Fraunhofer
FOKUS

Design and implementation of a minimal context reasoning engine for robust dynamic spectrum access wireless communication

by

Kelvin Williams

Matriculation Number 319772

A thesis submitted to:

Technische Universität Berlin
School IV - Electrical Engineering and Computer Science
Department of Telecommunication Systems
Next Generation Networks

Master Thesis

April 15, 2016

Supervised by:
Prof. Dr.-Ing. habil. Thomas Magedanz

Assistant supervisor:
Dipl.-Ing. Bernd Bochow

Eidestattliche Erklärung / Statutory Declaration

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich uter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Berlin, April 15, 2016 Kelvin Williams

Acknowledgments

I would like to thank my supervisor Bernd Bochow for his guidance and insights throughout the course of this thesis.

Abstract

In wireless communication scenarios, noise and interference can impair the quality of a communication. To mitigate this, a dynamic spectrum manager can allocate robust and low interference spectrum, where low latency communication is possible. To make optimal allocation decisions with respect to the application specific requirements, the spectrum manager needs context information about the spectral environment.

In this work a context reasoner was designed and implemented, which provides the spectrum manager with relevant context information about the spectral environment. A fuzzy reasoning approach was chosen to implement the context reasoner, due to the ability of fuzzy inference systems to handle noisy sensor data and uncertainty. Proof of concept rules for the dynamic spectrum access scenario were designed, implemented and validated.

Zusammenfassung

Bei der drahtlosen Kommunikation können Störer und Interferenzen die Qualität der Kommunikation beeinträchtigen. Um dem entgegenzuwirken kann ein dynamic spectrum manager den Kommunikationsteilnehmern robustes und störfreies Spektrum zuweisen, auf dem Kommunikation mit niedriger Latenz möglich ist. Um bezüglich der anwendungsspezifischen Anforderungen optimale Zuweisungsentscheidungen zu treffen, benötigt der spectrum manager Kontextinformationen bezüglich des Spektrums.

In dieser Arbeit wurde ein context reasoner entworfen und implementiert, welcher dem spectrum manager relevante Kontextinformationen über das Spektrum liefert. Aufgrund der Fähigkeit mit rauschbehafteten Messdaten und Unsicherheit umzugehen, wurde die Fuzzy Inferenz als Grundlage für den context reasoner gewählt. Proof of Concept Regeln für das dynamic spectrum access Szenario wurden entworfen, implementiert und validiert.

Contents

Acronyms **xiii**

1 Introduction **1**
1.1 Introduction and Motivation . . . 1
1.2 Objectives and Scope . . . 2
1.3 Outline . . . 4

2 State of the Art and Related Work **5**
2.1 Rule-Based Systems . . . 5
2.2 Ontology-based Reasoning . . . 7
2.3 Artificial Neural Networks . . . 9
2.4 Fuzzy Inference Systems . . . 11
2.5 Related Work . . . 16

3 Methodology **19**

4 Parameters **23**
4.1 Output Parameters . . . 23
4.2 Input Parameters . . . 24
4.3 Parameter Reduction . . . 35

5 Concept and Design **37**
5.1 Requirements . . . 37
5.2 High Level Data Flow Architecture . . . 38
5.3 Context Reasoner Architecture . . . 40
5.4 Interfaces and Messages . . . 43
5.5 Preliminary Considerations . . . 45
5.5.1 Uncertainty . . . 45
5.5.2 Timing . . . 47
5.5.3 Reasoning Time Frames and Sliding Windows . . . 48
5.5.4 Trend and Prediction . . . 49

6 Implementation **51**
6.1 Software and Libraries . . . 51
6.1.1 Boost Libraries . . . 51
6.1.2 Fuzzylite Library . . . 52

6.2 Timers . . . 54
6.3 Threading and Synchronization . . . 55
6.4 Proof of Concept Demonstration and Validation . . . 56
6.4.1 Proof of Concept Fuzzy Design . . . 56
6.4.2 Validation . . . 59

7 Conclusion **63**
7.1 Summary . . . 63
7.2 Outlook . . . 64

List of Tables **67**

List of Figures **69**

Bibliography **71**

Appendices **77**

Appendix **79**
1 Appendix 1: FROUT type definition . . . 79
2 Appendix 2: Interface listing . . . 79
3 Appendix 3: Interface listing . . . 82

Acronyms

ACPI Advanced Configuration and Power Interface. 55

ANN Artificial Neural Network. 9–11, 67

ARIMA Autoregressive Integrated Moving Average. 49

ARMA Autoregressive Moving Average. 49

ASAP Application Service Access Point. 42, 43

ASN1 Abstract Syntax Notation One. 43

BER Bit Error Ratio. 27–32, 34

CLIPS C Language Integrated Production System. 53

DSM Dynamic Spectrum Management. 42

ETX Expected Transmission Count. 16, 17

EWMA Exponentially Weighted Moving Average. 49

F-LQE Fuzzy-LQE. 16, 17

FCL Fuzzy Control Language. 53, 54

FFIS Fast Fuzzy Inference System. 53

FFLL Free Fuzzy Logic Library. 53

FIS Fuzzy Inference System. 53, 54

FisPro Fuzzy Inference System Professional. 53

FleMMingo Flexible Wireless Machine to Machine Communication in Industrial Environments. 39

FLL Fuzzy Lite Language. 53, 54

FRIN Fuzzy Reasoner Input. 40, 41

FSIV Fuzzy Shared Input Vector. 40

HPET High Precision Event Timer. 55

IEC International Electrotechnical Commission. 54

LGPL Lesser General Public License. 54
LQE Link Quality Estimation. 16, 17
LQI Link Quality Indicator. 16

M2M Machine to machine. 23
MA Moving Average. 49
MCS Modulation and Coding Ccheme. 27–31, 34

NI Noise plus Interference. 26–32, 34

OWL Web Ontology Language. 8, 9
OWL-DL OWL Description Logic. 8

PRR Packet Reception Ratio. 16, 17

RAT Radio Access Technology. 30, 31, 34
RDF Resource Description Framework. 8
RDFS RDF Schema. 8
RSSI Received Signal Strength Indicator. 16, 17
RTC Real Time Clock. 55

SAP Service Access Point. 42, 43, 67
SNIR Signal to Noise plus Interference Ratio. 25–32, 34
SNR Signal to Noise Ratio. 16, 17
SWRL Semantic Web Rule Language. 8, 9

TCP Transmission Control Protocol. 39
TSC Time Stamp Counter. 55
TX Transmitter or Transmit. 28–31, 34

UDP User Datagram Protocol. 39

W3C World Wide Web Consortium. 8, 9

XML Extensible Markup Language. 42

1 Introduction

1.1 Introduction and Motivation

This thesis addresses the problem of enabling low latency, low interference, robust and reliable communication in wireless dynamic spectrum access (DSA) environments, where radios can dynamically change spectral bands rather than being confined to statically allocated frequencies. First, the problem is motivated by the example of use cases and scenarios that require low latency and robust communication, after which the solution approach taken in this work is introduced.

Motivation and Use Cases

Within the context of the upcoming 5G standards, terms such as ultra low latency and zero-latency are being used with increasing frequency [1]. The commonly cited latency figure associated with these terms is 1 ms [2]. Such stringent latency requirements are motivated by application scenarios such as industrial manufacturing and automation, self-driving cars, or remote surgery [3].

In indoor industrial environments where wireless M2M (Machine to machine) communication is prevalent, the high number of wireless communications systems in close proximity can cause high interference levels which make wireless communication in these areas difficult to impossible. Additionally, non-communicative electric devices, especially large machinery, can also cause interference. The consequence of high interference levels is a high error rate, many lost messages and required retransmissions. In an industrial automation scenario where robotic assembly systems operate autonomously, delayed or lost control messages and high error rates could lead to financial, material and human damage.

Self-driving cars require low latency and reliable communication to increase safety on roads, where failure to receive messages indicating hazardous situations ahead would have serious consequences. In remote surgery applications low latency and high reliability are required for real-time tactile feedback. Several other use cases for low latency and high reliability communication have been identified and outlined in [4].

Approach

The problem of enabling low latency and robust communication in wireless environments can be approached with a dynamic spectrum manager, observing the spectral environment by receiving sensing information from the communications systems operating in the environment

and allocating spectral bands in which low latency and robust communication is possible.

As part of a previous EU project, QoSMOS (Quality of Service and MObility driven cognitive radio Systems) [5] and an ongoing Fraunhofer FOKUS project ‚FleMMingo (Flexible Wireless Machine to Machine Communication in Industrial Environments) [6], such a spectrum manager for dynamic spectrum access has been designed and implemented. The spectrum manager receives spectrum access requests from client wireless systems and either grants suitable spectral bands according to the clients' requirements or denies those requests. A decision engine implemented within the spectrum manager is responsible for weighing the clients' request and making the spectrum allocation decisions.

While the decision engine already implements basic spectrum allocation strategies, there is a need for more sophisticated ones, which can optimize spectrum allocation with respect to requirements such as less interference and latency, lower error rate, increased bandwidth or higher spectral utilization efficiency. To optimize the decision making process, the decision engine needs context information about the spectral environment, based on which it can make allocation decisions with respect to the clients' and the application requirements.

