentration. The state of the battery is represented by the ratio of available energy to total energy, thus ranging from 1 (full battery) to 0 (empty battery). Each of the actions described above reduce the battery in the following amount (communication energy usage simplified from [43]):

- Sending a *N* byte message: *ByteTxPower x ByteTxTime x N.*
- Receiving a *N* byte message: *ByteRxPower x ByteTxTime x N.*
- Listening *t* seconds for messages: *t x ListenPower.* No sleep time was considered for this model.
- Sampling the gas concentration: *SamplePower x SampleTime.*

Base stations

There will be only one base station for this scenario, called the *Antenna*. It will gather the estimations the nodes send periodically and update its own estimation with the data, based on the same merging algorithm the nodes use. This estimation will be used to compute several performance indexes for the simulation.

Communication

Two communication channels are defined for this network: one handles node communication and the other is used for *Drone-Antenna* communication. Both have the following characteristics:

- The channels are wireless.
- The power loss is modeled by a simplified Friis transmission equation [44]. The loss factor in this model will be $A\,E\,\frac{1}{4\pi R^2}$, where *A* is the receiving surface, *E* is the efficiency of the receiving system and *R* is the distance between sender and receiver. The received power will be the transmission power times the loss factor.
- If the received power is less than a given threshold, the message is not received.
- Successful messages can collide if the receiving times overlap. In this case, a single message may be received if throughout its duration its power exceeds the sum of all other powers by at least a threshold given in decibels.

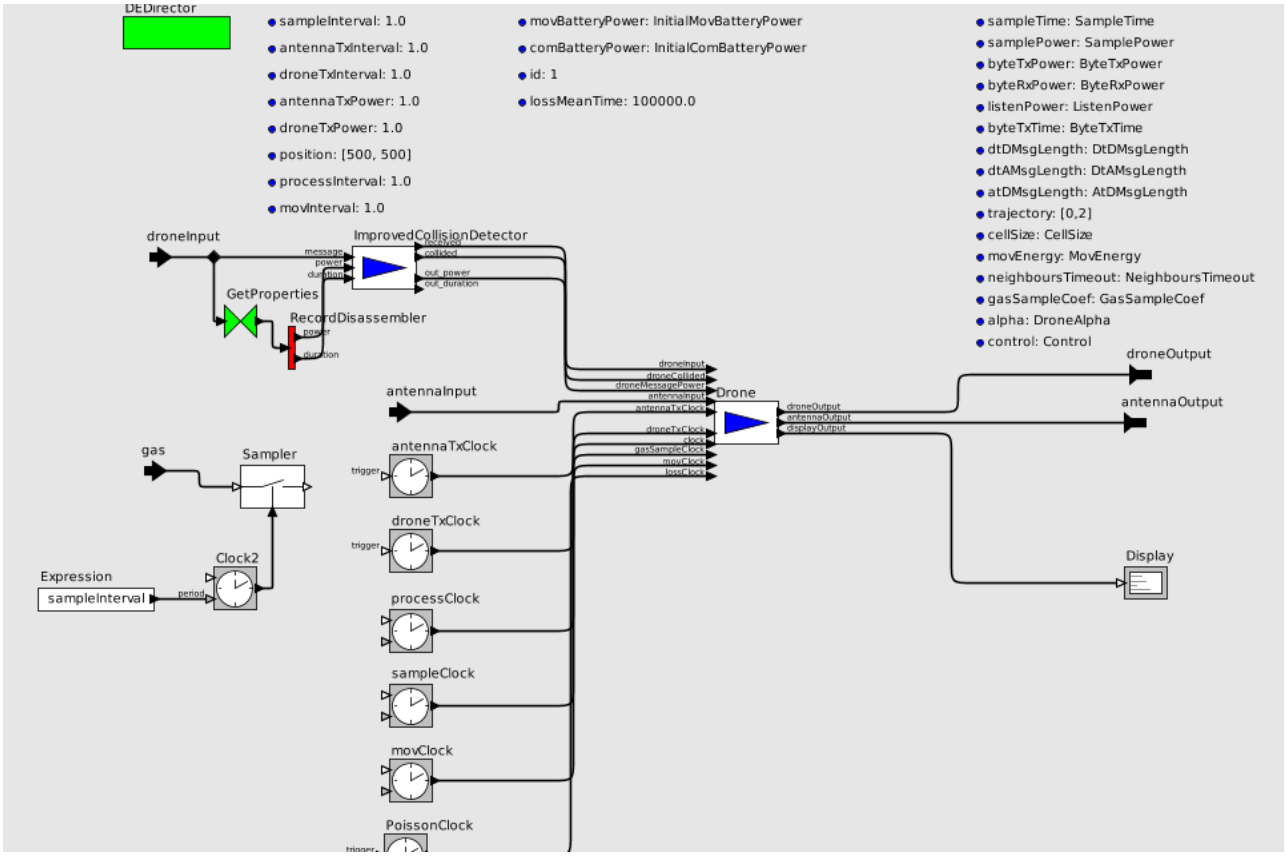

Figure 10: Actor model of a Drone

- Every message has a chance to be lost because of unidentified reasons, with a probability given by *LossProbability*.

Drone communication is restricted to one hop links, while *Drone-Antenna* communication is restricted to messages from *Drones* to the *Antenna*. As part of the communication, a very simple neighbor detection will be used: a node will add the sender of a received message to the neighbor list and will remove it if the decision element has been executed *NeighboursTimeout* times since the last received message.

4.2.2. Communication network

In this section we describe a second scenario which will be focused on the communication process of a network. This is the main testing scenario of the project and features a more elaborated decision element. Appendix B provides a full description of all the parameters of the model along with some minor details skipped in this section.

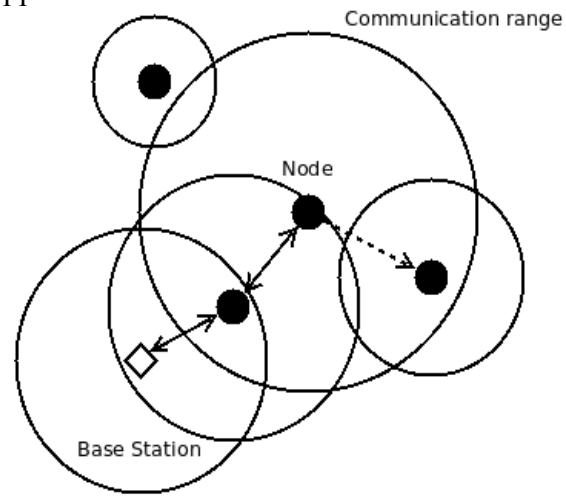

Figure 11: Overview of the communication network. Two nodes are connected if and only if they share a bidirectional link (depicted in solid lines with arrows). An unidirectional link (dashed line with an arrow) is not allowed for communication

Description

For this scenario, a number of nodes deployed in a region will form a communication network with a multi-hop topology that will try to send some data to a special node, the base station. A diagram of the system is depicted in Figure 11. The kind of information sent is left unspecified on purpose, as we will be only concerned about the communication process itself.

Three goals have been defined for the scenario:

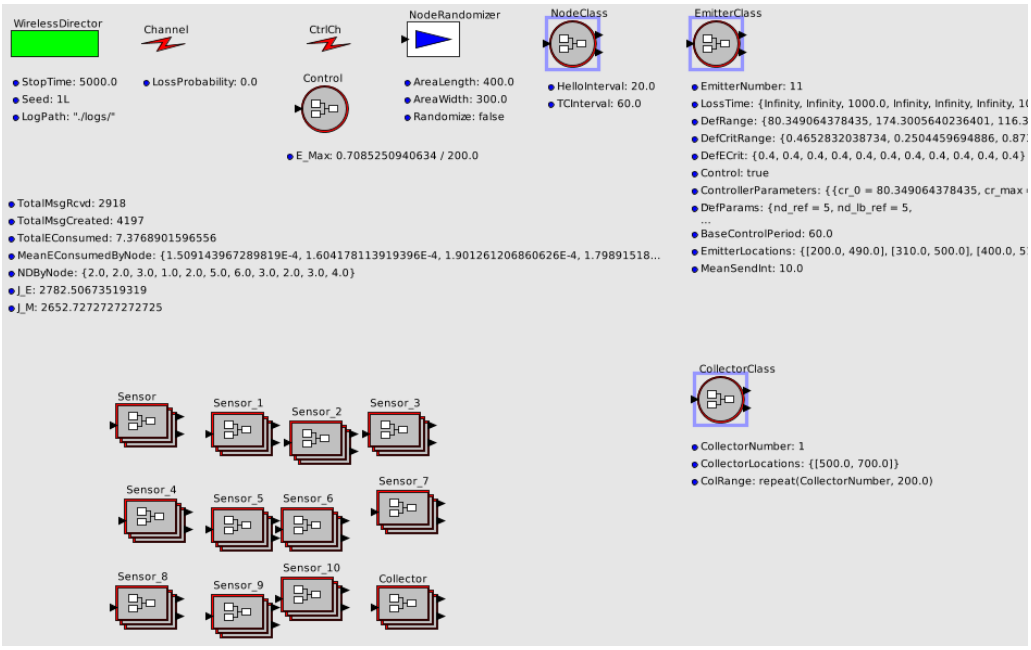

Figure 12: Model of the system in Ptolemy II / Visualsense. At the bottom left corner, a network topology is represented

- Maximize the number of nodes that are able to transmit data to the base station.
- Maximize the life time of the network. Alternatively, minimize node battery usage.
- Nodes should start saving battery when low.

Area of operations and environment

The network will be deployed in a region of *AreaWidth* by *AreaLength*. No grid will be defined for the region in this scenario and no other environment system will be modeled. The top level model of the scenario is shown in Figure 12.

Nodes

The only tasks assigned to the nodes in this scenario are to send messages to the base station and to relay messages sent from other nodes. The first task is performed with a mean interval time of *MeanSendInt*. The second task is fully described in the Communications subsection. A detail of the model of a node is depicted in Figure 13.

The battery model, shown as it is represented in Ptolemy II / Visualsense in Figure 14, considered for the nodes will be based on the assumption that the only relevant, battery consuming tasks are sending and receiving messages. The state of the battery is represented by the ratio of available energy to total energy, thus ranging from 1 (full battery) to 0 (empty battery). Each of the actions described above reduce the battery in the following amount (simplified from the gas concentration scenario):

- Sending a message: *MsgSendEnergy x _EmissionPower.*
- Receiving a message: *MsgRcvdEnergy.*

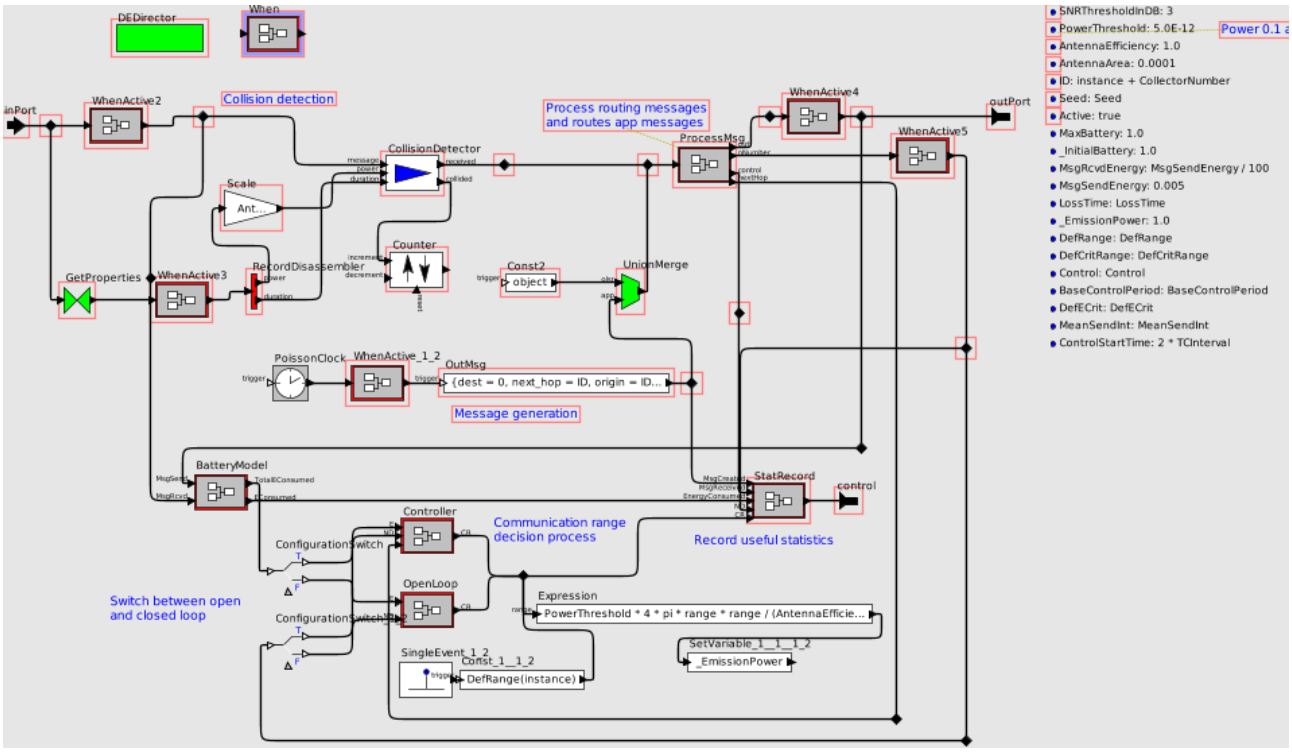

Figure 13: Detail of the actor model of a node

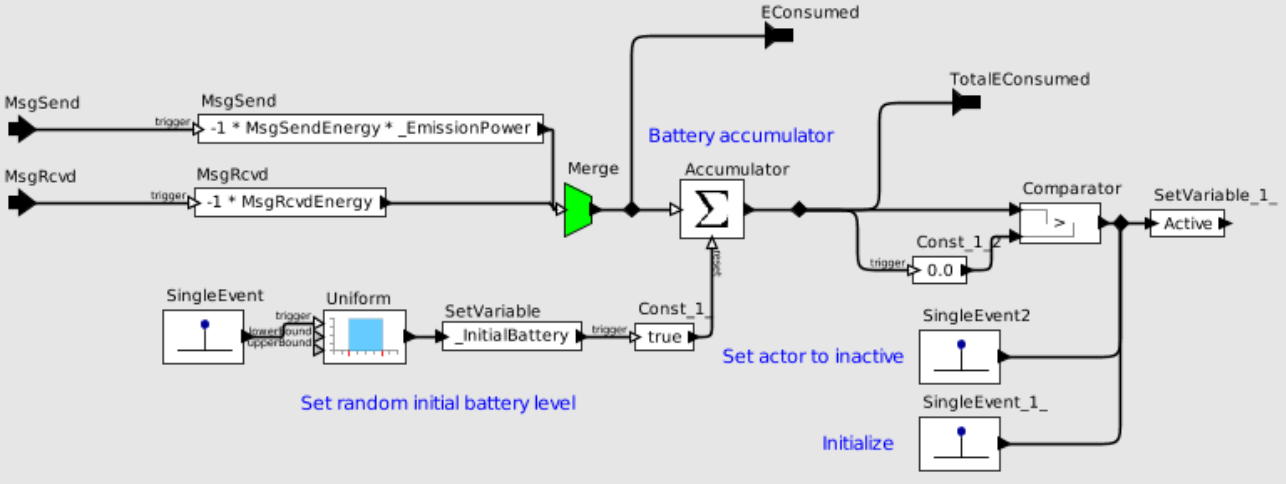

Figure 14: Detail of the actor model of the battery

Base stations

A silent, non battery consuming node will act as the base station and will be the destination of the messages sent from the other nodes. No other tasks besides counting the received messages will be performed by it.