In this work a minimum context reasoning engine is designed and implemented to supply the decision engine in the aforementioned spectrum manager with suitable facts about the spectral environment. A cognitive rather than an algorithmic approach was chosen because algorithmic solutions require formal mathematical models of the problem space and the system. However, deriving mathematical models that accurately describe real-world systems is often a very complex or even impossible task [7], as dynamical system generally do not have closed-form solutions [8].

While this work is framed in the context of dynamically accessing spectrum for low latency and robust wireless communication, it could easily be reframed to other scenarios. For instance, instead of radio channels, wired links could be considered, which would require a different set of context parameters. Furthermore, instead of the target objectives of allocating low latency and interference links, the requirement could be to provide the most secure link. Both of these alternative scenarios could be approached in the same way as the originally stated one with a context reasoning engine.

In the following section the objectives of this work are stated after which an outline of this thesis is given in Section 1.3.

1.2 Objectives and Scope

This work addresses the problem of dynamic spectrum management by supplying a decision engine implemented within a spectrum manager with suitable facts about the spectral environment which allow the decision engine to make spectrum allocation decisions. This problem is approached by implementing a reasoning engine, operating on context information, inferring facts about the spectral environment and thus allowing the decision engine to allocate spectrum that satisfies the QoS (Quality of Service) requirements of the scenario.

Figure 1.1 shows a simplified view of the spectrum manager architecture within which the context reasoner will operate. The context reasoner receives sensing and utilization reports from sensors and other measurement devices in the spectral environment. These received data are evaluated to extract relevant facts to be supplied to a decision engine, enabling it to allocate available spectrum as indicated by the spectrum portfolio database.

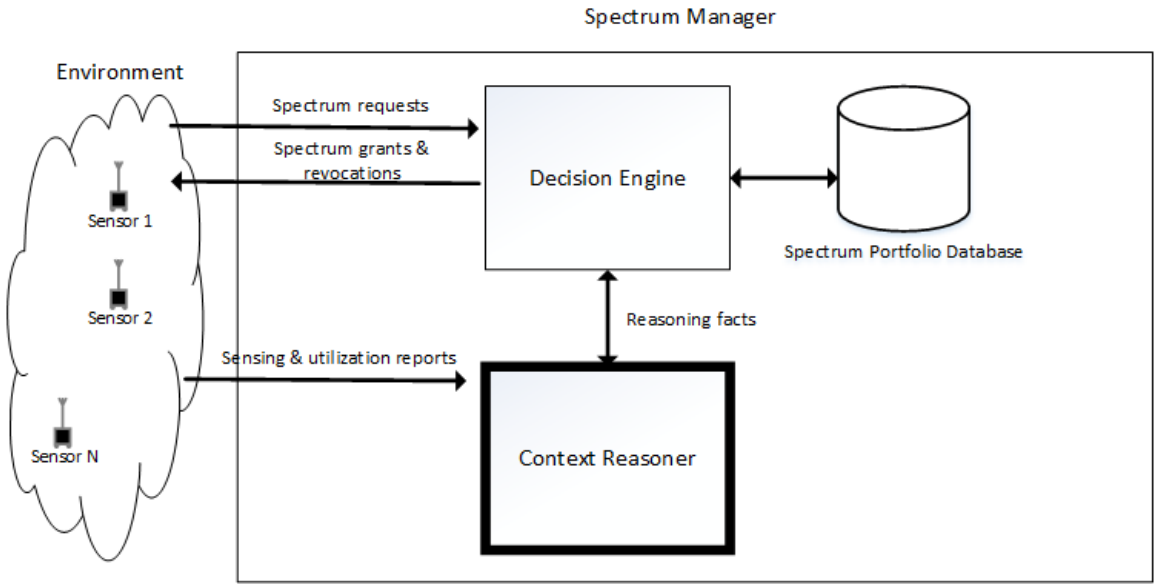

Figure 1.1: Spectrum manager architecture within which the context reasoner is integrated

The following items have been identified as objectives of this work:

(a) Implementing a proof of concept reasoner capable of processing spectrum portfolios. The spectrum portfolio is a data format encapsulating information about spectral bands. Besides pertinent characteristics of the frequency band in question, a spectrum portfolio can also include the usage parameters as well as measurements data of these frequency bands. Spectrum portfolios are used for communicating actions between spectrum managers and spectrum users as well as for exchanging context information about spectral bands. The structure of spectrum portfolios and the operations that can be performed on them has been defined in detail in previous Fraunhofer FOKUS projects mentioned in the preceding section. However, within this work, the purpose of spectrum portfolios is limited to identifying spectral bands, by including portfolio identifiers with the outputs supplied by the context reasoner.

(b) As part of the proof of concept implementation two basic rule-sets shall be set up and demonstrated. These rule sets should implement the following functionality:

- Infer the best match of spectrum portfolios that will minimize interference for a given set of constraints. This objective aims to enable robust and low-interference communication.
- Infer the best match of spectrum portfolios that will maximize the time until increasing interference levels demand to switch over to a different frequency band. This objective aims to maximize the clients' sojourn time in a certain frequency band.

(c) Documenting the results achieved, focusing on flexibility and performance of the solution. The implemented solution should be reusable by other parties and adaptable to other problems and requirements with minimal effort.

The following items narrow down the scope of this work, distilling the focus of the thesis.

These are items that are considered out of scope of this work and will therefore be not be addressed.

- Make spectrum allocation decisions. Making spectrum allocation decisions is a task for the decision engine which is also implemented within the spectrum manager. The reasoner is only responsible for supplying suitable facts to facilitate spectrum allocation decisions.
- Spectrum portfolio manipulation. While it is mentioned in the objectives that a solution must be able to process spectrum portfolios, this only refers being able to read spectrum portfolio data structures and asserting facts about the spectral bands associated with these portfolios. In particular, set theoretic operations on spectrum portfolios such as merging and splitting portfolios, as defined in the QoSMOS deliverables [9], will not be performed within the scope of this work.

1.3 Outline

In the following chapter the state of the art in reasoning is presented and the choice of using a fuzzy reasoning approach is justified. The other reasoning methods presented are rule-based reasoning, ontology-based reasoning and artificial neural networks. Section 2.5 gives a short overview of related work in the field of fuzzy reasoning and in particular in applications closely related to the one in this work.

Chapter 3 presents and explains the methodology used in this work. In Chapter 4 the output parameters for the context reasoner are defined. After defining the output parameters, a detailed analysis of the context space of possible input parameters follows in Section 4.2. From the list of input parameters analyzed in this section, the final input parameters to be used by the context reasoner are reduced in the following section.

In Chapter 5 the spectrum manager environment in which the context reasoner is to be integrated is presented. After that, the context reasoner architecture is introduced along with the modules comprising the context reasoner. In Section 5.4 the interfaces for communication between entities in the spectrum management environment are presented. Various preliminary considerations, such as uncertainty factors and timing issues, are discussed in Section 5.5.

In Chapter 6, the implementation of the context reasoner is discussed. External software and libaries used are presented in the first section of the chapter. In the following sections, relevant implementation details are given. In Section 6.4 a proof of concept fuzzy inference system is designed and used to demonstrate sample rule sets. The chapter concludes with an evaluation and a discussion of the performance of the implemented solution.

In the final chapter the thesis is summarized and conclusions are drawn. An outlook is given, identifying and presenting future work that can be performed to further improve the context reasoner.

2 State of the Art and Related Work

This chapter introduces state of the art approaches to reasoning. Rule-based reasoners, ontology-based reasoners, artificial neural networks and fuzzy reasoners are presented and explained. Following, a short overview of related work is given, to establish a context for the work done in this thesis.

2.1 Rule-Based Systems

A reasoner can be described as a program that is capable of inferring logical consequences from a set of explicitly asserted facts and statements. In a rule-based reasoner the inference process is performed by applying a set of rules to stored facts and axioms in an attempt to model the human cognitive process [10].

A rule-based engine typically consists of a working memory, a rule-base and an inference engine, which in turn consists of a pattern matcher, an agenda and an execution engine. These components are depicted in Figure 2.1

- **Working memory:** The working memory can be likened to the short-term memory of a human. It stores the facts and axioms of the system, which are subject to change and dynamic.
- **Rule-base:** Following with the analogy, the rule-base or production memory would be the long term memory. This is where the rules, also called productions, are stored. Unlike the facts in the working memory, the rules in the rule-base are static. The rules are stored in the form of IF-THEN clauses, where the part that follows after the IF clause is called the *antecedent* and the part that follows after the THEN clause is the *consequent* of the rule.
- **Inference engine:** The inference engine is the active component of the rule-based system. It matches the conditions in the working memory to the rules in the rule-based and then decides which rules to fire.

  While rule-based reasoners operate on rules written in IF-THEN form, resembling conditional clauses in procedural programming languages, it is crucial to understand that the execution of rules in a rule-based reasoning engines is not done in a procedural fashion. Rules are fired when the data satisfy the antecedent, regardless of the order in which the rules appeared in the program.