Communications

Only one channel will be required to handle the communication process of the network. It will be a wireless channel with the same characteristics as the channels described in the Communications subsection of section 4.2. However, the topology of the net will be multi-hop, which means that a routing protocol will be needed.

Several routing protocols specifically designed for mobile ad hoc networks, such as AODV [45] or OLSR [46], have been proposed. These routing protocols are either on-demand or table-driven. On-demand routing protocols find a route by flooding the network with route request whenever a route to a destination is demanded. Conversely, table-driven protocols maintain a list of destinations and their routes by periodically distributing routing tables throughout the network. We will restrict our selection to table-driven protocols, as on-demand protocols are not able to detect all of the neighbors of a node, which we will need in our approach.

The routing protocol which will be used in this scenario is based on the table-driven OLSR protocol. This protocol has been selected because of its simplicity and because it is based on the well known OSPF protocol [47]. It uses *hello* and *topology control* (TC) messages to discover and then disseminate link state information throughout the mobile ad hoc network. Individual nodes use this topology information to compute next hop destinations for all nodes in the network using shortest hop forwarding paths.

4.3. Self-adaptive strategies

We proceed here to define the self-adaptive strategies used for each scenario. In the case of the gas propagation sensor network, we design a very simple strategy aimed to test the feedback loop architecture. On the other hand, the decision element for the communication network will be more elaborate, designed to maintain a desired level of connectivity while reducing the energy consumption of the network.

4.3.1. Gas propagation scenario

As previously discussed, the decision element in this scenario will be simple, given that its main purpose is to check the design and implementation of the architecture. The scenario will be modeled so that each node will execute a decision algorithm with some of the following inputs and outputs:

- Possible inputs: battery level, gas estimation, number of collisions detected, number of neighbors.
- Possible outputs: *Drone* transmission interval, *Antenna* transmission interval, gas sample interval, decision process execution interval, transmission power.

We will be using a single input, single output fuzzy controller, with the battery level as input and a multiplier to adjust an initial sample interval, *SampleInterval,* as output.

The design of the controller under the MAPE-K architecture will demonstrate how it is not always possible to clearly identify its four functions. Below it is described a possible design, depicted in Figure 15, while a complete description of a general fuzzy controller like the one we will be using is available in section 3.3.

- Monitoring: a single value will be computed in this phase: the difference between the battery level and the critical battery level, averaged with the previous difference. This is done in order to minimize the impact of an erroneous value being supplied to the controller.
- Analysis: in a fuzzy control system, the process of computing the membership functions of each input and output variable (or fuzzyfication) acts as the analysis phase of the MAPE-K architecture. In this particular instance, the fuzzyfication process is resumed in Table 3.

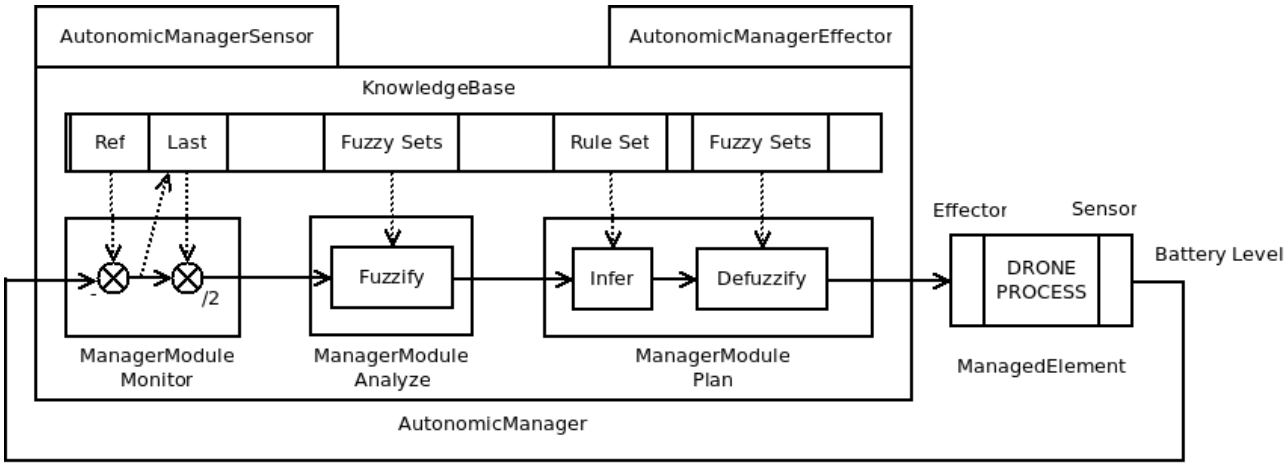

Figure 15: Block diagram representing the adaptive system system. Each major component is named after its base class according to the design of the feedback loop architecture described in section 4.1.2. Dotted arrows represent the use or storage (possibly after a transformation) of knowledge base values.

- Planing: the evaluation of the rules in Table 4 and defuzzyfication of the output variables using the choice of operations in Table 5 constitutes the planing phase. It should be noted that for a fuzzy controller it is not clear where the analysis phase ends and the planing phase starts, as the change request could be the output of the evaluation of the rules. We have chosen to split it into two phases to further test the correctness of the implementation, but the whole fuzzy control system will be considered a single analysis/planing phase for all subsequent decision element designs.
- Execution: this phase will be left empty, as the execution is a straightforward parameter change.

The output of the controller will be sent to the *Effector* of the *Drone,* which will obtain the final sample interval multiplying the controller action by the initial *SampleInterval.*

Variable	I/O	Range	Partition	Distribution
"EnergyError"	I	[-1 1]	"LL" (Low Level)	Trapezoidal [-2 -1 -0.1 0.1]
			"HL" (High Level)	Trapezoidal [-0.1 0.1 1 2]
"SampleFactor"	O	[0 5]	"NV" (Normal Value)	Constant [1]
			"SV" (Slow Value)	Constant [2]

Table 3: List of variables and fuzzy sets

Rule Name	Rule
RF1	IF (EnergyError IS LL) THEN (SampleFactor IS NV)
RF2	IF (EnergyError IS HL) THEN (SampleFactor IS SV)

Table 4: Rule set. All rules have weight 1

Operation	Function
And	Product
Implication	Product
Defuzzification	Weighted Average

Table 5: Operation choice

4.3.2. Communication network scenario

The main objective of the scenario is to maximize the number of nodes connected to the base station. As a full study of the possible approaches to this objective is out of the scope of this project, we will take an approach that suits the control theory methodology developed in this work, fully described in [48]. In this case, the connectivity objective is achieved by controlling the number of neighbors of the nodes in the network.

Thus, when also taking into account the other objectives of the scenario, the goal of the decision element in this scenario is twofold: on the one hand, it will be maintaining the desired connectivity level of the net, while on the other hand, it will reduce the energy usage when the battery level is below a defined threshold. In order to accomplish these objectives, the following input and output variables have been identified for a decision algorithm:

- Communication Range (*CR*): the maximum distance from the emitting node at which another node is able to receive its messages. This will be the only output of the decision algorithm. By changing this parameter of the node, the controller will be able to affect both the connectivity of the net (as the number of neighbors of the node will be affected) and the battery consumption (as the power needed to send a message is proportional to the communication range).

  This variable is constrained by a lower and an upper bound, representing the fact that the communication circuit of the node needs a minimum power to emit and allows up to a maximum power for the emission. Usually the communication power is not a continuous variable in a real system but a discrete variable, allowing only a few preset power values to be used. We have decided not to restrict the possible values of the variable too much, setting up a step parameter, Δcr_{min}, such that the variation of the communication range should be done in a multiple of this parameter.

  Finally, as previously discussed, the power loss over distance is known for this model. Thus, the equation relating the emission power and the communication range can be established as the following:

$$P_T = PA\,E\,\frac{1}{4\pi\,CR^2} \tag{11}$$

  Where P is the emission power, CR is the communication range, P_T is the power threshold (*PowerThreshold*) and A and E are the area and the efficiency of the antenna respectively (*AntennaArea* and *AntennaEfficiency*).

- Battery level (*E*): the ratio between available energy and total energy. This will be used as an input variable to identify the instant the battery level reaches the critical level. It is continuous and between 0 and 1.

- Node Degree (*ND*): the number of neighbors of the node, as defined in the previous section. This will be used as an input variable. The node degree is related to the connectivity level of the net in the following sense: as the communication network is multi-hop, more pathways between nodes reduce the chance of two nodes not being connected at some point due to some unpredictable problem, thus improving the connectivity. But the number of pathways between nodes is directly related to the node degree of each node (as discussed in [48]), making it an easier to compute input variable. It is constrained to be a non negative integer variable.
- Connected (*C*): a boolean value indicating the presence or absence of a path from the node to the base station.

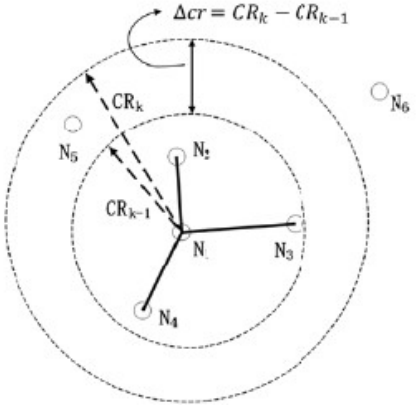

Figure 16: Visualization of the relation between the communication range and the node degree

The node degree is directly controlled by the value of the communication range, as we can see in Figure 16. If this relationship is known, the connectivity problem is easily solved: the desired connectivity is defined by the desired node degrees of each node and the communication range is set to the corresponding value. However, this is not usually the case. A number of factors, such as the influence of the communication range of other nodes in the node degree of a given node, or the unpredictable loss of a node, make the relationship between the two variables impossible to define.

The alternative is to design a control system that performs a self-adaptation of the communication range against the dynamic changes of the network. A basic feedback control loop will be responsible for the dynamic adjustment of the node degree to the desired reference value by changing the communication range accordingly. This control loop will be designed following the MAPE-k architecture and it is depicted in Figure 17 as the AutonomicManager 1. The functions performed by each module are defined as follows:

- Monitoring TM1: this module computes the deviation of the node degree from the reference:

$$e_{ND}(k) = ND_R(k) - ND(k) \tag{12}$$

  ND_R is the node degree reference, stored in the knowledge base. In order to avoid sign oscillations caused by a physical impediment to reach the exact reference node degree, for example when several neighbors are at the same distance from the node, a conditional activation of the next module is performed in this monitor. If $|e_{ND}|$ is less than a threshold named ξ_{ND}, the execution of this loop is aborted and no changes will be made to *CR*.

  As the node should always try to be connected to the base station, regardless of the value of e_{ND} the monitor will trigger the next module and set the error to ξ_{ND} if the node has no path to the base station. In Table 10, these two rules are stated more precisely, named Tol1 and Seek, respectively.

- Reasoning TR1: this module groups the functions assigned to the analysis and planning modules of the MAPE-k architecture. In this stage, the function of decision-making *FDM1* is applied to a normalized e_{ND} computed in the monitoring phase, and its output is denormalized before passing it to the next module. More precisely:

$$e_1(k) = k_{ND} e_{ND}(k) \tag{13}$$

$$\Delta u_1(k) = f_{DM1}(e_1(k)) \tag{14}$$

$$\Delta cr(k) = k_{cr} \Delta u_1(k) \tag{15}$$

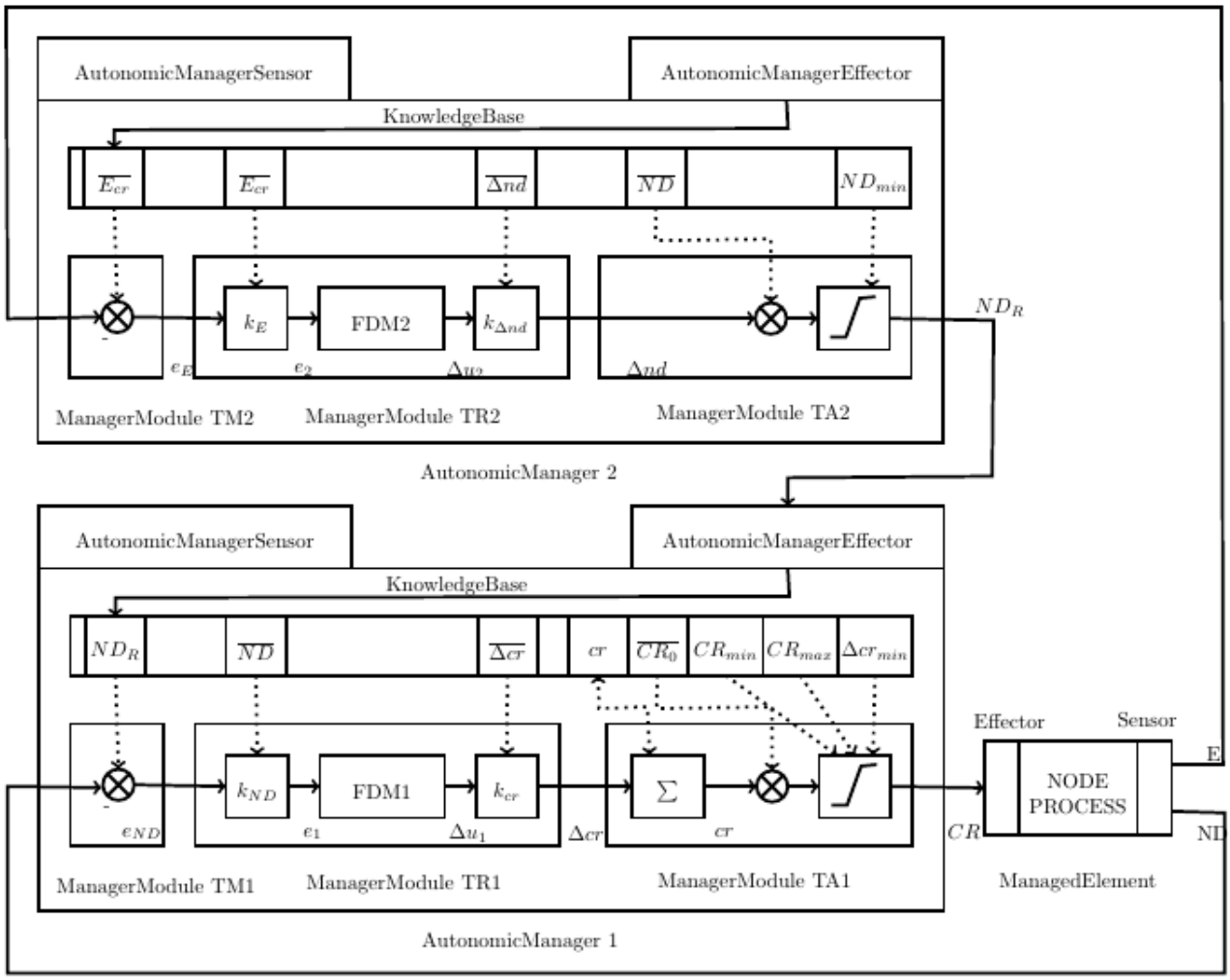

Figure 17: Controller modules and tasks. Each major component is named after its base class according to the design of the feedback loop architecture described in section 4.1.2. Dotted arrows represent the use or storage (possibly after a transformation) of knowledge base values.

In these equations, k_{ND} and k_{cr} are normalization factors that can be derived from the nominal parameters of the controller $\overline{ND}$, which represents the desired value of the node degree when the battery is exactly at the critical level, and $\overline{\Delta cr}$, which represents the communication range variation rate, as follows:

$$k_{ND} = 1/\overline{ND}\,,\; k_{cr} = \overline{\Delta cr} \tag{16}$$

The function of decision-making could be as simple as the sign of e_1 and zero if the input is zero, but then it would not take into account the inherent uncertainty of the input variable as a highly dynamic and unpredictable system. In order to cope with this, we will use a fuzzy control system, whose fuzzy partitions of the input space address precisely the uncertainty problem. In this case, three fuzzy sets will be used: negative values (*NV*), zero values (*ZV*) and positive values (*PV*). The output space will be partitioned as well in three fuzzy sets: negative change (*NC*), zero change (*ZC*) and positive change (*PC*). The membership functions defined for each fuzzy set are described in Table 6. The trapezoidal function has been selected over other well known functions in the literature (such as Gaussian, triangular or generalized bell [49]) due to its expressiveness and simplicity.

It is worth mentioning that the partition of the input space is not symmetric. This is due the difference in reaching the reference node degree while increasing or decreasing the

```
Instance config = getConfigFromKB().getLastInstance();
Instance state = getDataFromKB().getLastInstance();
Instance out = new Instance(data);
Instance state_new = new Instance(getDataFromKB());

double cr = state.getDouble("cr");
double CR_0 = config.getDouble("CR_0");
double CR_max = config.getDouble("CR_max");
double CR_min = config.getDouble("CR_min");
double dcr_min = config.getDouble("dcr_min");
double dcr = instance.getDouble("dcr");

cr += dcr;

if (CR_0 + cr > CR_max) {
    cr = CR_max - CR_0;
} else if (CR_0 + cr < CR_min) {
    cr = CR_min - CR_0;
}

state_new.setValue("cr", cr);

removeInstanceFromKnowledge(dataIndex, 0);
addInstanceToKnowledge(dataIndex, state_new);

double CR = CR_0 + ((int) cr / dcr_min) * dcr_min;
out.setValue("CR", CR);
sendInstanceToObservers(out);
```

Listing 1: Actuator TA1 processing function

communication range. The latter is more likely to be a waste of power, as the same node degree could be reached with a lower value of the communication range. Even though this solution does not solve the problem of finding the best value for a given node degree, it preserves the simplicity of the design while improving the behavior of the controller when facing this issue.

The remaining elements of the fuzzy controller, namely the rule base and the operation choice are defined in Table 7 and Table 8, respectively. For a detailed description of a general fuzzy control system, see section 3.3. The graph of the f_{FDM1} function is depicted on the left side of Figure 18.

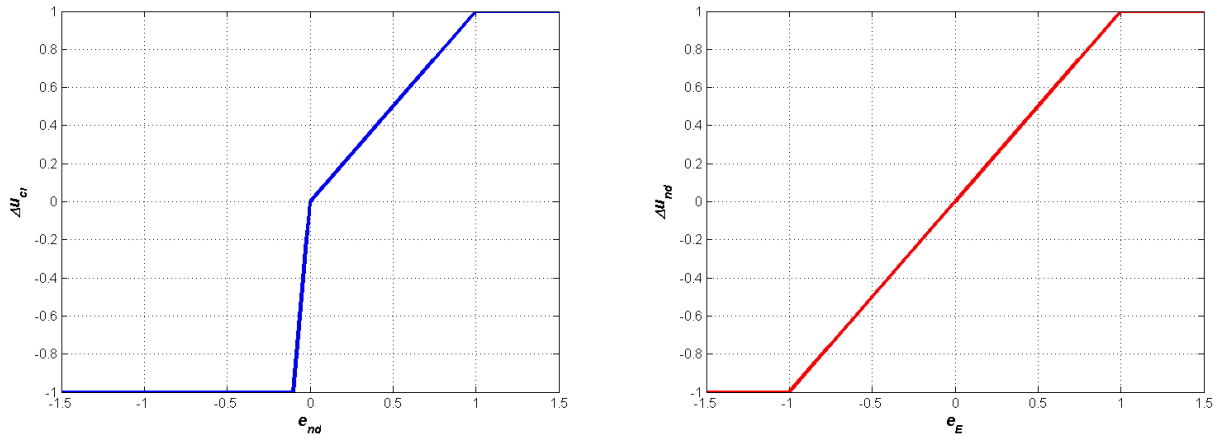

Figure 18: Graphs of the function described by each fuzzy inference system. On the left side, f_{FDM1} is depicted, while on the right hand side, f_{FDM2} is shown

- Actuator TA1: this module holds the information of the previous increments (stored in the knowledge base) and provides the process (the node communication system) with the adjusted communication range. In particular, the output of the module is defined by the following equations:

$$cr(k)=cr(k-1)+\Delta cr(k) \tag{17}$$

$$CR^*(k)=\left[\frac{cr(k)+\overline{CR_0}}{\Delta cr_{min}}\right]\Delta cr_{min}$$

$$CR(k)=\begin{cases} CR_{min}, & CR^*(k)<CR_{min} \\ CR^*(k), & CR_{min}\le CR^*(k)\le CR_{max} \\ CR_{max}, & CR^*(k)>CR_{max} \end{cases} \tag{18}$$

Three nominal parameters are used by this module: $\overline{CR_0}$ represents the initial value of the communication range, CR_{min} represents its minimum allowed value and CR_{max} represents its maximum allowed value. CR^* is defined to be a multiple of the minimum variation range. An example code of this module is listed in Listing 1.

Variable	I/O	Range	Partition	Distribution
"e"	I	[-2 2]	"NV"	Trapezoidal [-4 -2 -0.1 0]
			"ZV"	Trapezoidal [-0.1 0 0 1]
			"PV"	Trapezoidal [0 1 2 4]
"du"	O	[-2 2]	"NC"	Constant [-1]
			"ZC"	Constant [0]
			"PC"	Constant [1]

Table 6: List of variables and fuzzy sets of FDM1

Rule Name	Rule
R1	IF (e IS NV) THEN (du IS NC)
R2	IF (e IS ZV) THEN (du IS ZC)
R3	IF (e IS PV) THEN (du IS PC)

Table 7: Rule base of FDM1 and FDM2. All rules have weight 1

Operation	Function
And	Product
Implication	Product
Defuzzification	Weighted Average

Table 8: Operation choice of FDM1 and FDM2

Another control loop will be used to accomplish the critical battery level objective. In this case, the decision element will change the desired connectivity of the network by changing the node degree reference of the previous controller. In Figure 17, the control loop is depicted as the AutonomicManager 2, as well as its relationship with the primary control loop. The controller functions are similar to the primary one:

- Monitoring TM2: in this case the monitoring module computes the deviation of the battery level from the critical level, $\overline{E}_{cr}$:

$$e_E(k) = \overline{E_{cr}} - E(k) \tag{19}$$

  In a similar manner as the monitor 1, we refrain from further evaluating the controller if the deviation is, in this case, more than a threshold named ξ_{ND} (ie, we only evaluate this loop if we are in the transition phase). Contrasting with the module Monitoring 1, the purpose of this rule is not to avoid oscillations but to reduce the number of full evaluations of the controller. This rule is called Tol2 in Table 10.

- Reasoner TR2: this module follows the same design as Reasoner 1, although in this case the output will not be used as accumulating variations but as an absolute variation:

$$e_2(k) = k_E e_E(k) \tag{20}$$

$$\Delta u_2(k) = f_{DM2}(e_2(k)) \tag{21}$$

$$\Delta nd(k) = k_{\Delta nd} \Delta u_2(k) \tag{22}$$

  For this reasoner, the nominal parameters are $\overline{E}_{cr}$ and $\overline{\Delta nd}$, which represents the node degree variation rate. Then the normalization constants are:

$$k_E = 2/\overline{E_{cr}}, k_{\Delta nd} = \overline{\Delta nd} \tag{23}$$

  The function of decision-making, FDM2, selected for this reasoner is again a fuzzy control system. In this case, the choice is made as a design simplification as the characteristics of this variable were not fully understood before the control system was planned. The design of the fuzzy controller is the same as FDM1, the only change being the membership functions of the input fuzzy sets, which in this case make the partition symmetric. See Table 9 for the description of the variables of FDM2. The rule base and the choice of operations are the same as the ones of FDM2, described in Table 7 and Table 8 respectively. Equation (21) is plotted on the right side of Figure 18.

- Actuator TA2: for this control loop, the actuator does not need to accumulate previous changes. Thus, the output of the actuator is the sum of the variation to the desired value of the node degree when the battery is exactly at the critical level, $\overline{ND}$, respecting a lower bound set by design of the network, ND_{min}:

$$ND_R^*(k) = \overline{ND} + \Delta nd(k)$$
$$ND_R(k) = \begin{cases} ND_{min}, & ND_R^*(k) < ND_{min} \\ ND_R^*(k), & ND_R^*(k) \geq ND_{min} \end{cases} \quad (24)$$

Variable	I/O	Range	Partition	Distribution
"e"	I	[-2 2]	"NV"	Trapezoidal [-4 -2 -1 0]
			"ZV"	Trapezoidal [-1 0 0 1]
			"PV"	Trapezoidal [0 1 2 4]
"du"	O	[-2 2]	"NC"	Constant [-1]
			"ZC"	Constant [0]
			"PC"	Constant [1]

Table 9: List of variables and fuzzy sets of FDM2

There is a classic issue in control systems that has not been addressed in the design of the controller which we will discuss here: the saturation of the output to the maximum allowed value [50]. In the modeled network, the saturation of the controller is commonly reached in the last phase of the network life cycle, when several nodes are out of functioning due to battery depletion. However, this is not the only scenario where saturation is reached, as random losses of nodes or bad configurations of the controller could lead to the same situation. In these scenarios, the node degree of a node is less than the reference even when the controller increases the communication ranges until it saturates. This situation is not desirable because the node is consuming unnecessary energy.

In order to deal with this issue, we first note that he system is saturated when $e_{ND}(k) > 0$ (meaning the node degree is less than the reference) and $CR_f(k) \geq CR_{max}$ where $CR_f(k)$ is the average of CR corresponding to times k and $k-1$. The first condition is driving the controller to endlessly increase the communication range, so we start by exceptionally decreasing the reference to the current value of the node degree following the next updating rule:

$$ND_R^*(k) = ND_R(k-1) - \Delta ND_{min}$$
$$ND_R(k) = \begin{cases} ND_{min}, & ND_R^*(k) < ND_{min} \\ ND_R^*(k), & ND_R^*(k) \geq ND_{min} \end{cases} \quad (25)$$

Where ΔND_{min} is the minimum step value between two consecutive discrete values of ND. After this, the controller will no longer try to increase the communication range, but it will not reduce it either, wasting energy. Thus, we force the controller to perform a decreasing step by setting $e_{ND}(k) = -\xi_{ND}$ (allowing the execution of the reasoner by the rules of Monitoring 1) and proceeding with the evaluation of the control loop. We describe the rule, named Sat, in Table 10. The same discussion in Reasoner 1 about the efficiency of the solution applies here, but we consider it to be enough for the purpose of this project.

It should be noted that the solution described is, in fact, a complete control loop that should ideally be designed under the MAPE-k architecture. However, this would add a lot of complexity to the design, as it would be interleaved with both the primary and the secondary control loops (it would need inputs from the primary loop while its output would be the same as the secondary loop). In

addition, the typical scenario where the saturation problem arises is outside the stable phase of the network, ie, between the end of the initialization and when nodes start to run out of battery. Designing a controller with a correct behavior in this region is a non trivial problem in control theory (see [50]) lying beyond the scope of this project, making the simple ad hoc solution valid for our purposes.

Rule	Condition	Action
Sat	$e_{ND}(k)>0 \wedge CR_f \geq CR_{max}$	Apply Equation (25) and set $e_{ND}(k)=-\xi_{ND}$
Seek	$e_{ND}(k)<0 \wedge \neg C$	Set $e_{ND}(k)=\xi_{ND}$
Tol1	$\lvert e_{ND}(k)\rvert \geq \xi_{ND} \vee ND < ND_{min} \vee \neg C$	Continue with TR1
Tol2	$\lvert e_E(k)\rvert \leq \xi_E$	Continue with TR2

Table 10: Summary of monitor decision rules

Optimization of decision element parameters

The last step of the control system design process is the adjustment of the parameters. The objective of this phase is to find the best configuration of the designed controller according to some selected performance indexes. Two fundamentally different sets of parameters can be defined, each related to a specific objective when targeted by the adjustment process:

- The nominal values of system variables: system variables are considered those that allow characterizing the functionality of the system regardless of the use of a controller, such as communication range, node degree or critical energy level. The nominal values for those variables are the values tuned by the system designer and selected under a certain criteria to provide the desired behavior of the system (for example, low energy consumption or high connectivity). The adjustment procedure is performed over the open loop system (i.e., without feedback control) and the performance indexes are defined to quantify the behavior of the controlled system.
- The feedback control algorithm parameters: those strictly related to the behavior of the controller, such as change rates or tolerances. In this case, the procedure uses the closed loop system configured with the optimized nominal values of system variables and the performance indexes measure the behavior of the controller. Common indexes are related to the evolution of the error, such as the sum of squared errors.