  - **Pattern matcher:** When there are thousands or even millions of facts and rules stored in the working memory and the rule-base, deciding which rules to fire becomes a hard problem. The pattern matcher is responsible for evaluating combinations of facts and rules to determine which rules should be fired, which is the most expensive

operation in the inference process.

- **Agenda:** The agenda is the component in which the set of rules that are eligible to be fired at any one point, also called the conflict set, are stored. The inference engines in rule-based systems implement algorithms, such as the RETE [11] algorithm, that perform conflict resolution and determine the order in which the rules are to be fired [10]. Conflict resolution strategies might take into account specificity, complexity or priorities of rules when deciding the order in which to fire the rules in the agenda.
- **Execution engine:** The execution engine is responsible for executing the action that is specified in the consequent of a fireable rule. This might range from simply adding or removing facts to and from the working memory to calling procedures and functions.

[12], [10].

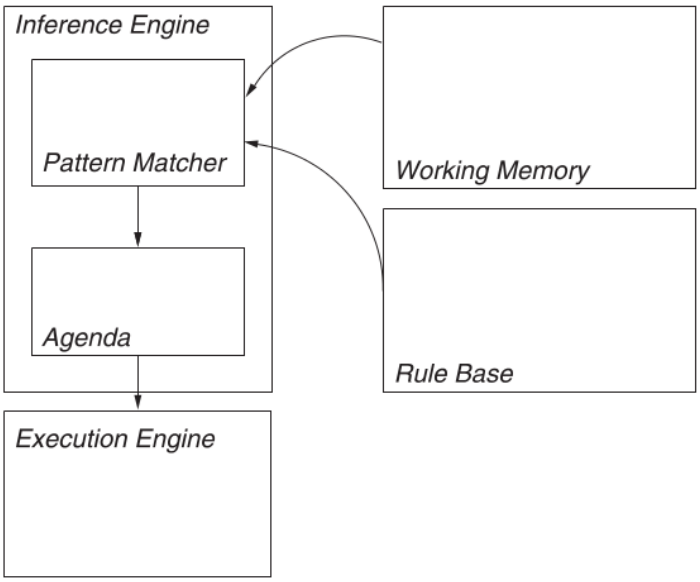

Figure 2.1: Basic components of a rule-based reasoning system [12]

Forward and Backward Chaining

There are two ways in which rules in rule-based systems can be examined. The first one, called forward chaining, works by first considering the antecedents and the other, called backward chaining works by first considering the consequents of the rules

The more commonly used method of inference is forward chaining, which can be thought of as a data-driven approach to inference. The antecedents of rules are examined and the rules whose antecedents are satisfied by the data in the working memory are fired. Firing rules may change the facts in the working memory which could in turn lead to more rules being fired. Rules will continue firing until there are either no more fireable rules or some goal is reached and the program terminates [13], [12].

Backward chaining, in contrast, can be thought of as a goal-driven approach to inference. Here the inference engine works backwards from a list of goal or hypotheses to find the set

of antecedents that must be satisfied to arrive at the desired goals. The rule-base is searched until a rule is found whose consequent matches a goal or hypothesis in question. If it is not known whether the antecedent of that rule is true, then that antecedent is added to the list of goals. This process is repeated until all required antecedents for the initial list of goals and hypotheses have been found. The backward chaining approach is commonly employed in automatic theorem provers and in applications where one might want to trace back the steps that lead to a conclusion. Backward chaining is also employed by expert systems to generate an explanation of how the system arrived at a particular conclusion [13], [10].

2.2 Ontology-based Reasoning

Perhaps the most commonly used and cited definition of an ontology among computer scientists is the one given by Gruber [14], stating that an ontology is an explicit specification of a conceptualization of some domain of discourse. This means that an ontology of a domain is a description of the vocabulary, the concepts and classes, the properties and the relationships between the concepts and classes in that domain. By creating an ontology, an abstract and simplified view of the domain of discourse is constructed. A set of instances belonging to the classes defined in the ontology is usually referred to as a knowledge-base. However, there is no consensus about where an ontology ends and the knowledge-base begins[15].

The semantic web is the application that is most commonly associated with ontologies by computer scientists. Within the context of the semantic web, ontologies are used for defining and exchanging common terminology between agents, enabling them to have a common understanding of concepts and terms used for describing entities [14].

Figure 2.2 shows an ontology in the domain of family relationships and a knowledge-base of individual instances conforming to that ontology. The ontology defines three classes, namely Person, Man and Woman. The classes Man and Woman are *subclasses* of the Person class, indicated by the filled arrows. The plain arrows represent *relationships* between classes. Relationships are read according to the orientation of the arrow. A Person's father or son are both members of the Man class whereas a daughter or mother are both members of the Woman class, for instance. If there is no arrow, the relationship can be read both ways. Numbers and asterisks at the lines representing the relationships indicate the allowed multiplicity of the relationships, stating how many instances of that class can have this relationship with instances of the class at the other end. An instance of the Man class can only be associated with one instance from the class Person in the father relationship, as people can only have one biological father. On the other hand, a member of the Man class can be one of many sons of a Person.

On the right side of Figure 2.2 is a knowledge-base with instances from the ontology. Some knowledge embedded in the knowledge-base is that there is a man named John Smith who is the father of a woman named Susan Smith, who in turn is the daughter of another woman named Mary Smith.

Once an ontology has been developed, it can be used to assert facts and make statements about members in the domain of discourse. These facts and statements can then be understood by any agent in possession of the ontology definition, which is one of the reasons for developing and distributing ontologies. Among some of the possible reasons for developing an ontology are:

- Sharing a specified understanding about a domain of discourse

- Allowing domain knowledge to be reused
- Making assumptions about and within that domain explicit. [15]

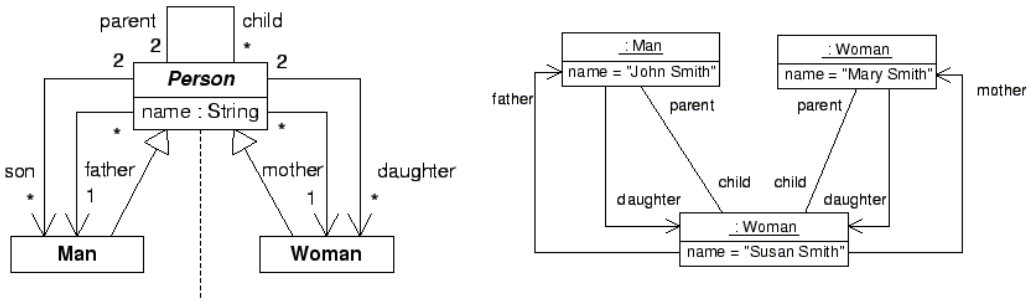

Figure 2.2: An ontology in the domain of family relationships (left) and a knowledge base of instances made from that ontology (right) [16]

OWL (Web Ontology Language)

OWL is an ontology language specified by W3C (World Wide Web Consortium) and is the de-facto standard language used to described ontologies. Despite its name, the language is not restricted to applications on the web. OWL builds on and extends RDF (Resource Description Framework) and RDFS (RDF Schema), which are less powerful languages for describing ontologies. RDF mainly supports simple predicate statements whereas RDFS extends RDF to include support for describing subclass and property hierarchies, with domain as well as range definition of these properties [17]. OWL defines several sub-languages of which OWL-DL (OWL Description Logic) is the most commonly used one. Ontology languages that are based on first-order logic without any restrictions have the drawback of possibly not being decidable. This means that it is possible to create expressions which would take an infinite amount of time to evaluate, as would for instance be the case when making a statement about the infinite set of all natural numbers. To avoid undecidability OWL-DL is simply a fragment of first-order predicate logic, making it less expressive than first-order predicate logic on one hand but decidable on the other hand [18].

Reasoning over Ontologies

The main reasoning tasks applied in ontology-based reasoning are the following [19]:

- **Subsumption:** Determines whether a concept or class is subsumed by another concept defined within the ontology.
- **Inference:** Extends relationships between concepts by propagating the properties of the stated relationships. Common properties are transitivity, functionality, symmetry or irreversibility.
- **Satisfiability:** Determines whether the ontology is free of contradictions.

Rule-based reasoning is typically not supported by ontology languages directly. To enable rule-based reasoning over ontologies and the knowledge-bases formed from those ontologies, several extensions to existing ontology languages have been proposed and implemented. SWRL

(Semantic Web Rule Language) is one such language proposed by W3C [20]. It extends OWL, adding facilites to enable rule-based reasoning over OWL ontologies. SWRL allows the declaration of rules with antecedents and consequent clauses like those employed in rule-based systems. The difference to rule-based systems, as presented in the previous section, is that knowledge does not have to be specifically conceptualized by an ontology in those systems. In particular, rule-based systems do not explicitly specify means for defining classes, relations and other concepts which are at the core of ontology-based knowledge-engineering. However, every knowledge- and rule-based system implicitly or explicitly operates on some conceptualization of the domain of interest [14].