In the remaining of this section we thoroughly describe the methodology for the adjustment of the nominal values of system variables, which can be resumed in the following steps:

1. Model the desired behavior of the system as an open loop system parameterized by a set of system variables. This will be the reference the controller tries to follow.
2. Define a performance index that measures how well the system accomplishes the objectives set for it.
3. Execute an optimization algorithm over the parameter set. The model developed in step 1 will be used to obtain the performance index associated with each configuration.
4. Set the values of the controller parameters derived from the nominal values of the system variables.

The open loop design for this scenario has already been discussed: a desired connectivity level is defined (in terms of the communication range of each node) and is switched to another connectivity level after the node reaches a critical battery level. More precisely:

- At time 0, the battery of the node i is full and the initial communication range is set to $\overline{CR_0}^i$.
- At some time t_{cr}, the battery level of the node i reaches the critical value $\overline{E}_{cr}^i$ and the communication range is set to $\overline{CR}_F^i = \overline{cr}_F^i \cdot \overline{CR}_0^i$, $0 < \overline{cr}_F^i < 1$, which will reduce the energy consumption of the node until the depletion of the battery.

A visual representation of this behavior is depicted in Figure 19. The selected system variables for each node i are $\overline{CR_0}^i$, $\overline{E}_{cr}^i$ and $\overline{cr}_F^i$. The union of the system variables of every node will be the target of the optimization algorithm. In order to define the associated performance index, we must recall that the goals of the system are to maintain a good connectivity level (related to the paths between nodes and the base station) and to use a low amount of energy. The first objective can be measured by an index related to the number of messages received by the base station, as each one can be thought of as the evidence of a path from some node at some instant. The second objective is easily mapped to the rate of energy usage of the network. Then, the performance index is defined as:

$$\begin{aligned} J_E &= \alpha_E \sum_{i=1}^{n} \frac{E_i}{T_i} \\ J_M &= \alpha_M M_{BS}^T \\ J &= w_E J_E - w_M J_M \end{aligned} \tag{26}$$

Where E_i is the total energy consumed by the node i, T_i is the time the node has been active (until battery depletion), M^T_{BS} is the total number of messages received by the base station and n is the number of nodes. The coefficients α_E and α_E normalize each cost index in order to make its units agree and its order comparable. In this case, the unit selected is seconds and the factors are defined as follows:

$$\begin{aligned} \alpha_E &= \frac{\dot{E}_{max}}{T_{max}} \\ \alpha_M &= n \nu_M \end{aligned} \tag{27}$$

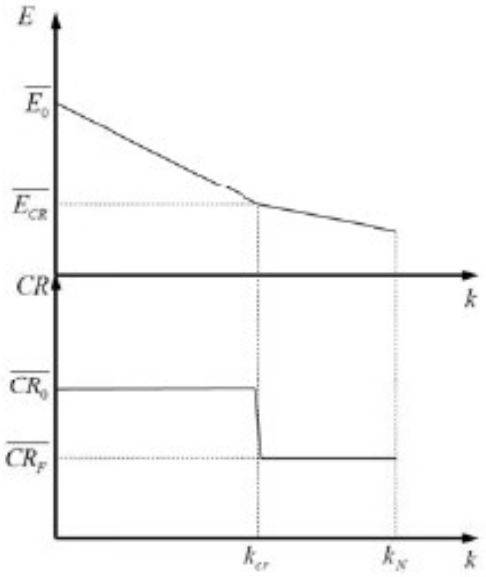

Figure 19: Graphical representation of the desired theoretical behavior of the energy level and the communication range for each node

.

Here, E_{max} and T_{max} are the maximum rate of energy consumption of the whole net and its maximum life time respectively, while ν_M is the expected value of the message emission frequency of the nodes.

The performance index defined is meant to be minimized by the optimization algorithm. There are several good candidate algorithms which can be employed for this step. According to previous experiences [51] cross entropy algorithm [26] is simple and easy to implement. It also has a good performance and low computational resources when solving multidimensional problems [52]. Section 3.2 provides a brief description of the algorithm.

After the optimization process is executed, a number of parameters of the control system can be set from the optimized nominal values. First, the corresponding node degrees, $\overline{ND_0}^i$ and $\overline{ND_F}^i$, for the communication ranges, $\overline{CR_0}^i$ and $\overline{CR_F}^i$, should be obtained from the model. Then, the following parameters can be set:

$$\overline{ND_i} = \frac{\overline{ND_0^i} + \overline{ND_F^i}}{2}$$
$$\overline{\Delta nd_i} = \frac{\overline{ND_0^i} - \overline{ND_F^i}}{2} \tag{28}$$

The critical energy level nominal value, $\overline{E_{cr}}^i$, does not need any modifications before being used as a parameter.

Finally, the remaining parameters of the control system should be adjusted. If the parameters are to be optimized, a similar methodology to the one explained above should be used. Instead of the open loop system, the closed loop system with the optimized nominal values should be used, and instead of a cost index related to the performance of the system against its goals, a cost index related to the behavior of the control system should be defined.

However, we have decided to manually tune the remaining parameters because of the low impact this step would have on the objectives of this project.

5. Test and results

In this section we execute several simulations of the scenarios described in section 4, using the Ptolemy II / Visualsense models developed. Our main focus will be measuring the self-adapting capabilities of the system, which we will do by comparing the performance of the system when the decision element is on and off. The performance of the system will be quantified using performance indexes suitable for each case.

In subsection 5.1 we conduct a shallow study on the gas propagation scenario as a proof of concept of the methodology followed in this project. In subsection 5.2 we perform several tests on the communication network scenario. We start by analyzing the automatic optimization process of the system values using the cross-entropy method. Then, we assess the performance of the self-adaptive strategy, as well as the impact of its parameters.

5.1. Gas propagation scenario

A single test will be performed on this scenario, comparing the performance of the network with and without the execution of the decision element. The objective of this test is to serve as a proof of concept of the kind of tests that we will be performing on the next scenario. Specifically, we want to ensure the simulating capabilities of the model built in Ptolemy II in section 4.2.1, as well as the implementation of the feedback loop architecture designed in section 4.1.

We first start by setting the parameters of the scenario as shown in Table 11. Refer to Appendix A for a full description of each parameter. The position of the network drones is shown in Figure 9.

Parameter	Value
AreaWidth	600 m
AreaLength	600 m
GasAlpha	-12.0/800.0 s^{-1}
CellSize	60 m
Wind	[-15.0/800.0, 1.0/800.0, 2.0/800.0, 7.0/800.0] s^{-1}
Source	10 by 10 zero matrix with (0,2) entry set to 70.0 *µg/(s cm³)*
ChannelRange	Infinity
PropagationSpeed	Infinity
LossProbability	0.0
ByteTxTime	1.0 s
DtDMsgLength	1
DtAMsgLength	1

Parameter	Value
AntennaReceptionThreshold	1.0E-7 W
AntennaRecoveryThreshold	3 dB
AntennaLocation	[-200, 700]
DroneAlpha	-12.0/800.0 s^{-1}
GasSampleCoef	[-15.0/800.0, 1.0/800.0, 2.0/800.0, 7.0/800.0] s^{-1}
NeighboursTimeout	10
InitialComBatteryPower	1.0
SampleTime	1.0 s
SamplePower	0.01
ByteTxPower	0.0001
ByteRxPower	0.0001
ListenPower	0.0

Table 11: Parameters of the simulation for the Gas Propagation scenario

Then, we execute the simulation of this network configuration two times: once without the control algorithm and a second time with the control algorithm active on each drone. The performance is based on the goals defined for the scenario in the design section, ie, maximizing the estimation

accuracy and the lifetime of the network. We define, for a gas concentration arranged in a matrix G (with the same coordinates as described in section 4.2.1) and an estimation matrix $\hat{G}$, the following performance index:

$$Relative\ Error = \frac{\|G - \hat{G}\|_\infty}{\|G\|_\infty} \tag{29}$$

The results show an increase on the lifetime of the network of a 40%, while only increasing the relative error of the estimation in 0.046 in average. Figure 20 depicts the evolution of the estimation error over time, showing the increase in lifetime and the slight loss in estimation accuracy. The behavior of the control system can be seen in Figure 21 and Figure 22, which show the expected decrease in the sample interval as battery runs out and the decrease in battery consumption rate respectively.

We will not be making further observations of the results here, nor will we execute the simulation with other sets of parameters. The main objective of learning and testing the capabilities of the Ptolemy II software regarding WSN simulation, and testing the implementation of the MAPE-K architecture has been achieved, although the resulting model is too complex to be approached within the scope of this project.

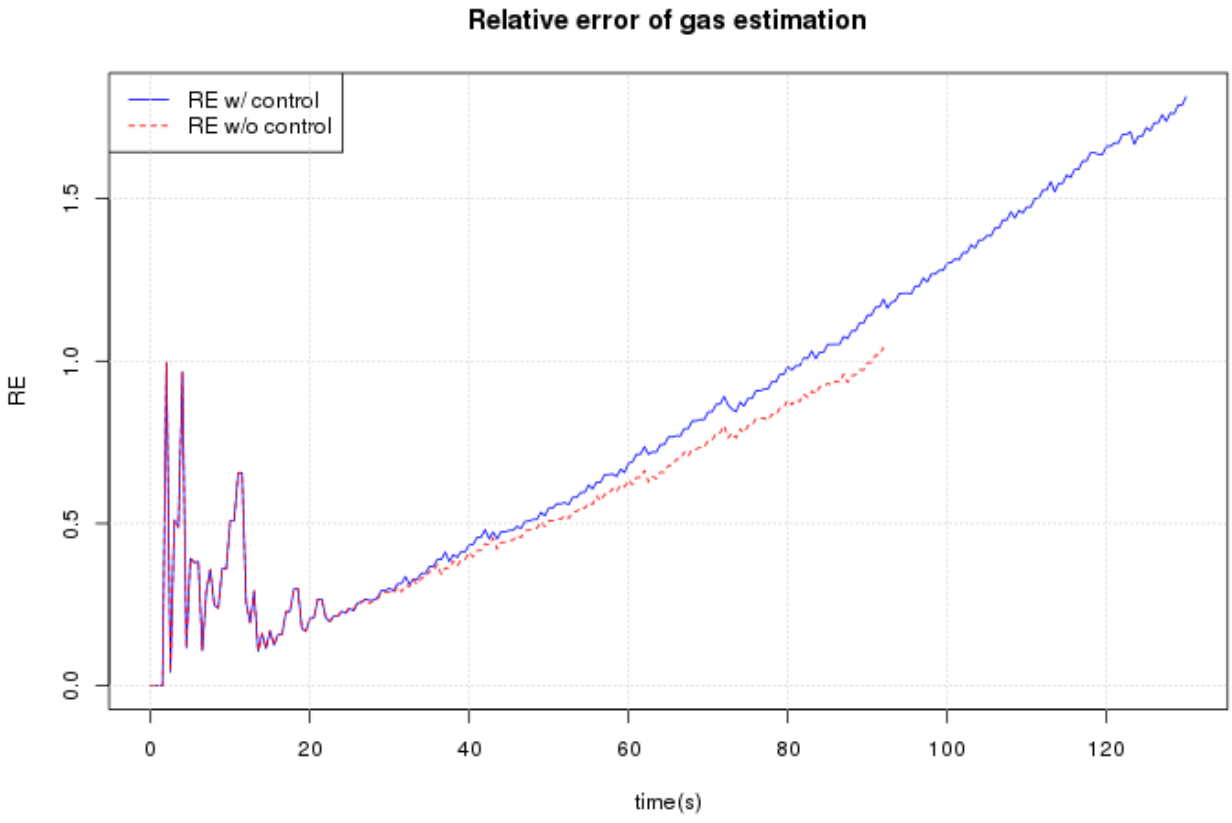

Figure 20: Relative error of gas estimation over time. Graph stops when all drones run out of battery

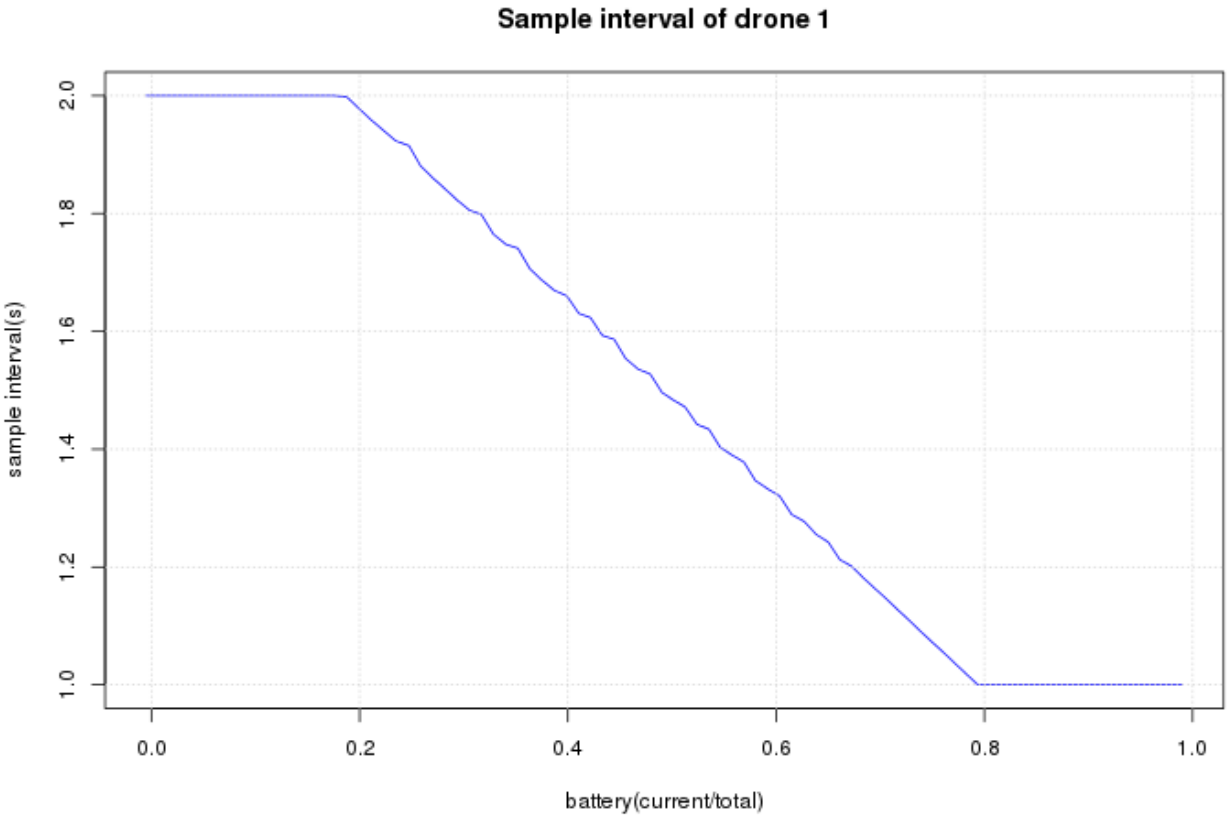

Figure 21: Sample interval against battery level for drone 1

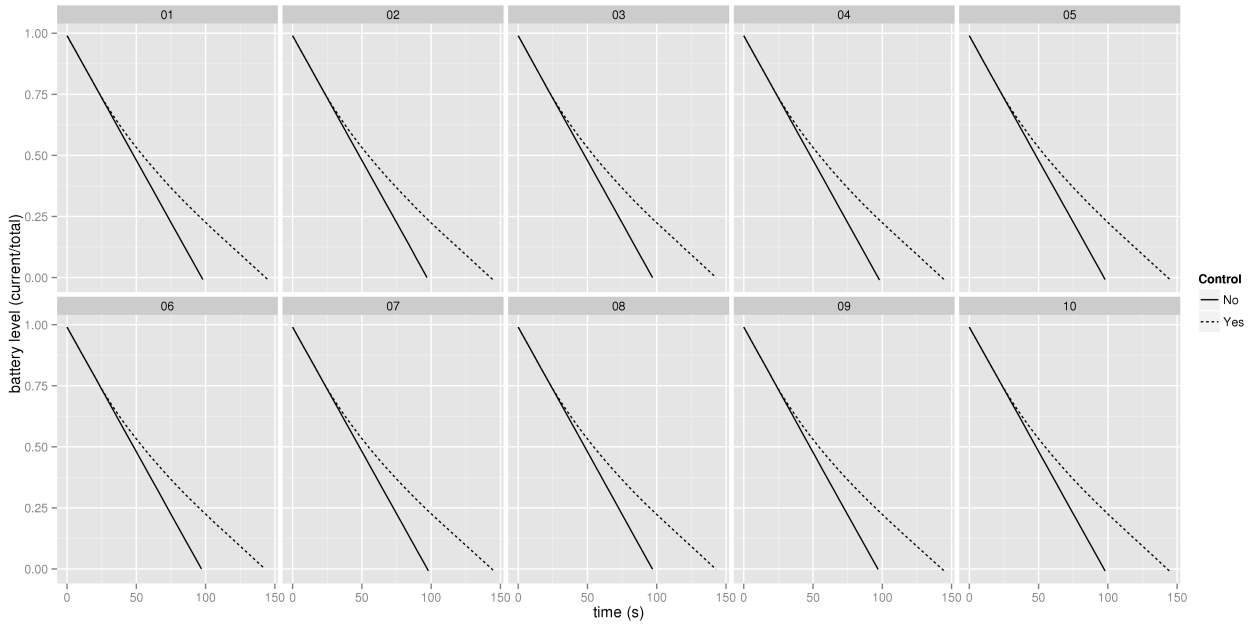

Figure 22: Battery level over time for each node

5.2. Communication network scenario

In this scenario we want to test the complete methodology that has been proposed throughout this work. We first start by analyzing the performance of the optimization algorithm for the adjustment of the parameters of the control system. Then, we simulate the model and check the behavior of the proposed decision element when the network is under unexpected conditions.

5.2.1. Parameter adjustment process

Once the model of the scenario and the control system have been designed and implemented, the next step of the control engineering methodology is to adjust the parameters of the controller. The process has been covered in detail in the corresponding subsection of section 4.2.2 and we present here an analysis of the results.

In order to put the adjustment process into use, a complete description of the scenario is needed. Table 12 contains the values of the parameters of the model used. Not included in the table are the positions of the nodes and the base station, which form a 4 by 3 grid pattern where the base station is at the corner, as depicted in Figure 12. The positions are slightly disturbed from a perfect grid distribution so that there is more variety in the external conditions of the nodes. A full explanation of the parameters can be found in Appendix B. Finally, we have determined empirically a simulation time of 5000 seconds, which is around the time the battery is depleted in all nodes for most configurations.

Parameter	Value
Seed	1
EmitterNumber	11
BaseControlPeriod	60.0 s
MeanSendInt	10.0 s
CollectorNumber	1
ColRange	200.0 m
LossProbability	0.0
AreaLength	400.0 m
AreaWidth	300.0 m
HelloInterval	20.0 s

Parameter	Value
TCInterval	60.0 s
Δcr_{min}	5 m
SNRThresholdInDB	3 dB
PowerThreshold	5.0E-12
AntennaEfficiency	1.0
AntennaArea	0.0001 m^2
MsgSendEnergy	0.0005
MsgRcvdEnergy	MsgSendEnergy / 100
ControlStartTime	2 * TCInterval

Table 12: Parameters of the simulation for the Communication Network scenario

Regarding the performance index defined in Equation (26) and (27), a few parameters need to be set. Specifically, E_{max} was determined empirically by simulating the network over a short period of time (200 seconds) while every node was transmitting at the maximum range (250 meters); T_{max} was set to the simulation time, 5000 seconds; ν_M was set to be the inverse of MeanSendInt; and the weight coefficients were both set to 0.5, meaning the energy part of the index was valued as much as the connectivity part.

Finally, the choice of the parameters of the cross entropy algorithm is detailed in Table 13. Even though the critical energy level, $\overline{E}_{cr}$, was intended to be one of the targets of the optimization, we have decided against it given the high number of variables to be optimized, which didn't allow for an acceptable convergence level in the number of iterations we could execute. Instead, we will be leaving it as a parameter set by the manager of the network, who will typically choose a value in

Parameter	Value
N	100
ρ	0.9
α	0.8
β	0.8
q	7
Initial $\overline{CR_0}$	150.0 m
Initial $\overline{cr_F}$	0.5

Parameter	Value
Initial standard deviation	0.5 times the initial mean
Minimum $\overline{CR_0}$	50.0 m
Maximum $\overline{CR_0}$	250.0 m
Minimum $\overline{cr_F}$	0.0
Maximum $\overline{cr_F}$	1.0

Table 13: Parameter choice for the cross entropy method. Initial values and constraints are the same for every node

relation with the time needed to retrieve a node and charge its battery. The value used in our test will be 0.4 for all nodes. With this in mind, the dimension of the optimization problem is $2x$, with x the number of nodes, and for each iteration of the algorithm, N samples of $2x$ parameters will be generated and simulated in ptolemy II.

Node	01	02	03	04	05	06	07	08	09	10	11
$\overline{\boldsymbol{CR_0}}$	80.35	174.31	116.36	137.71	79.91	233.73	163.13	130.59	130.65	125.84	175.29
$\overline{\boldsymbol{cr_F}}$	0.47	0.25	0.87	0.82	0.037	0.73	0.91	0.89	0.99	0.94	0.64

Table 14: Output of the cross-entropy method

The ending criterion we have used is a fixed number of iterations. The choice was made given the fact that, on a computer mounting an Intel® Core™ i7-3630QM 2.4GHz and 8GB of DDR3 memory running Linux, each sample takes between 50 and 80 seconds, which means between 80 and 130 minutes to complete each iteration. As the convergence for a problem with so many optimization targets is slow, we couldn't afford to use a convergence criterion. Instead, we set the number of iterations to 30 in order to restrict the execution time of the algorithm to less than 3 days. The best value found after 30 iterations is shown in Table 14.

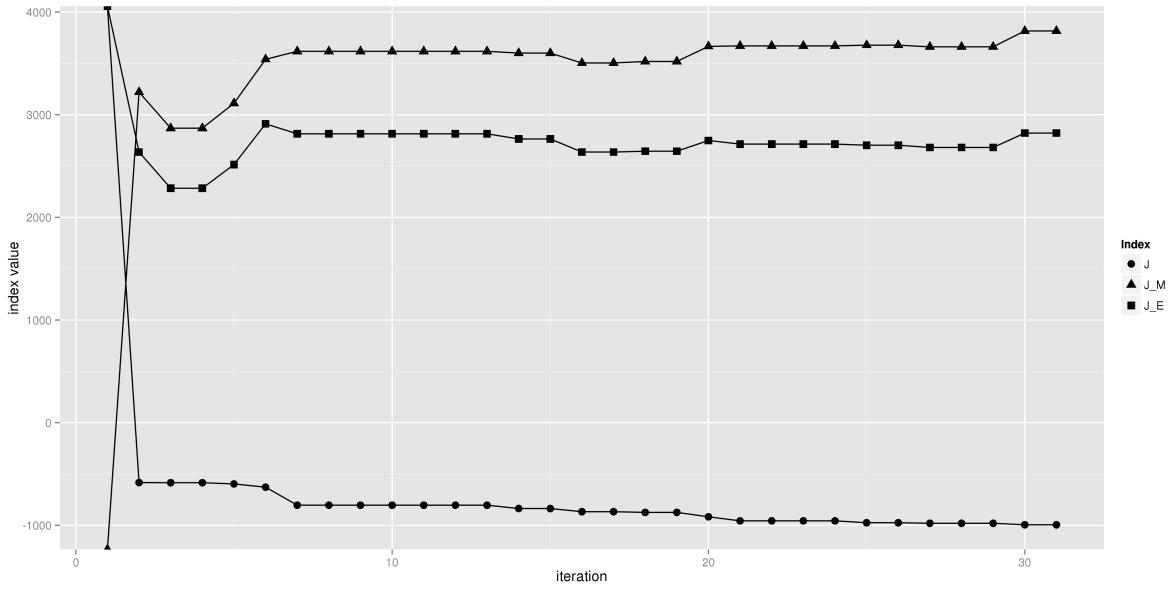

Figure 23: Best performance index found over iteration. Components of the index are also represented

In Figure 23 the evolution of the performance index over the iterations of the algorithm is represented. The figure shows the expected decreasing behavior of the index, approaching values around -1000. The same figure depicts the energy and connectivity components of the index as defined in Equation (26). It is worth observing that both components do not evolve in a regular way, which would be the ideal situation of being able to keep increasing the connectivity level of the network while reducing the energy consumption. This is only achieved in a few iterations, while in general one comes at the cost of the other. However, it is remarkable that it is possible to improve the connectivity without increasing the energy cost.

Figure 24 and Figure 25 show the mean and the standard deviation for both parameters of a selection of nodes over the algorithm iteration, respectively. Convergence in $\overline{CR_0}$ is acceptable, reducing the standard deviation to around a 5% of the initial value (evolution of the deviation is similar for other nodes). However, the evolution of the deviation for the $\overline{cr_F}$ parameter is not as good for some of the nodes, for example in nodes 1 and 5, only achieving a reduction of about a 50% of the initial value. This could be the result of a low dependency of the performance index to the parameter, resulting in changes in other parameters shadowing the contribution of the low impact ones. Another possible explanation could be the existence of multiple optimum.

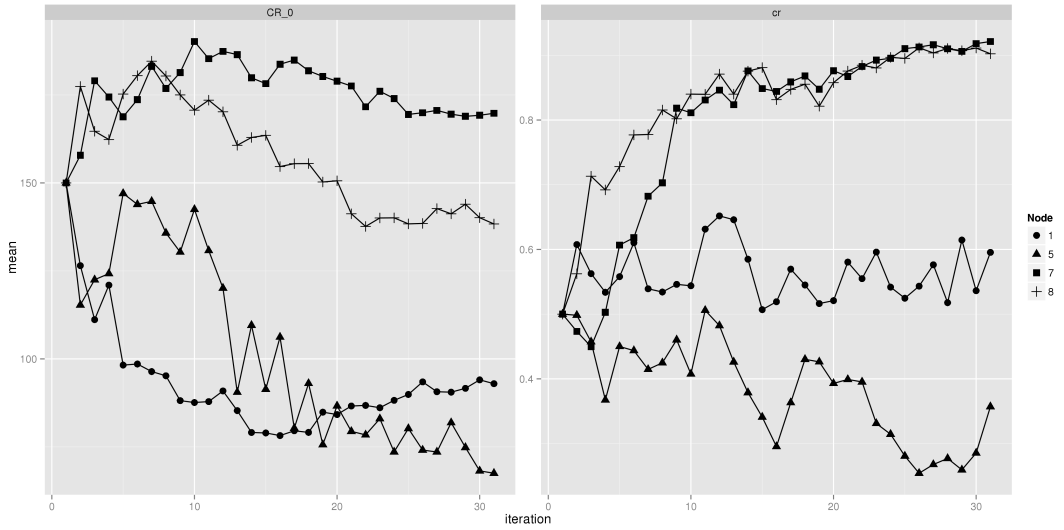

Figure 24: Mean over iteration for some selected nodes. $\overline{CR_0}$ is shown in the left plot, $\overline{cr_F}$ is shown in the right plot

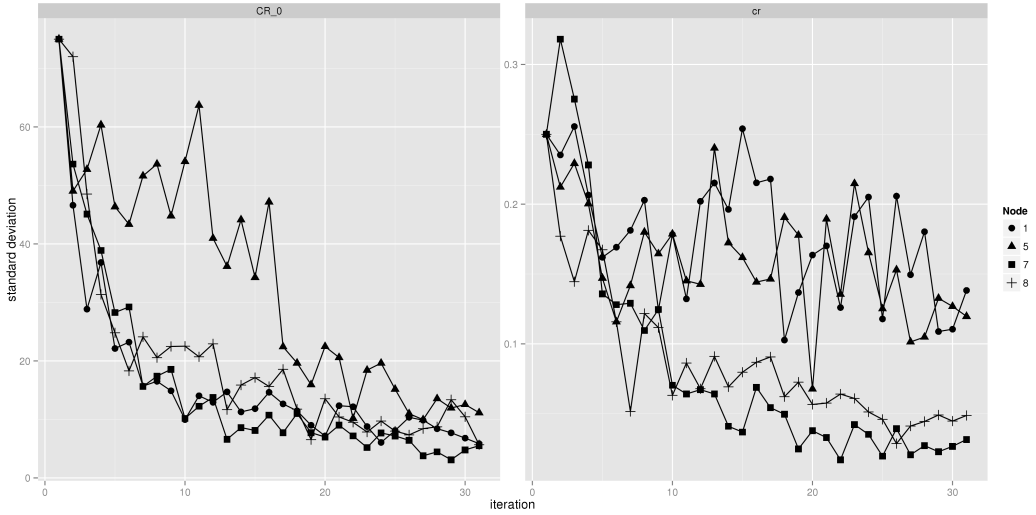

Figure 25: Standard deviation over iteration for some selected nodes. $\overline{CR_0}$ is shown in the left plot, $\overline{cr_F}$ is shown in the right plot

5.2.2. Network simulation

We describe here the tests performed to measure the usefulness and performance of the control system designed for this scenario. The specific objective of the controller will be to self-adapt the network system when unexpected conditions arise in order to maintain a behavior similar to the ideal open loop situation.