2.3 Artificial Neural Networks

ANN (Artificial Neural Network)s are general methods in machine learning and artificial intelligence for learning unknown target functions from input-output pairs. The structure and functionality of ANNs is loosely based on that of neurons in the human brain.

An ANN consists of *neurons* connected by weighted links. Weights are the primary means of storage of information in an ANN. Furthermore, learning in a neural network is generally accomplished by updating the weights. Figure 2.3 shows a single sigmoid neuron unit from an ANN. The weighted sum of all the inputs into the neuron is used as the input for the *activation function* of the unit. The net input can thus be written as $net = \sum_{i=0}^{n} w_i x_i$. The activation function used is the differentiable logistic function $\sigma(y) = \frac{1}{1+\exp^{-y}}$ and the output of the unit is the linear combination of inputs applied to the activation function[21].

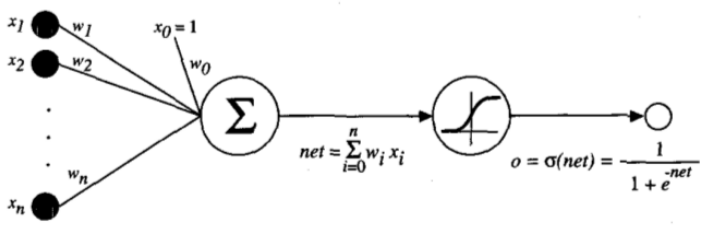

Figure 2.3: A single neuron unit from an ANN [21]

Feedforward Networks

While there are a variety of network structures that can be employed to build an ANN, the simplest and most common network structure is the feedforward network. In a feedforward network, all links are unidirectional and form no cycles, whereas recurrent networks can form arbitrary topologies and exhibit at least one feedback loop in their network structures, hence the name recurrent. Feedback loops can be used to implement dynamical behavior and give the network a memory, enabling applications not possible with regular feedforward network structures [22]. However, feedforward networks are much simpler to train as they are better understood than recurrent networks, which can become unstable or oscillate depending on the input values.

Networks with one or more hidden layers are called *multilayer networks* and have been shown

to be universal approximators [23]. Hidden layers enable the network to learn more complex and higher order functions, than single layer networks [23]. Figure 2.4 shows a multilayer feedforward network with one hidden layer. The network is fully connected, meaning that each node in layer $n+1$ receives inputs from each node in layer n.

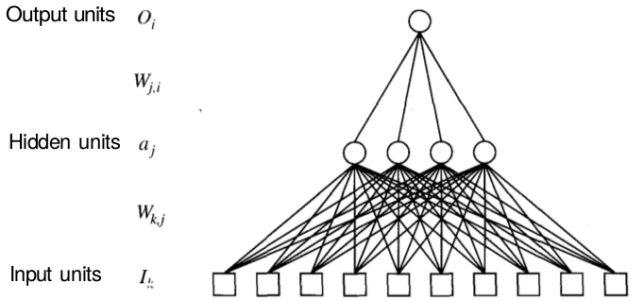

Figure 2.4: Multilayer feedforward network [23]

Learning

The backpropagation algorithm is the most popular algorithm for learning functions in multilayer feedforward neural networks. The backpropagation algorithm performs a gradient descent search in the hypothesis space, by propagating the error in the output back through the network and updating the weights to minimize the error in the learned function. However, learning a function in a multilayer network is not guaranteed to converge to a global optimum and is intractable in the worst case [23]. Furthermore, choosing a too small network will result in the network being incapable of representing complex functions, choosing a too big network, however, will lead to the network "memorizing" all the training examples rather than learning the target function and failing to generalize. The latter phenomenon is called overfitting, to which neural networks are subject, when there are too many parameters in the model.

Capabilities and Applications of ANNs

As already mentioned, neural networks have the ability to learn and generalize functions from a set of input-output values and their distributed nature allows them to solve large-scale problems which are currently intractable. Moreover, as the a neural network essentially performs non-linear regression, they are quite capable of solving problems with noisy and complex input data, such as sensor measurement data and camera or microphone recordings.

Some characteristics of problems that can be solved by ANNs with the backpropagation learning algorithm are[21]:

- **Input-output pairs:** Data is represented by input-output pairs.
- **Target function:** The target function may be real- or discrete-valued or even a vector of real- or discrete-valued attributes.
- **Errors and noise:** The training data may contain errors and noise. Due to the ability to generalize, ANNs are robust against noise in their training sets. Damaged links between individual neurons can be handled gracefully, due to the distributed nature of the network. Extensive damage must be inflicted on the network before its performance is

degraded significantly.

- **Learning time:** Long learning time is acceptable.
- **Fast evaluation:** Once training has concluded, fast evaluation of the internal target function may be required.
- **Transparency:** Transparency and the ability of humans to understand the learned function is not required, as ANNs are essentially black boxes.
- **Prior knowledge:** The problem does not require human knowledge to be applied, as is the case in rule-based reasoning.

Neural networks have been used in a variety of applications ranging from self driving cars [24], to indoor positioning [25] and facial recognition [26]. In recent years there has been ongoing research and development in the field of deep learning, where companies such as Google use deep neural networks to solve interesting and complex reasoning problems, such as mastering the ancient Chinese game of Go, to the level of being capable of defeating professional human players [27].

2.4 Fuzzy Inference Systems

The previous sections introduced different state of the art methods of reasoning that could be used to implement the context reasoner component. Rule-based reasoners and ontology-based reasoners do not possess the inherent capability to handle uncertainty and noisy data. Reasoning with those reasoners is exact and noisy inputs will corrupt the output. Furthermore, while rule-based systems and ontology-based reasoners are decidable, they are not tractable and have an exponential worst-case time complexity [10].

ANNs on the other hand are capable of handling noise and uncertainty well and deliver outputs fast once they have been trained. However, there is no simple method to encode human expert knowledge into ANNs, making them less transparent and more difficult to design for human domain experts. Furthermore, ANNs require a lot of input-output data to be trained correctly. Preferably, the data should come from the environment in which the network operates after the training phase. As only synthetically generated data is available in this work, using ANNs to implement the reasoner would be a suboptimal choice.

The context reasoner implemented in this work is implemented using a fuzzy reasoning system. Fuzzy reasoners are inherently capable of handling noise and uncertainty and can also handle numerical context, thus meeting the requirements set for the reasoning engine. This section presents the concept of fuzzy logic and introduces fuzzy inference systems.

Fuzzy Sets

Fuzzy sets were proposed by Lotfi Zadeh as a generalization of classical set-theory to define the concept of degrees of membership in a set [28]. In conventional set-theory, a set is a collection of entities which form the set. Entities from the universe over which the set is defined are either members of specific set or they are not. There are no other possible degrees of membership in classical set-theory other than *fully* or *not at all*.

Fuzzy sets expand the theory of classical crisp sets to allow entities to have degrees of mem-

bership in sets. A fuzzy set A over a universe of discourse U has a membership function $\mu_A(u)$, which defines the degree of membership of all members u of the universe U in A. the membership function $\mu_A : U \rightarrow [0,1]$ completely characterizes the fuzzy set A. It maps each value in the universe to a degree of membership in A, so that A can be written as the set $A = \{(x, \mu_A(u))\}$. The most commonly encountered membership functions are the triangular, trapezoidal, Gaussian, sigmoidal and linear membership functions [7].

Operations on Fuzzy Sets

Basic operations defined on fuzzy sets and their membership functions are:

- **Equality:** Two fuzzy sets A and B on the universe U are equal if their membership functions are equal for all elements in U: $\forall u \in U : \mu_A(u) = \mu_B(u)$.
- **Subset:** The fuzzy set A is a subset of B if $\forall u \in U : \mu_A(u) \leq \mu_B(u)$
- **Intersection or t-norm:** There are various definitions for the intersection operation on the fuzzy sets A and B. The most commonly used one is the min operation. $\forall u \in U : \mu_{A \cap B} = min(\mu_A(u), \mu_B(u))$.
- **Union, t-conorm or s-norm:** The union of two fuzzy sets A and B also has varying definitions. The most commonly used one is the max operation. $\forall u \in U : \mu_{A \cup B} = max(\mu_A(u), \mu_B(u))$.
- **Complement:** The complement A′ of set A is simply $\forall u \in U : \mu_{A'}(u) = 1 - \mu_A(u)$.

Linguistic Variables

A linguistic variable is one which can take the value of words in a language. These values are called linguistic terms. One example of a linguistic variable is "Age". Possible linguistic terms of the variable age could be "young", "old" and "very old".

The set of all values a linguistic variable can take is called a term-set, which can have an infinite number of terms. The term-set for the linguistic variable "Age" could be, $T(AGE) = \{young + notyoung + veryyoung + notveryyoung + veryveryyoung + ... + old + notold + veryold + ... + middleaged + notmiddleaged + ... + extremelyold...\}$, where $+$ denotes the union of the terms [29].

Linguistic variables can be defined as fuzzy sets over the universe of discourse. The linguistic variable "Age" could be defined over the universe $U = [0 - 100]$, with the terms "young", "middle-aged" and "old" associated with membership functions as shown in Figure 2.5

Hedges are linguistic modifiers such as "very", "somewhat" and "quite". When applied to linguistic terms associated with a membership function, a hedge modifies the membership function by applying a predefined application-dependent transformation, such as squaring, on the membership function.