We start by defining a parameter configuration set for the control system. Some of the parameters are set as defined by the adjustment methodology described in section 4.2.2. A sensible choice for the node degree tolerance, ξ_{ND}, would be the the minimum number of node degree variation achieved when the communication range is changed by the minimum amount. We can estimate this value from the density of nodes, $\bar{\delta}_{ND}$, and the minimum variation of the area covered by the node, ΔA_{min}:

$$\begin{aligned} \Delta A_{min} &= \pi \Delta cr_{min} (2\mathrm{CR}_{min} + \Delta cr_{min}) \\ \bar{\xi}_{ND} &= \lceil \bar{\delta}_{ND} \Delta A_{min} \rceil \end{aligned} \tag{30}$$

The ceiling operation ensures that the tolerance is an integer value greater or equal than one. The other tolerance parameter, ξ_E, should be set to avoid the execution of the secondary loop when the battery level is far from the critical point. Following Equation (20) and (23), as well as the design of the fuzzy controller, no changes will be made when the absolute value of the error is more than $\bar{E}_{cr}/2$, so this will be our choice for the tolerance parameter.

Regarding the parameters introduced in Equation (25), we will set both ND_{min} and ΔND_{min} to 1, as the number of nodes of the network is very low (just 11); ξ_{ND} will be 1 for this network ($\bar{\delta}_{\mathrm{ND}}$ equals 9.17e-5 nodes/m^2 and ΔA_{min} equals 1650 m^2); and finally, we will choose a value for the parameter $\overline{\Delta cr}$ in the range of 5% to 15% of the range of the *CR* parameter, which is 200 m.

Performance in presence of unexpected node losses

One of the most frequent unexpected conditions that a communication network needs to face is the loss of some nodes. Usually, wired networks deal with this problem by having redundant paths available and a routing protocol able to dynamically modify a route in case of node losses. In the WSN scenario under study, the physical link between nodes can also be dynamically modified by changing the emission power.

In order to test the performance of the control system designed to deal with this situation, we have modified the network model to include the loss of some nodes. The choice of what nodes will be turned off has to be taken in a sensible way, for some connectivity issues can be dealt with by the routing protocol in a similar manner as the wired network case. We have to induce a physical separation of the network into two unconnected subnetworks so that the routing protocol would be unable to solve the connectivity issue.

For the grid-like network topology we have set up, a possible choice would be one of the inner columns or rows. We have selected the third column, composed of nodes 3, 7, and 11, and set it to be lost at second 1000 of the simulation. Then, we execute the simulation with and without the control system active. The parameter $\overline{\Delta cr}$ in this test will be set to 15 meters.

The results are resumed in Table 15. We have included the not normalized performance index for the connectivity, the total number of messages received by the base station, which show an increase of 107.5% when the controller is active. From the other performance indexes, we can observe that the connectivity of the network is improved at the cost of an increase on the energy consumed. However, the index *J* shows that it balances out in favor of the simulation with the control system.

Controller	Messages received by BS	J_E (s)	J_M (s)	J (s)
No	1667	2060.60	1515.45	545.15
Yes	3459	3038.75	3144.55	-105.80

Table 15: Comparison of the performance for the simulated network

In Figure 26 the evolution of the communication range for each node is depicted (node 00 corresponds to the base station and in this figure the data shown is not valid; the actual range of the base station is shown in Table 12). The change of the node degree compared with the nominal reference is shown in Figure 28. Finally, we depict in Figure 27 the battery consumption per second for each node, obtained every 50 seconds. For every figure, two vertical lines have been added: the solid one represents the time each node crosses the critical battery level and the dashed one is set at the time the three nodes are lost.

A few observations can be made based on these figures:

- The controller effectively adjusts the node degree to the nominal reference (within the tolerance region) even after the loss of three nodes in the network.
- The node degree does not oscillate around the reference.
- The battery saving phase is barely noticeable for the node 06, which it the only surviving ones that is set up with a different node degree reference for each phase and it is not in the tolerance region when it reaches the critical level (unlike node 02). This reflects the low impact this phase has on the overall behavior of the system, as was conjectured in the previous subsection.

Influence of the parameter $\overline{\Delta cr}$ in the behavior of the control system

One of the most fundamental objectives of a control system in control theory is to ensure that the corrective actions performed by the controller result in the measured output or controlled variables of the system holding the reference point. We have identified in $\overline{\Delta cr}$ the key parameter related to achieve the best adjustment of the system to the reference, as it directly affects the rate at which the

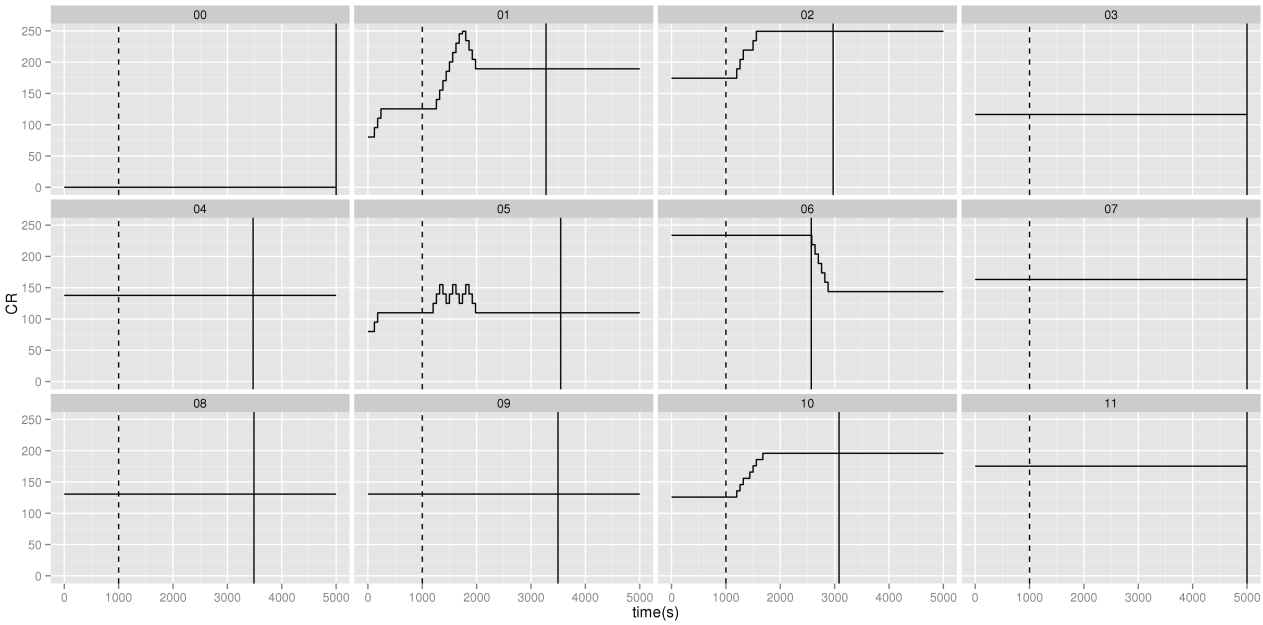

Figure 26: Communication range over time for each node

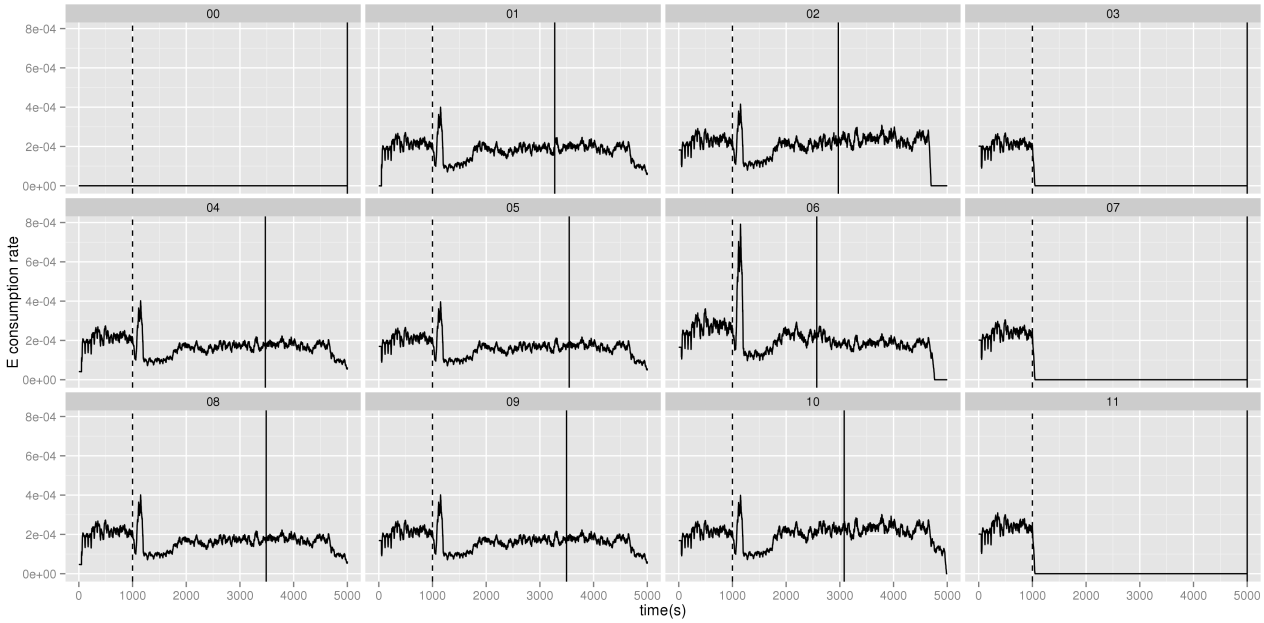

Figure 27: Rate of energy consumption over time for each node

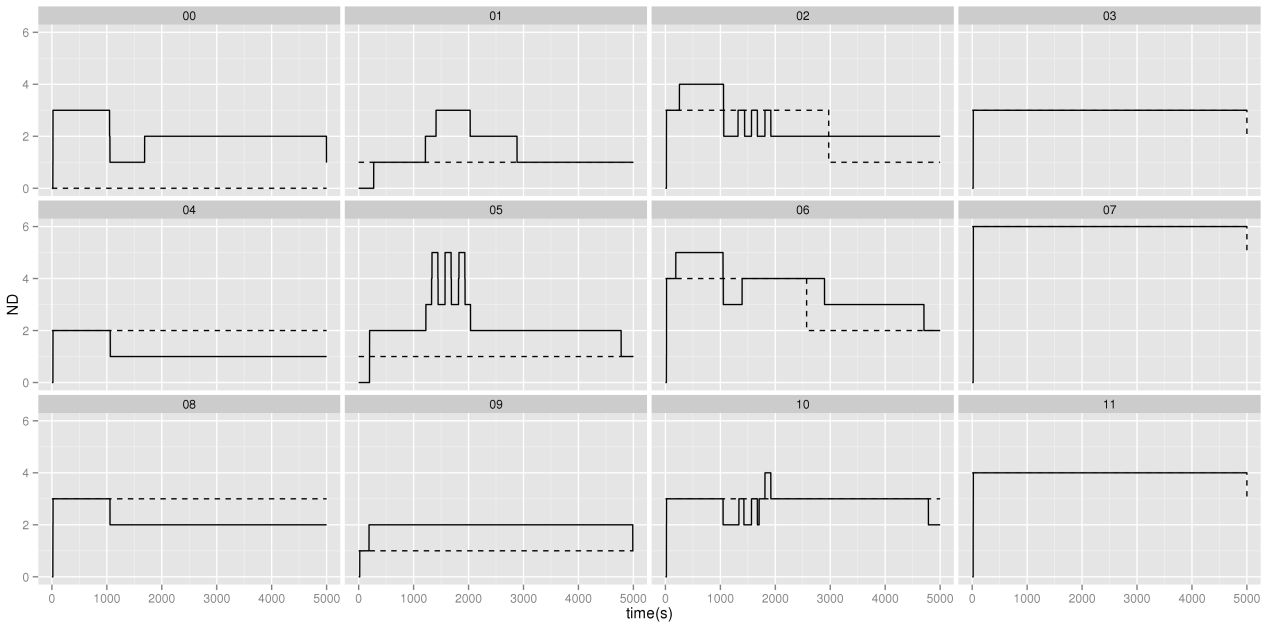

Figure 28: Node degree evolution over time against nominal reference

controller corrects the system towards the set point. Higher values of the parameter may be able to achieve the reference in a shorter time, but at the cost of introducing oscillations in the output of the system; on the other hand, lower values may have a higher adjusting delay, but there should be less oscillations and they should be of a lower amplitude.

In order to test the performance of the control system in relation to the parameter $\overline{\Delta cr}$ we execute the simulation of the same scenario described in the previous subsection with different values of the parameter. We will be comparing each run using the sum of squared errors of the system output variable *ND*. A resume of the results is shown in Table 16.

After the analysis of the results, the following observations can be made:

- The value 25.0 achieves the best performance, both from the point of view of the controller and the system. Comparing Figure 29, obtained from the execution with $\overline{\Delta cr}$ set to 25.0,

$\overline{\Delta cr}$ (m)	SSE	Messages received by BS	J_E (s)	J_M (s)	J (s)
10.0	600	3388	3002.28	3080.00	-77.72
15.0	587	3459	3038.75	3144.55	-105.80
20.0	556	3371	3005.81	3064.55	-58.74
25.0	531	3552	3058.67	3229.10	-170.43
30.0	539	3463	3188.46	3148.18	40.28

Table 16: Comparison of the performance of different controller configurations

and Figure 28, a faster response to the failure at the 1000 second mark can be appreciated, which could help explain the difference in performance.