Fuzzy Inference Systems

A fuzzy inference system, also called a fuzzy controller or a fuzzy reasoning engine, uses fuzzy logic to perform reasoning and control tasks. Figure 2.6 shows the architecture of a fuzzy inference system. It consists of four basic components, namely a fuzzifier, a rule-base, an inference-

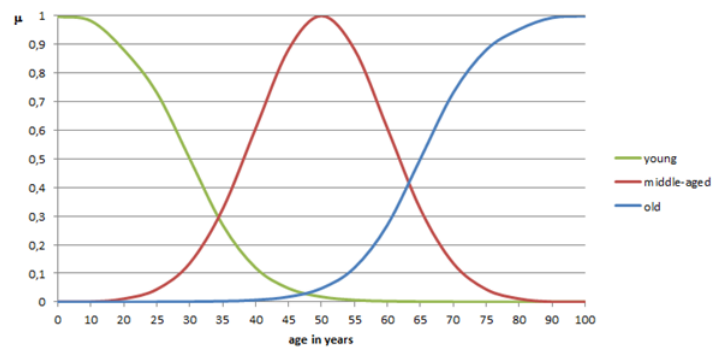

Figure 2.5: Membership functions for the linguistic variable Age and the terms young, middle aged and old [30]

engine and a defuzzifier.

- **Fuzzifier:** The fuzzifier component is responsible for taking the crisp inputs and mapping those inputs to fuzzy sets, by determining the degree of membership in the fuzzy sets of the linguistic variables.
- **Rule-base:** The rule-base holds the heuristic rules that are used by the fuzzy reasoning system to generate the outputs. The rules are simple IF-THEN clauses.
- **Inference engine:** The inference engine is the core of the fuzzy reasoning system. It operates on the rules in the rule-base and the fuzzy sets supplied by the fuzzifier module to derive the fuzzy output sets. A fuzzy rule, IF A THEN B is called an implication, because A implies B. One commonly used implication operator is min implication, also called the Mamdani implication, which truncates the membership function of the consequent to reflect the degree of membership obtained from the evaluation of the antecedent.
- **Defuzzifier:** The defuzzifier module receives fuzzy sets from the inference engine and is responsible for transforming those fuzzy sets into crisp values. In the defuzzification step, the fuzzy sets obtained from the fuzzy inference process are typically weighted and combined to give a crisp output number. Defuzzification methods most commonly used are the *maximum*, *centroid* and the *center of sum of areas* methods [7].

Sample Fuzzy Inference Process

The process of fuzzy inference is shortly explained using the example of the tipping problem, where the percentage to be tipped in a restaurant is determined, based on the quality of the food and the service. The inputs are a rating between 1 and 10 for the food and the service and the output is a number indicating the percentage of the bill to be given. The membership functions for the terms of the input and output linguistic variables can be seen in Figure 2.7. The first column shows the membership functions for the linguistic terms poor, good and excellent of the service variable, while those for the food input and the tip output variable are shown in the second and third column. The following rules are used to determine the amount to be tipped:

1. IF service is poor OR food is rancid THEN tip is cheap.
2. IF service is good THEN tip is average

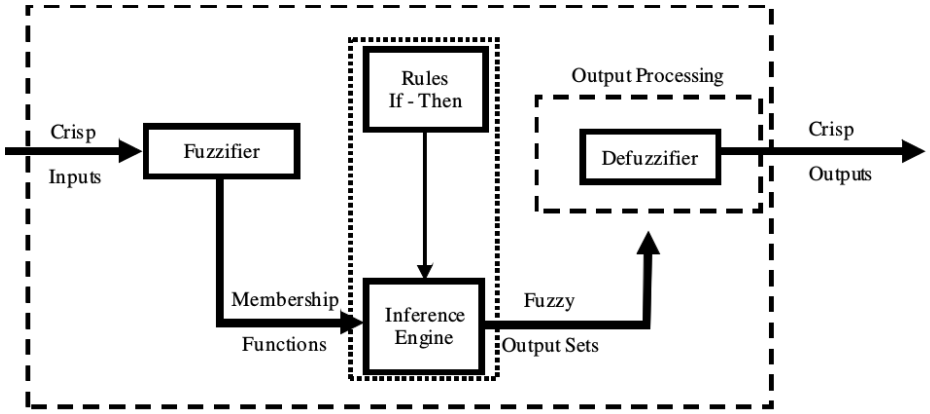

Figure 2.6: Architecture of a fuzzy inference system [31]

3. IF service is excellent OR food is delicious THEN tip is generous.

Following is the sequence of steps a fuzzy inference system performs:

1. **Fuzzification:** The first step taken by the fuzzy inference system is the fuzzification of the crisp input values. In the example illustrated by Figure 2.7, food is rated with an 8 and the service is rated with a 3. These inputs are fuzzified by determining their membership values in the fuzzy sets that are the linguistic terms of the inputs. For instance, the value 3 has a membership of 0 in the fuzzy set of excellent service, whereas the value 8 has a membership of 0.7 in the set of delicious food.

   Next, the rules are evaluated. The first and third rules use OR disjunctions in the antecedent, which in a Mamdani fuzzy inference system is performed with the max operation. For the third rule the max value of the membership of the input in excellent service and the membership of the input in delicious food is 0.7.

2. **Implication:** After the activation values of the antecedents of all rules have been evaluated, the implication method is applied to determine the fuzzy output membership functions. The implication method used in Mamdani inference systems is the min operator. Performing the min implication with the membership value obtained in the previous step, causes the output membership function to be clipped at the level of the membership value. In this example, the membership function of the generous linguistic term is clipped at 0.7 after the min implication step.

3. **Aggregation:** Once all output membership functions from all rules have been determined, they are aggregated. The most commonly used aggregation method uses the max operator, which forms a new output fuzzy set by taking the max value of all the input fuzzy sets over the range of the universe.

4. **Defuzzification:** The aggregation step returns single fuzzy set. Defuzzification is performed on this set to obtain a single crisp output value that is the output of the fuzzy inference process. The most popular defuzzification method is the centroid method, which determines the output to be the center of gravity of the fuzzy set, which can be calculated as $\frac{\int_U \mu_A(u) u du}{\int_U \mu_A(u) du}$ in the case of a continuous membership function.

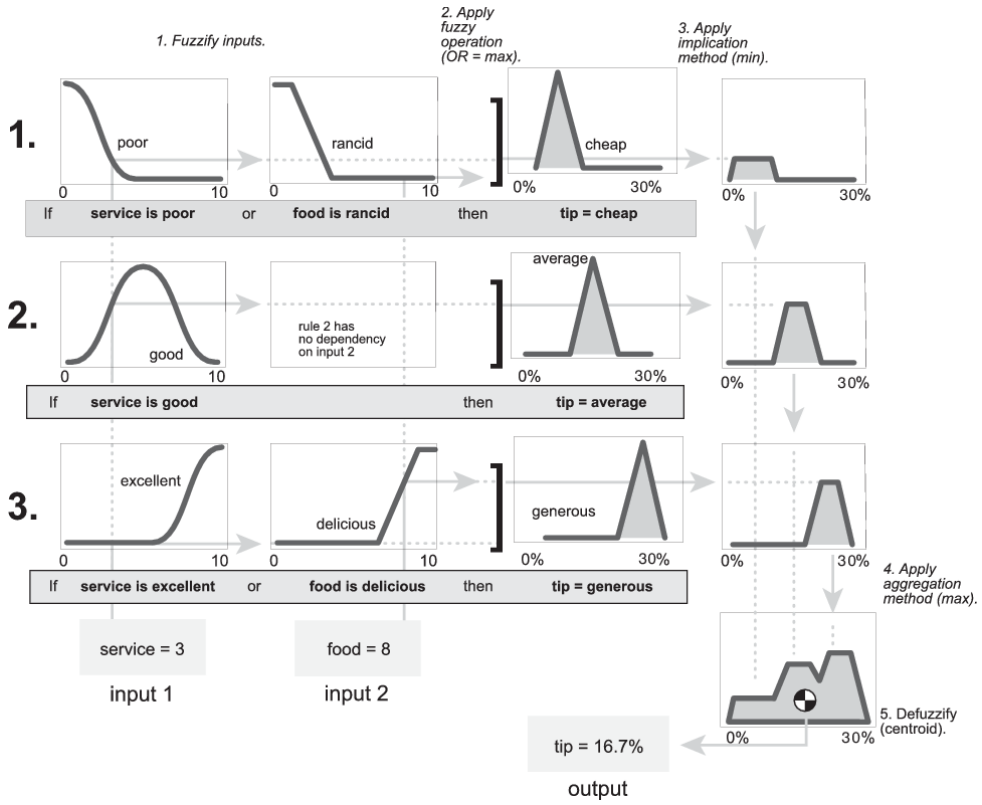

Figure 2.7: The process of fuzzy inference [32]

Advantages of Fuzzy Inference Systems

One advantage of fuzzy systems is that the expert can translate their knowledge to the rules to be used by the fuzzy system into natural IF - THEN constructs. This greatly simplifies communication between the domain experts and the knowledge engineers. Furthermore, systems can be described as a combination of numerical and linguistic values, combining the benefits of purely numerical or linguistic methods.

Fuzzy logic-based solutions can be employed to solve problems for which there are no precise mathematical descriptions or for which such descriptions are only available for restricted conditions, as fuzzy systems are capable of approximating any arbitrarily complex non-linear function, as has been shown in [33]. Additionally, problems for which analytical solutions would be too computationally complex, can also be solved with the help of fuzzy logic.

The ability of fuzzy systems are to make decisions with imprecise and incomplete information and being inherently capable of handling the uncertainty that is associated with natural language is another desirable characteristic of fuzzy inference systems. Therefore, fuzzy inference systems should be considered when the input-data is noisy, as the inputs to the fuzzy system may change over a range without significantly changing the output value of the fuzzy system. This allows fuzzy systems to be robust in the presence of errors caused by noise and imprecision.

In summary the main advantages of fuzzy systems are:

- Due to the approximate nature of fuzzy systems, cheap and noisy sensors can be used
- Human expert knowledge and empirical rules can be used instead of formal mathematical models
- Fuzzy algorithms are robust in the face of highly variable environments and erroneous and forgotten rules
- Simple and computationally efficient reasoning process
- Shorter development time than systems based on rigorous formal models
- Unlike complex differential equations, fuzzy rules are conceptually simple to understand
- Modeling non-linear functions of arbitrary complexity
- Can be combined with conventional control methods [31]

2.5 Related Work

In [34], [35] and [36] a linguistic fuzzy rule-based engine was used to assess terrain safety for the purpose of landing site selection in real-time during spacecraft descent. Inputs obtained from on-board sensors of the spacecraft along with fuel consumption and a measure of scientific return were used to determine a final score for each possible landing site. The scores for each landing site were then used to re-target the spacecraft if the original landing site was found to be dangerous. The final landing site score was obtained by first determining the fuzzy landing site safety score from the raw sensor data. In a second and third fuzzy reasoning stage the overall landing site score was determined by taking into account fuel and scientific return of the site as well as landing site scores of neighboring sites and landing site scores from earlier times in the descent.

In [36], in addition to inferring scores for different landing sites a certainty score is given. The certainty score is an indicator of the confidence in the correctness of the inferred landing site score. The problem of selecting an appropriate landing site from a set of different landing sites based on sensor data and use case specific criteria is very similar to that of selecting a frequency band out of a set of bands based on context data characterizing these bands.

In [37], [38] and [39] a fuzzy logic approach was taken for robot navigation on challenging terrain. A terrain Fuzzy-Rule-Based Traversability Index is inferred with a fuzzy rule set taking into account terrain characteristics such as roughness, slope and discontinuity. A fuzzy reasoning approach was chosen rather than an analytical mathematical approach to obtaining a traversability score due to fuzzy system's ability to deal with uncertain and imprecise data. The Traversability Index acquired by process of fuzzy reasoning is similar to the landing site score in [34].

In [40] Baccour et al. introduce a fuzzy link quality estimator, F-LQE (Fuzzy-LQE). They argue that existing LQE (Link Quality Estimation) techniques such as, PRR (Packet Reception Ratio), ETX (Expected Transmission Count), Fourbit and LQI (Link Quality Indicator) [41], only take into account a single link quality metric and are thus inaccurate. Baccour et al. distinguish between two classes of LQE techniques, namely hardware-based and software-based. Hardware based LQE metrics such as RSSI (Received Signal Strength Indicator) and SNR (Signal to Noise Ratio) are those that can be read directly from the transceivers, whereas software-based LQEs such as PRR or ETX are those that attempt to count or approximate the reception ra-

tio or the average number of packet transmissions and retransmissions. F-LQE, on the other hand, takes into account four link quality properties, namely packet delivery, asymmetry, stability and channel quality. These properties are each defined in linguistic terms and the overall quality of the link is obtained by applying a fuzzy rule set to these link quality properties.

Building on the work of Baccour et al. the authors of [42] incorporated the F-LQE technique proposed in [40] in the WiseRoute routing protocol for wireless sensor networks. Their major contribution was proposing an experimental approach to choosing the fuzzy membership functions in order to maximize the Packet Delivery Ratio of the network.

Jayasri and Hemalatha [43] also built on the work of Baccour et al., proposing a link quality estimator on the basis of hardware metrics such as RSSI and LQI, using a Kalman filter and a fuzzy system. The Kalman filter is used to smooth and eliminate measurement noise of the RSSI values fed into the system.

In [44] Renner et al. developed a link quality estimator that incorporates four link quality metrics, giving a holistic assessment of the link and its dynamic behavior. This work aims to improve link quality estimation by including a metric for prediction accuracy and knowledge about short and long-term behavior of the links by means of the variation and also trend of the link quality. Long and short-term behavior of link quality is assessed by not only taking into consideration current values or the average of the last few values but also the variation of link quality and its trend. Four link quality characteristics are calculated, namely short-term quality, long-term quality, variation and an indicator of the current trend.

In [45] link quality was assessed by using time-series SNR values by modeling the relationship between SNR and achieved communication data rates. While the data rates over SNR plots obtained from measurements generally follow the theoretical curve, achieved data rates still vary greatly for individual SNR values. One measurement shows data rates ranging from 0 to 5 Mbps where the SNR is consistently over 40 dB. Therefore, the authors argue that individual SNR points are not enough to predict channel quality and suggest a time-series modeling, assuming that time-series will allow to predict link quality better than individual SNR points. Both a linear and a non-linear model are investigated, concluding that the non-linear model outperforms the linear model in prediction accuracy of the link data rate based on measured SNR values.

The applications of fuzzy reasoning presented in this section show that fuzzy reasoning is a viable problem solving approach, particularly when formal mathematical models of the problem are not available or too complex and when human intuition provides a sufficiently good solution instead.

3 Methodology

This chapter describes the methodology used in this work. The methodology presented here should enable a reader familiar with the subject matter to independently replicate the outlined process in order to arrive at similar results. The items presented can be regarded as concrete work steps that describe the process of developing a solution to the problem at hand. While the following work steps are enumerated and presented sequentially, it will not be possible to work through these steps in a purely sequential manner. An iterative approach is needed where it will be necessary to go back and revisit previous items as requirements and circumstances change and new information is acquired.

1. **Determine output parameters**

   As a first step in the methodology the output parameters that will be supplied to the decision engine as inputs are defined. The output parameters have to be defined with regard to the objectives and requirements of the problem. Furthermore, the criteria by which each output parameter was chosen has to be stated.

   How: The output parameters are determined with the objectives in mind by asking what parameters the cognitive reasoner needs to make relevant inferences about the state of a spectral band.

   Why: Starting with the output parameters allows one to approach the problem with the end goal in mind. This approach simplifies the process of arriving at a solution by enabling one to successively work from the desired outputs from which the required inputs can then be defined.

2. **Determine input parameters**

   After the desired output parameters according to the requirements and objectives have been defined, the input parameters required to infer the output parameters have to be defined.

   How: The input parameters can be determined by looking at the output parameters and analyzing the inputs on which the outputs depend. After this analysis the dependency of each output parameter on other parameters should be defined. Furthermore, the dependencies of input parameters amongst each other must also be analyzed in this work step.

   Why: This will later help to reduce the set of parameters to the most relevant ones and remove redundancies. The input parameters determined in this step will serve as the set

from which the actual parameters, which will later be used in the implementation, will be chosen.

The reason for defining all possible parameters without regard to whether or not they will be available is to gain a complete as possible overview of the context space, which will later allow a reduction of the list of input parameters, removing redundancies and identifying the most relevant parameters. At this point in the methodology it will not yet be considered where the parameters come from or if they can be obtained at all. When reducing the parameter set at a later point these considerations will be taken into account.

3. **Categorize parameters**

   In this work step the previously determined parameters are categorized with respect to certain characteristics. Note that the categories into which the parameters are classified are application dependent and must be defined to reflect the nature of the application in question. While this work step is listed here as a separate step, it is also possible to categorize the parameters as they are being determined in the previous step.

   How: Parameters are categorized by going through the list of defined parameters and considering their respective characteristics. Such characteristics could be whether the parameters are basic parameters that can be measured, whether they can be calculated arithmetically from other parameters or determined cognitively or whether they are preset parameters.

   Why: Dividing parameters into separate categories is a first step in determining the relevance of parameters. This categorization can help in the next step when the parameter list is being reduced and redundancies are removed. Furthermore, the categorization of parameters will influence the choice of an approach to solving the problem and also the interface design and implementation in later stages.

4. **Parameter reduction**

   During the analysis of input parameters in the previous steps the focus was on modeling the context space of parameters as completely as possible without regard to whether or not these parameters would be used in an actual implementation. Having modeled the context space and categorized the parameters in the previous steps, the list of parameters can now be reduced to those most relevant according to the requirements and objectives.