- The value 30.0 achieves a similar performance regarding the controller, but is significantly worse when we look at the performance index J. However, the difference is explained mainly because of the J_E index, which means that more energy was used. Indeed, in Figure 30 we can see that only one node averages a lower consumption rate when compared to the results with $\overline{\Delta cr}$ set to 15.0.

  As we have previously mentioned, the controller does not make an effort to achieve the most efficient communication range for a given node degree reference. The extra consumption derived from this decision is exemplified in this test.

Influence of the tolerance region

Before the ceiling operation is performed in Equation (30), the actual value of the tolerance corresponding to the selected network parameters is 0.15. This could mean that this network does not need a tolerance, as the nodes should be able to change their node degree one by one. We repeated the simulation for the same five values of $\overline{\Delta cr}$, this time setting the tolerance value to 0.

A summary of the results is shown in Figure 31. Even though the controller error (*SSE*) is decreased in general (with special significance for lower values of $\overline{\Delta cr}$) when the tolerance is null, the performance of the network worsens. We can see a possible explanation in Figure 32, where the

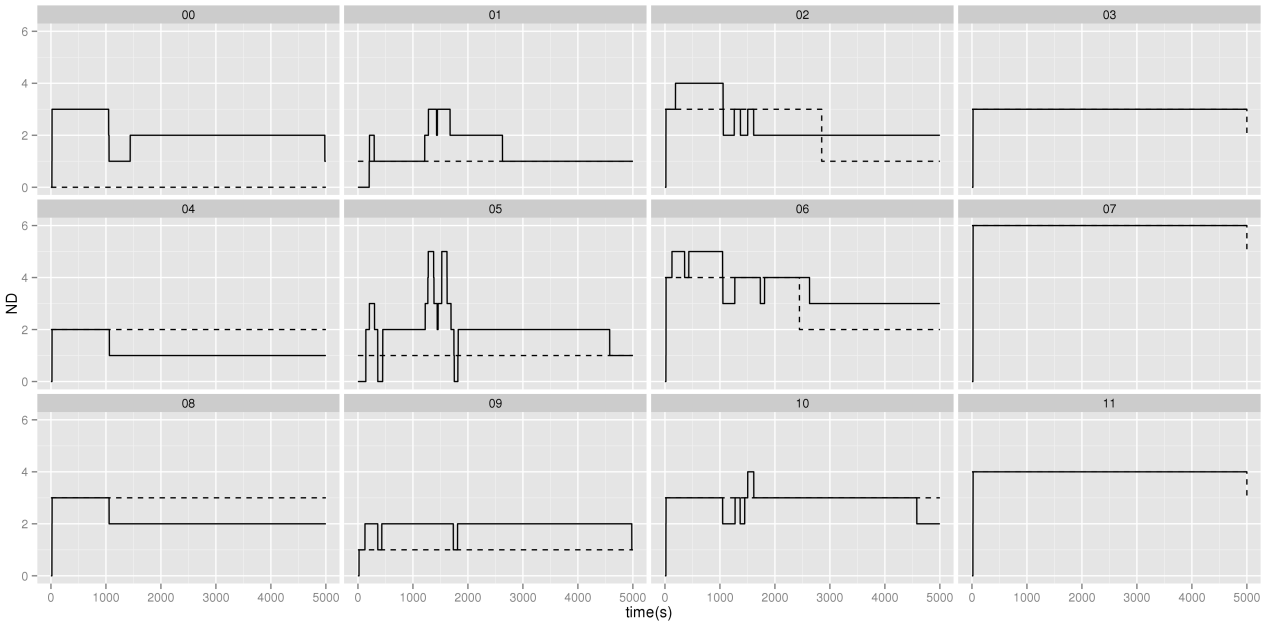

Figure 29: Node degree evolution over time for each node. Data obtained for $\overline{\Delta cr}$ set to 25.0

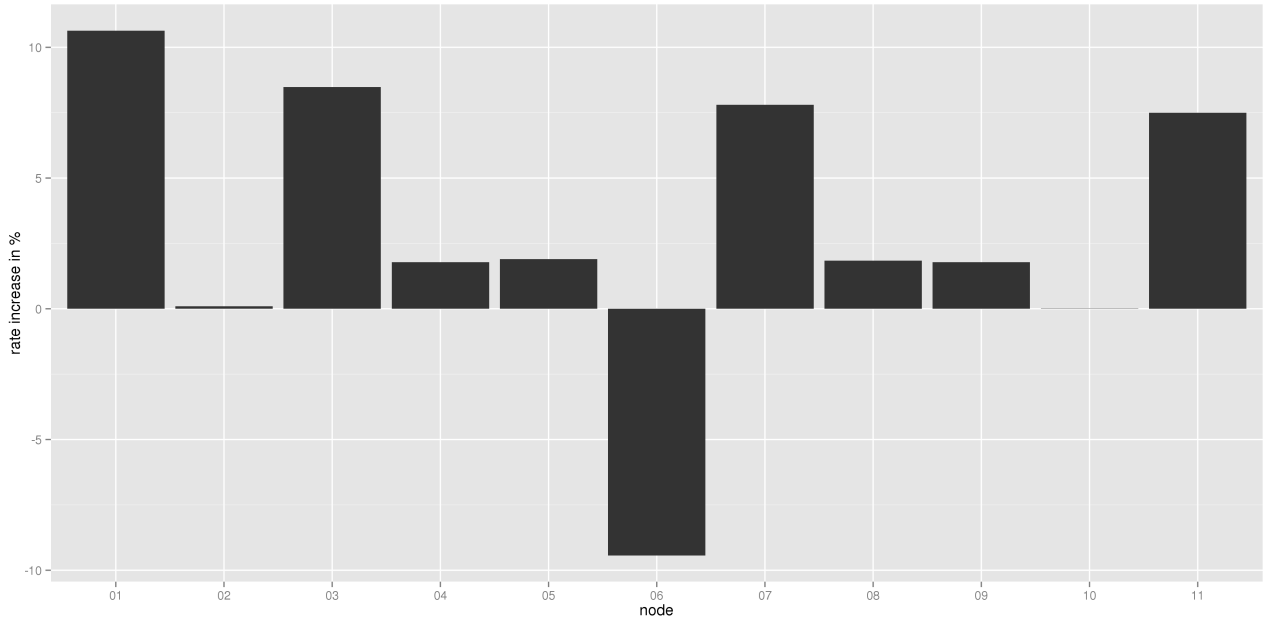

Figure 30: Increase in the mean of the consumption rate for each node when $\overline{\Delta cr}$ is set to 30.0 against 15.0

node degree for each node is plotted against time for the simulation with $\overline{\Delta cr}$ set to 15.0. The lack of tolerance allows the controller to better adjust the node degree to the reference, as we can see in nodes 01, 04 or 08, so that the controller error is reduced. However, the oscillations around low values of the node degree reference in nodes 05 and 09 result in those nodes losing all of their neighbors for short periods of time, decreasing the amount of messages routed to the base station.

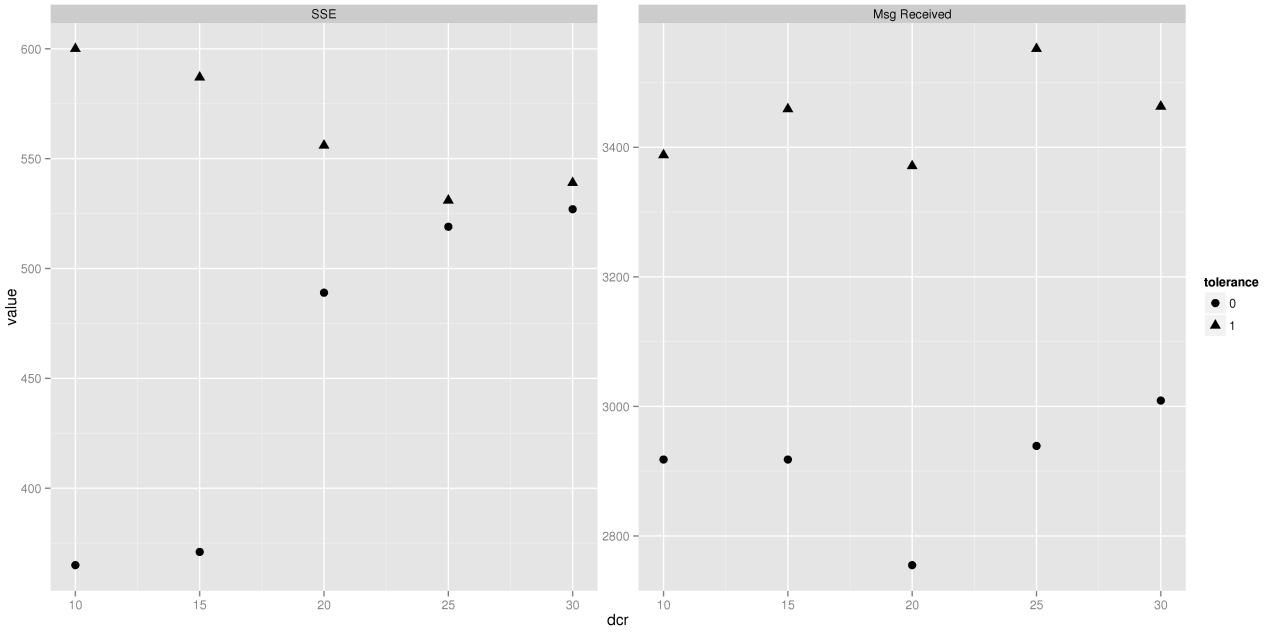

Figure 31: Performance of the controller with and without tolerance. On the left side, the SSE is plotted for each $\overline{\Delta cr}$, while the messages received by the base station are plotted on the right side

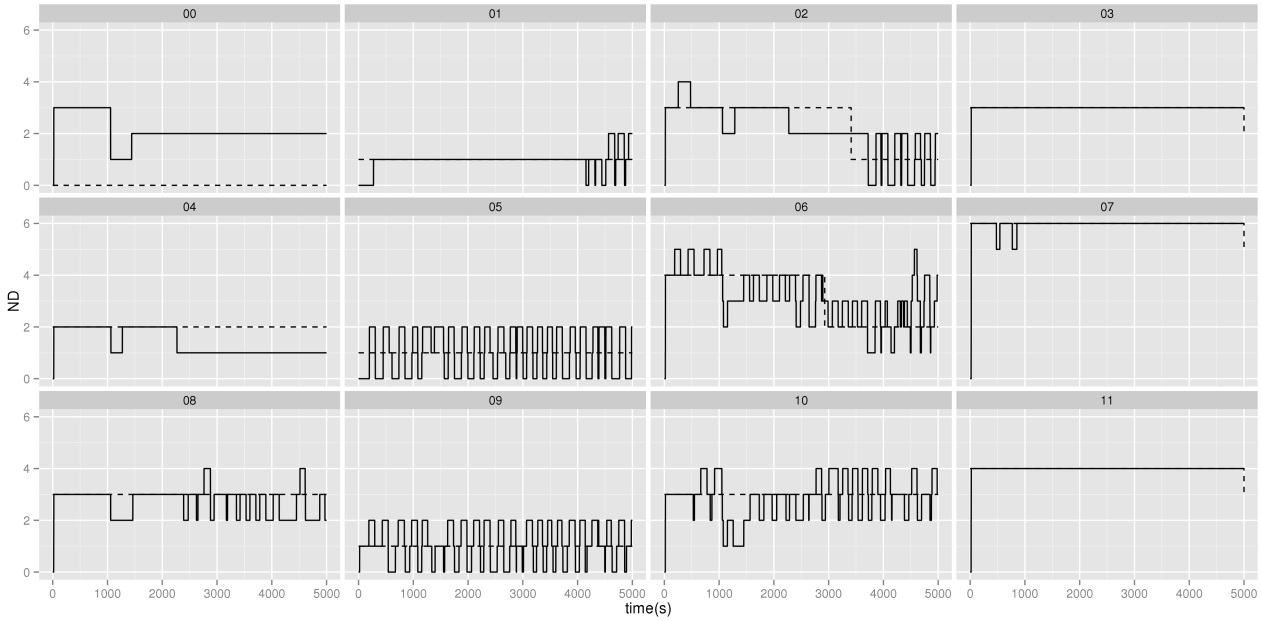

Figure 32: Node degree evolution over time for each node. Data obtained for Δcr set to 15.0 with a tolerance value of 0

6. Conclusions and future work

The complexity of current software systems and uncertainty in their environments has led the software engineering community to look for inspiration in other fields, like robotics, artificial intelligence, control theory and biology, for new ways to design and manage systems and services. In this sense, the capability of the system to adjust its behavior in response to the environment in the form of self-adaptation has become one of the most promising research directions.

However, the software engineering discipline does not provide the necessary tools for the design and analysis of dynamic systems, as it has historically been focused on the development of static applications. At the same time, the field of control engineering centers its study on dynamic processes under uncertainty conditions, while lacking on the construction of large scale software systems. In this project, we have approached the problem from the two angles, borrowing tools and methodologies from both fields aimed at the development of self-adaptive systems.

Our first step was to model into a software framework the concept of a feedback loop, key in the theory of control. This allowed us to treat the feedback loop as any other first class entity in the development process of the self-adaptive software system, which we chose it to be a wireless sensor network. Then, we took the actor oriented approach as the design methodology for the system, making use of already available tools to obtain a simulation of the network as a by-product. Finally, we implemented new self-adaptive strategies for the considered scenarios, built on top of the feedback loop library previously developed, and included in the model of the system as a first class actor.

Given the results of the tests performed on the simulated scenario, the objective of developing, using the proposed methodology, a self-adaptive strategy that improves the performance of a software system has been achieved. Moreover, during the development process we have assessed the value of the combined fields approach, benefiting from the use of standard algorithms and techniques borrowed from control theory, as well as making use of software engineering tools to carry out the design process.

However, we have identified one major drawback in the high completion time of a full simulation of the system. This is also magnified by the fact that the node count of the simulated network was on the low end of usual large scale networks. The issue is specially concerning when automatic optimization algorithms are to be employed as part of the design process of the self-adaptive strategy, as a large amount of simulation runs need to be executed.

Future work should address this issue, possibly by improving the implementation of the system in the modeling tool or, if it is not possible, by exploring the use of other tools. When selecting other software packages, it should be taken into high consideration the possibility of executing the simulation in a parallel system such as a cluster, as the modeled systems are very suitable to be run in this manner.

In spite of this, the Ptolemy II software should not be discarded, as the ability to combine different models of computation on the same system model could prove very useful. A possible extension of this work would be a rework of the gas propagation scenario combining the discrete event model of computation with continuous time semantics to model the gas diffusion process.

Regarding the self-adaptive strategy defined in the communication network scenario, two broad issues should be addressed in forthcoming work. The first one, which also includes the model of the scenario, is the excessive simplification of the behavior of the model from the actual system. Specifically, the battery model, the communication channel and the communication range variation do not represent with enough accuracy their real world counterpart, possibly reducing the effectiveness of the self-adaptive strategy and definitely rendering useless the parameter

optimization step. In order to improve the quality of the models, the knowledge of specialists in the low level details of the system will be essential.

The second issue involves improving the actual strategy. Some questions raised from the test analysis were left open: is it correct to set a common tolerance value for every node in the network, or is it better to try to define the best tolerance value for each node separately? Is it possible to achieve the most efficient configuration that still adjusts the system to the reference? Even though we have considered the possibility of the self-adaptive strategy not being effective in the real world system, answering these questions even for a simplified model could still be very useful for the development of new self-adaptive software systems.

Finally, the methodology described in this work is missing the last step: testing the implementation on a real system. This involves the refinement of the device models above mentioned, as well as porting the solution from the simulation tool to the actual embedded system. Only when the self-adaptive system is in a production environment can we assess the success of this project towards solving a real problem of the society.

7. References

[1] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for re for self-adaptive systems," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*, 2010, pp. 95–103.

[2] G. Brown, B. H. Cheng, H. Goldsby, and J. Zhang, "Goal-oriented specification of adaptation requirements engineering in adaptive systems," in *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, 2006, pp. 23–29.

[3] P. den Hamer and T. Skramstad, "Autonomic service-oriented architecture for resilient complex systems," in *Reliable Distributed Systems Workshops (SRDSW), 2011 30th IEEE Symposium on*, 2011, pp. 62–66.

[4] K. Geihs, P. Barone, F. Eliassen, J. Floch, R. Fricke, E. Gjorven, S. Hallsteinsen, G. Horn, M. U. Khan, and A. Mamelli, "A comprehensive solution for application-level adaptation," *Softw. Pract. Exp.*, vol. 39, no. 4, pp. 385–422, 2009.

[5] H. Liu and M. Parashar, "Accord: a programming framework for autonomic applications," *Syst. Man Cybern. Part C Appl. Rev. IEEE Trans. On*, vol. 36, no. 3, pp. 341–352, 2006.

[6] M. Salehie and L. Tahvildari, "Towards a goal-driven approach to action selection in self-adaptive software," *Softw. Pract. Exp.*, vol. 42, no. 2, pp. 211–233, 2012.

[7] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[8] N. Bencomo, P. Grace, C. Flores, D. Hughes, and G. Blair, "Genie: Supporting the model driven development of reflective, component-based adaptive systems," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 811–814.

[9] T. Vogel, S. Neumann, S. Hildebrandt, H. Giese, and B. Becker, "Model-driven architectural monitoring and adaptation for autonomic systems," in *Proceedings of the 6th international conference on Autonomic computing*, 2009, pp. 67–68.

[10] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Syst. Appl.*, vol. 40, no. 18, pp. 7267–7279, Dec. 2013.

[11] J. A. Tanner, "Feedback control in living prototypes: A new vista in control engineering," *Med. Electron. Biol. Eng.*, vol. 1, no. 3, pp. 333–351, 1963.

[12] G. A. Agha, "Actors: a model of concurrent computation in distributed systems," 1985.

[13] "Unified Modeling Language (UML)." [Online]. Available: http://www.uml.org/. [Accessed: 22-Jun-2014].

[14] "SysML.org: SysML Open Source Specification Project." [Online]. Available: http://www.sysml.org/. [Accessed: 30-Jun-2014].

[15] "The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems | www.omgwiki.org/marte." [Online]. Available: http://www.omgmarte.org/. [Accessed: 30-Jun-2014].

[16] "Ptolemy Project Home Page." [Online]. Available: http://ptolemy.eecs.berkeley.edu/. [Accessed: 29-Apr-2014].

[17] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon, "Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment," in *International Conference on Information Processing in Sensor Networks, 2009. IPSN 2009*, 2009, pp. 277–288.

[18] P. Corke, T. Wark, R. Jurdak, W. Hu, P. Valencia, and D. Moore, "Environmental Wireless Sensor Networks," *Proc. IEEE*, vol. 98, no. 11, pp. 1903–1917, Nov. 2010.

[19] V. C. Gungor, B. Lu, and G. P. Hancke, "Opportunities and Challenges of Wireless Sensor Networks in Smart Grid," *IEEE Trans. Ind. Electron.*, vol. 57, no. 10, pp. 3557–3564, Oct. 2010.

[20] A. Somov, A. Baranov, A. Savkin, M. Ivanov, L. Calliari, R. Passerone, E. Karpov, and A. Suchkov, “Energy-Aware Gas Sensing Using Wireless Sensor Networks,” in *Wireless Sensor Networks*, vol. 7158, G. Picco and W. Heinzelman, Eds. Springer Berlin Heidelberg, 2012, pp. 245–260.
[21] “ns-3.” .
[22] P. Levis, N. Lee, M. Welsh, and D. Culler, “TOSSIM: Accurate and scalable simulation of entire TinyOS applications,” in *Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 126–137.
[23] IBM, “An Architectural Blueprint for Autonomic Computing,” IBM, Jun. 2005.
[24] “IBM developerWorks : Autonomic Computing Toolkit overview.” [Online]. Available: http://www.ibm.com/developerworks/autonomic/r3/overview.html. [Accessed: 01-Jul-2014].
[25] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III, and Y. Diao, “ABLE: A toolkit for building multiagent autonomic systems,” *IBM Syst. J.*, vol. 41, no. 3, pp. 350–371, 2002.
[26] R. Rubinstein, “The Cross-Entropy Method for Combinatorial and Continuous Optimization,” *Methodol. Comput. Appl. Probab.*, vol. 1, no. 2, pp. 127–190, Sep. 1999.
[27] R. Y. Rubinstein and D. P. Kroese, *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer, 2004.
[28] L. A. Zadeh, “Fuzzy sets,” *Inf. Control*, vol. 8, no. 3, pp. 338–353, 1965.
[29] E. H. Mamdani, “Application of fuzzy algorithms for control of simple dynamic plant,” in *Proceedings of the Institution of Electrical Engineers*, 1974, vol. 121, pp. 1585–1588.
[30] E. P. Klement, R. Mesiar, and E. Pap, *Triangular Norms*, vol. 8. Dordrecht: Springer Netherlands, 2000.
[31] J. Eker, J. W. Janneck, E. A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong, “Taming heterogeneity - the Ptolemy approach,” *Proc. IEEE*, vol. 91, no. 1, pp. 127–144, Jan. 2003.
[32] “Java Software | Oracle.” [Online]. Available: http://www.oracle.com/us/technologies/java/overview/index.html. [Accessed: 22-Jun-2014].
[33] “Eclipse desktop & web IDE.” [Online]. Available: http://www.eclipse.org/ide/. [Accessed: 22-Jun-2014].
[34] “Git.” [Online]. Available: http://git-scm.com/. [Accessed: 22-Jun-2014].
[35] “Dropbox.” [Online]. Available: https://www.dropbox.com/. [Accessed: 22-Jun-2014].
[36] “The R Project for Statistical Computing.” [Online]. Available: http://www.r-project.org/. [Accessed: 22-Jun-2014].
[37] “Home | LibreOffice - Free Office Suite.” [Online]. Available: http://www.libreoffice.org/. [Accessed: 22-Jun-2014].
[38] “Zotero | Home.” [Online]. Available: https://www.zotero.org/. [Accessed: 22-Jun-2014].
[39] J. De Oliveira Filho, Z. Papp, R. Djapic, and J. Oosteveen, “Model-based Design of Self-adapting Networked Signal Processing Systems,” in *Self-Adaptive and Self-Organizing Systems (SASO), 2013 IEEE 7th International Conference on*, 2013, pp. 41–50.
[40] C. van Leeuwen, J. Sijs, and Z. Papp, “A reconfiguration framework for self-organizing distributed state estimators,” in *2013 16th International Conference on Information Fusion (FUSION)*, 2013, pp. 499–506.
[41] L. Gao, S. Liu, and R. A. Dougal, “Dynamic lithium-ion battery model for system simulation,” *IEEE Trans. Compon. Packag. Technol.*, vol. 25, no. 3, pp. 495–505, Sep. 2002.
[42] J. Sijs and Z. Papp, “Self organizing distributed state-estimators,” *Intell. Sens. Netw. Integr. Sens. Netw. Signal Process. Mach. Learn.*, 2012.
[43] J. Polastre, J. Hill, and D. Culler, “Versatile Low Power Media Access for Wireless Sensor Networks,” in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, New York, NY, USA, 2004, pp. 95–107.
[44] H. T. Friis, “A note on a simple transmission formula,” *Proc IRE*, vol. 34, no. 5, pp. 254–256, 1946.

[45] S. R. Das, E. M. Belding-Royer, and C. E. Perkins, “Ad hoc On-Demand Distance Vector (AODV) Routing.” [Online]. Available: http://tools.ietf.org/html/rfc3561. [Accessed: 21-Jun-2014].
[46] P. J. <philippe.jacquet@inria.fr>, “Optimized Link State Routing Protocol (OLSR).” [Online]. Available: http://tools.ietf.org/html/rfc3626. [Accessed: 21-Jun-2014].
[47] J. Moy, “OSPF Version 2.” [Online]. Available: http://tools.ietf.org/html/rfc2328#page-185. [Accessed: 22-Jun-2014].
[48] Y. Huang, J.-F. Martínez, J. Sendra, and L. López, “The Influence of Communication Range on Connectivity for Resilient Wireless Sensor Networks Using a Probabilistic Approach,” *Int. J. Distrib. Sens. Netw.*, vol. 2013, p. e482727, Sep. 2013.
[49] A. Kaufmann and D. L. Swanson, *Introduction to the theory of fuzzy subsets*, vol. 1. Academic Press New York, 1975.
[50] K. Ogata, “Modern control engineering.”
[51] R. E. Haber, R. M. del Toro, and A. Gajate, “Optimal fuzzy control system using the cross-entropy method. A case study of a drilling process,” *Inf. Sci.*, vol. 180, no. 14, pp. 2777–2792, 2010.
[52] D. P. Kroese, S. Porotsky, and R. Y. Rubinstein, “The cross-entropy method for continuous multi-extremal optimization,” *Methodol. Comput. Appl. Probab.*, vol. 8, no. 3, pp. 383–407, 2006.
[53] C. Ptolemaeus, *System Design, Modeling, and Simulation: Using Ptolemy II*. Ptolemy. org, 2014.

Appendix A – Description of the parameters of the gas propagation system model

We provide here a complete description of the parameters used in the model of the Gas Propagation scenario defined in section 4.2.1. We only list here the relevant parameters for this project, as some others can be found in the Ptolemy II model but are part of discarded work or not used features.

Parameter	Description
AreaWidth	Width of the area of operations
AreaLength	Length of the area of operations
GasAlpha	Parameter α in Equation (7) for the gas propagation
CellSize	Size of the cells in which the area of operations is divided
Wind	Parameter B in Equation (7) for the gas propagation (without the last 1)
Source	Matrix with the component u^q of the parameter V^q for each cell in Equation (7) for the gas propagation
ChannelRange	Maximum range of a transmission in the channel
PropagationSpeed	Propagation speed for the transmissions in the channel
LossProbability	Chance of a transmission loss in the channel
ByteTxTime	Time it takes to transmit a byte in the channel. It is used as part of the battery model
DtDMsgLength	Length of a message from *Drone* to *Drone*. Used as part of the battery model
DtAMsgLength	Length of a message from *Drone* to *Antenna*. Used as part of the battery model
AntennaReceptionThreshold	Minimum received power needed to process a message
AntennaRecoveryThreshold	Threshold used in collisions. A single message is processed during a collission if its power exceeds the sum of the powers of the other messages by at least this threshold
AntennaLocation	Location of the *Antenna*
DroneAlpha	Parameter α in Equation (7) for the gas estimation
GasSampleCoef	Parameter B in Equation (7) for the gas estimation (without the last 1)
NeighboursTimeout	Number of executions of the decision element that a node remains in the neighbors list since the last received message
InitialComBatteryPower	Initial value of the battery level
SampleTime	Time needed for sampling the level of gas concentration. Used as part of the battery model
SamplePower	Power needed for sampling the level of gas concentration. Used as part of the battery model
ByteTxPower	Power needed for transmission. Used as part of the battery model

Parameter	Description
ByteRxPower	Power needed for reception. Used as part of the battery model
ListenPower	Power needed for listening to the channel. Used as part of the battery model

Appendix B – Description of the parameters of the communication network system

We provide here a complete description of the parameters used in the model of the Communication Network scenario defined in section 4.2.2. We only list here the relevant parameters for this project, as some others can be found in the Ptolemy II model but are part of discarded work or not used features.

Parameter	Description
Seed	Seed used for the random number generation. A value of 0 would use a random seed
EmitterNumber	Number of nodes of the network
BaseControlPeriod	Base period used for the execution of the decision element. Each monitor may define their own periods as multiples of this one
MeanSendInt	Mean parameter of the Poisson process used by the nodes for sending messages to the Base Station
CollectorNumber	Number of Base Stations
ColRange	Communication Range of the Base Stations
LossProbability	Chance of a transmission loss in the channel
AreaLength	Length of the area of operations. Used when randomizing the positions of the nodes
AreaWidth	Width of the area of operations. Used when randomizing the positions of the nodes
HelloInterval	Time interval for sending Hello messages on the OLSR protocol
TCInterval	Time interval for sending TC messages on the OLSR protocol. Should be around 3 times the HelloInterval
Δcr_{min}	Communication range change step
SNRThresholdInDB	Threshold used in collisions. A single message is processed during a collission if its power exceeds the sum of the powers of the other messages by at least this threshold
PowerThreshold	Minimum received power needed to process a message
AntennaEfficiency	Parameter E in the power loss factor
AntennaArea	Parameter A in the power loss factor
MsgSendEnergy	Energy coefficient for sending messages. The total energy spent sending a message is the product of this parameter and the emission power
MsgRcvdEnergy	Energy spent receiving a message
ControlStartTime	Starting time for the decision element