   How: During the reduction of parameters one will have to consider the actual availability of parameters. This means, that only parameters which can actually be supplied to the system should remain after this step. Furthermore, parameters can be reduced by considering the interdependencies in the parameter context space and choosing the parameters on which the most other parameters depend.

   Why: Parameter reduction is necessary to reduce the complexity of the context space and also to remove unnecessary redundancies. Moreover, the step of parameter reduction will distill the most relevant parameters of those that can be obtained from the parameter context space.

5. **Concept and design**

   After all parameters have been defined, a suitable strategy to infer the output parameters from the input parameters can be determined in this work step. Protocols and messages as well as interfaces are to be defined at this stage. Furthermore, the architecture of the

implementation is determined at this point.

How: The solution approach can be chosen by surveying related work that has been done and picking an approach with similar application constraints. The system is designed by considering the application requirements and possible constraints such as preexisting implementations into which the system might have to be integrated.

Why: The designs specified in this stage will serve as a guide in the implementation stage.

6. **Implementation**

In this work step the decided upon approach is implemented. When implementing the designs it is necessary to also document the implementation. As part of the implementation, a validation must be performed.

How: The solution is implemented by following the steps laid out in the previous work step. It should be noted that the implementation step can affect decisions made in previous steps, as resource and performance constraints have to be taken into consideration in this step.

Why: In a work that is not purely theoretical, an implementation, even if only a proof of concept implementation, serves as a proof of viability of the previously designed solution to a problem.

4 Parameters

This chapter presents an analysis of the context parameters in the dynamic spectrum access reasoning scenario. First the desired output parameters are defined, after which a detailed analysis of the context space is presented. Following, the parameters used in the implemented context reasoner are distilled from the list of parameters in the context space.

4.1 Output Parameters

The problem addressed in this work is finding and allocating spectrum that allows low-latency, low-interference and robust communication. To this end a solution is implemented that supplies a decision engine with facts about the spectrum, enabling the decision engine to make allocation decisions according to the requirements.

The approach taken in this work is to implement a reasoning engine that works on context information about the spectrum and provides facts about said spectrum as output. Suitable output parameters reflecting the stated requirements are defined in this section. These parameters are defined by asking which parameters a decision engine would require in order to make a decision on which spectrum block to allocate or whether or not to allocate a spectrum block at all.

This section first defines how the terms latency and robustness are to be understood and interpreted within the context of this work and then presents the output parameters provided by the context reasoner to indicate the quality of spectral bands.

Defining latency and robustness

The terms latency and robustness have different meanings depending on the application scenario, which is why as a first step the application scenario and requirements must be analyzed before defining those terms.

The size of packets in M2M communications is typically much smaller than that of packets in human generated traffic. Packet sizes are as small as 30 bytes. Typical communication scenarios include those where short control messages and sensor data are repeatedly sent across a link [46] [47]. In some mission critical applications, where late delivery of packets could have catastrophic consequences, systems must guarantee successful transmission of packets with a loss rate of less than 1 per billion. One example of such an application is in industrial automation, where only one in a billion packets may be lost or exceed a given latency delay budget [48], which in some applications can be as low as 1 ms [3].

In such scenarios, where small individual packets are sent periodically rather than large mes-

sages, it is not the effect of the data rate on latency that matters most, but rather the contribution of interference and resulting retransmissions and packet loss.

For the purpose of this work, latency is defined as the delay between the first bit of a packet being transmitted at the sender and that packet being available in the IP layer of the receiver. This definition of latency includes the requirement of successfully having to decode the packet, since only correctly received packets are forwarded to the IP layer.

A robust communication is one where packet loss occurs at a lower rate than a prescribed threshold, such as the above mentioned 1 per billion for instance. The wireless channel characteristic that is directly related to robustness is interference. With all other parameters being equal, the higher the interference on a channel, the less robust of a wireless communication will be possible on that channel.

As can be gathered from the above descriptions, latency and robustness are closely related, which is why there will be one single parameter to indicate the quality of a spectral band and its ability to support communication within the required limits of latency and robustness.

Following are the parameters that will be the outputs of the context reasoner:

- **Quality value:** The quality value estimate refers to the ability of the link to support a communication whit a desired level of latency and robustness as indicated by the packet error rate.
- **Confidence value:** Since the quality estimate will be derived from a reasoning process there will also be a certainty value indicating the context reasoner's confidence in the quality estimate made.
- **Prediction value:** As one of the requirements is to enable the decision engine to infer which channel will provide the best performance for some time in the future, one output parameter should be a prediction value.

Spectrum blocks in the spectrum portfolio database can be tagged with the outputs of the reasoner indicating the quality of each spectrum block. Upon receiving a spectrum request, the decision engine can choose a spectrum block to deploy, by comparing the quality estimates of available spectrum blocks to the client's requirements.

4.2 Input Parameters

In this section the input parameters needed to infer the output parameters determined in the previous section are being identified and categorized. At this point, no regard is given to whether or not these parameters will actually be obtainable later, as this will be considered when performing the parameter reduction in the next work step.

It should be noted that no attempt is being made at capturing every possible parameter and the relationships between them, as that would result in an unnecessarily detailed analysis of said parameters. The parameter analysis is only being performed to a level of detail that will provide enough of an overview of the parameters and how they affect and depend on each other so that in a later stage the most relevant parameters can be identified from this set of parameters.

As mentioned in the methodology chapter, work step 2 and 3, which are " Determine input parameters " and " Categorize parameters " respectively, can be performed simultaneously, as

is the case in this work.

Furthermore, the categories into which the parameters belong are indicated by the letters M, A, C, I in square brackets. These letters stand for the four parameter categories, which are:

- **Measurable**: These are parameters that can be measured directly and reported to the reasoning engine. This shall include parameters whose values do not have to be calculated or inferred from other parameters.
- **Adjustable**: Like measurable parameters, adjustable parameters do not have to be calculated or inferred from other parameters. Adjustable parameters are those that can be tuned or set to certain values and settings at will or in response to other parameters. Optimal parameter settings for these parameters might be inferable heuristically by process of reasoning.
- **Computable**: Refers to those parameters which are neither directly measured nor set but mathematically and algorithmically calculated from a set of other parameters.
- **Inferable**: Parameters for which there might not be a formal mathematical closed form solution with which to calculate them. Those parameters might be inferable by process of reasoning.

Following is the list of analyzed parameters, each with a short description, giving some insight as to why the parameter is being considered. Every listed parameter also includes a list of parameters on which it depends and a list of parameters which it affects. The meaning of these two words in the context of this parameter list is explained as follows:

- **Depends on**: Refers to the parameters that, when changed, will have a direct effect on the value of the parameter in question. Furthermore, it refers to the list of parameters that could be used to calculate or infer the parameter in question if that parameter were not given. It also includes the parameters that can affect the setting of the parameter in question if it is an adjustable parameter.
- **Affects**: Refers to the parameters whose values change if the value of the parameter in question is changed. In case of adjustable parameters, it also refers to those whose settings are chosen in response to the value of the parameter in question.

Bandwidth

Bandwidth is a very basic parameter in this application scenario. When the radios in the communication environment sense the spectrum, measurements will be performed over certain portions of the entire spectrum. Naturally, it is necessary to state the frequencies over which these measurements were performed, so that the reasoning engine can correctly characterize and state facts about the frequency bands in question. The effect of bandwidth on the quality of a communication is twofold. According to the Shannon-Hartley theorem [49], the bandwidth of a channel is proportional to the channel's capacity. However, when increasing a channel's bandwidth, that channel also captures more noise, decreasing the SNIR (Signal to Noise plus Interference Ratio) (Signal to noise plus interference ratio) and thus the quality of the channel. For the input parameter bandwidth, a distinction has to be made between utilized and available bandwidth.

Utilized: [M], [A]: The amount of bandwidth actually used by communicating systems can be

measured and fed into the reasoner as an input parameter. Alternatively, since the total amount of utilized bandwidth is primarily determined by the communications system it is also possible to view it as an adjustable parameter.

Available: [M], [C]: Much like the amount of utilized bandwidth, the amount of available bandwidth can be measured and fed into the reasoner as an input parameter. Additionally, from knowing how much bandwidth is being utilized, the amount of available bandwidth can be calculated. The amount of available spectrum can also be inferred from the spectrum portfolio database.

Affects:

- *SNIR*. Increasing the bandwidth that the receiver senses at a given NI (Noise plus Interference) power spectral density will also lead to an increase in overall NI power and therefore a decrease in SNIR. However, if allowed to transmit with a constant power spectral density when increasing the bandwidth, the SNIR would remain the same.
- *Channel capacity* and thus *data rate*. Through the Shannon-Hartley theorem, the bandwidth is directly related to the channel capacity. The higher the channel's bandwidth, the higher the channel capacity. However, with increasing bandwidth, the SNIR decreases which leads to a reduced channel capacity and thus reduced data rate and communication latency.

SNIR (Signal to noise plus interference ratio)

In wireless communications, SNIR is one of the most fundamental parameters that characterize a communication channel. It can be used as the basis for deriving many other characteristics of the channel. If a radio or a spectrum analyzer is capable of detecting and identifying individual interference patterns and interferers rather than just overall noise and power levels on frequency bands, this information should be shared with the reasoner, as this knowledge will enable the reasoner to infer more accurate facts about the spectrum. Due to the difficulty of detecting interference patterns and the limitations of the sensors, measurements will probably only be able to distinguish between interfering communication system and non-communication system interferers. A distinction is made between in-band and out-of-band SNIR.

In-band [M]: The in-band SNIR is a basic parameter that can be measured directly by the communications systems and fed into the reasoner directly as input.

Out-of-band [C], [I]: Out-of-band SNIR cannot be measured directly, since an SNIR can only be measured if there is an ongoing communication and there is a useful signal. Since out-of-band SNIR cannot be directly measured by the communications system it must be calculated or inferred from the obtained in-band measurements.

Depends on:

- *Spectrum occupancy*. The more nodes are concurrently communicating on a given or even adjacent channels, the lower the SNIR will be at the receiver. Spectrum occupancy information can be extracted from the spectrum portfolio database.
- *Bandwidth*. Increasing the bandwidth at the receiver without increasing the senders transmit power decreases the SNIR. If the transmit power is increased accordingly in order to maintain a constant transmit power spectral density when increasing the bandwidth, the

SNIR will remain constant.

- *NI power levels*. Higher NI power levels on a channel lead to a lower SNIR when the transmit power is kept constant.

Affects:

- *BER (Bit Error Ratio)*. With all else equal, and particularly without an error correction scheme that could mitigate the effects of a noisy communications channel, a lower SNIR leads to higher BER.
- *Channel capacity*. According to the Shannon-Hartley theorem, the channel capacity of a communication channel is proportional to the SNIR of the channel.
- *Data rate*. On a channel with a lower SNIR, the net useful data rate decreases due to a necessary increase of redundancy for FEC (Forward error correction).
- *MCS (Modulation and Coding Ccheme)*. The choice of an MCS (Modulation and coding scheme) depends on the experienced SNIR levels on the channel. Higher SNIR levels will require slower and more robust modulation schemes to be chosen.

NI power levels

The reason NI power levels and SNIR are listed as two separate parameters is because NI power levels can be measured by spectrum sensors without being part of a communication. Noise and interference on a channel both directly affect achievable communication quality on that channel.

[M]: Like SNIR, NI power levels are a basic parameter of the communications channel. They can be obtained by simply measuring a channel's power levels. However, since there will be communicative interferers present, the channel's measured power levels will not be pure noise but rather noise plus interference.

Noise/Interference [I]: Separating the noise and the interference power levels from each other from the obtained NI power level measurements could possibly be done cognitively by recognizing and filtering known interferer patterns. This includes both regular radio channel noise as well as noise from non-communicative interferers, such as starting machines. However, separating the two might prove to be quite difficult as it requires quite advanced filtering algorithms.

Affects:

- *SNIR*. In areas where no SNIR measurements could be obtained, the SNIR could be estimated from the knowledge of the NI power spectral density in the environment.
- *BER, channel capacity, data rate, MCS*. Same as with SNIR

BER

BER of a communications channel is often used as an indicator of channel quality or a factor that is considered when estimating the quality of a channel [50].

[M]: Based on the communicating radios' capabilities, spectrum measurement reports might include experienced BERs

[C]: BERs can be computed from SNIR, NI power levels, MCS and TX (Transmitter or Transmit)

power. BER is a parameter that indicates the quality of a communication channel and as such can be used make estimates about communication latency, robustness and quality in general. BER is inversely proportional to communication latency, as a higher BER leads to more packet retransmissions thus increasing latency.

Depends on:

- *SNIR*. The lower the experienced SNIR on a channel the higher the BER will be.
- *NI power levels*. Same as with SNIR.
- *MCS*. Here, only the BER after forward error correction is considered. The faster the modulation scheme, the higher BER will be for a given SNIR.
- *TX power*. The BER is inversely proportional to the TX power of the sender, since the TX power is proportional to the received SNIR.

Affects:

- *Data rate*. The higher the BER, the more retransmissions may occur which reduces the overall useful data rate. The relationship between BER and data rate is affected by the applied error correction on higher layers.
- Choice of *MCS*. Experienced BER levels on a channel can affect the decision on which MCS to use.
- Choice of *packet size*. For a given BER, smaller packets have a lower probability of retransmission than larger packets. However, the relative packet overhead for smaller packets is higher than that for larger packets. This means that choosing the packet size for a given BER in order to minimize retransmissions is an optimization problem.

Channel capacity

The channel capacity of a communication channel is the theoretical upper bound on the rate at which any information can be transmitted over a noisy channel. It is proportional to the bandwidth of the channel and to the logarithm of the experienced signal to noise ratio.

[C]: Channel capacity can be calculated from bandwidth and SNIR. In theory, the higher the capacity of a communication channel, the lower the achievable communication latency over that channel.

Depends on:

- *Bandwidth*. The higher the available bandwidth, the higher the capacity of the channel. However, increasing the bandwidth of the channel, while keeping the transmit power constant, also increases the amount of noise experienced, which limits the maximum attainable channel capacity by bandwidth increase.
- *SNIR*. The higher the SNIR, the higher the capacity of the channel.

Affects:

- *Data rate*. As the capacity of the channel bounds the achievable symbol rate and the information capacity of a symbol on said channel with a given bandwidth and a given SNIR, the data rate achievable on a channel is also bounded by the channel capacity.

Spectrum occupancy

Current spectrum occupancy information can be extracted from the spectrum portfolio database. The amount of nodes currently communicating has a direct effect on the SNIR levels. Furthermore, communication on one channel, will also affect neighboring channels. The spectrum portfolio database provides information about currently allocated spectrum. By knowing about the geographic and spectral distribution of communicative interferers, the reasoning engine can more accurately characterize the spectral environment. For instance, knowing that a certain number of nodes are communicating at a given time in a given spectral band, in a given geographic region, will enable the reasoning engine to make estimates about the spectral environment based on the given facts.

[M]: Regarded as a measurable parameter, since it can be taken as input directly, without having to calculate or infer it from any other input parameters.

Depends on:

- Current spectrum occupancy is largely an independent parameter, however, it might be possible to predict future spectrum utilization access requests from historic spectrum utilization patterns.

Affects:

- *SNIR*. Current spectrum utilization has a direct effect on the observed SNIR. The more communication there is, the lower the SNIR, which in turn leads to a higher latency due to an increase of packet retransmissions.

Data Rate (Throughput)

Refers to the net data rate of useful data, excluding overhead for FEC and other overhead. Given a message of a certain size, increasing the data rate will decrease the latency of end to end communication. However, increasing the data rate usually means either communicating over a larger bandwidth or increasing the modulation and coding rate, both of which can result in a higher BER and thus to more retransmissions.

[C]: Can be calculated from MCS, SNIR, NI power levels, BER, channel capacity, bandwidth and TX power.

[M]: Actually achieved data rates might be measured and reported by communicating radios in their spectrum utilization reports.

Depends on:

- *MCS*. Theoretically, a faster MCS rate will result in a higher data rate, with the caveat, that this will also increase the BER and retransmissions and thus reduce the total net data rate.
- *SNIR*. According to the Shannon-Hartley theorem, the achievable channel capacity at a given bandwidth is proportional to the SNIR on that channel.
- *NI power level*. Same as SNIR
- *Channel capacity*. This is the upper bound for the theoretically achievable data rate on a channel.
- *Bandwidth*. According to the Shannon-Hartley theorem, the achievable channel capacity

at a given constant SNIR is directly proportional to the frequency bandwidth available.

- *TX Power*: A higher transmit power will lead to a higher SNIR at the receiver, which will lead to lower BER and fewer retransmissions and thus a higher data rate.

Packet size

The choice of packet size can affect the quality of a communication, since packets have to be retransmitted in their entirety if a packet error occurs. Furthermore, reporting packet size in sensing reports, allows the reasoner to more accurately qualify other reported parameters such as experienced error ratios or data rates.

[A]: Packet size can be adjusted depending on the characteristics of the channel in order to minimize retransmissions. For a given data rate and BER, the communication latency is proportional to the packet size. The longer the packet, the more time is required to process the packet.

Depends on:

- *BER*. Packet size could be adjusted in response to experienced BER.

Affects:

- *Data rate*. With all else equal, increasing packet size should increase the data rate over the channel. This is because the ratio of useful data to overhead grows with increasing packet size. However, with increasing packet size, packet errors and thus retransmissions become much more likely, therefore decreasing the data rate. This means, that for a given channel with a given BER, maximizing the data rate with respect to packet size is an optimization problem.
- *Processing and serialization delay*. Larger packets require more processing time. Longer buffering and queuing times will also be incurred by larger packets.

RAT (Radio Access Technology)

The choice of RAT affects parameters such as MCS and data rate. RAT can be chosen based on basic channel characteristics such as SNIR and noise power, in order to optimize the quality of a communication.

[A]: Can be chosen based on characteristics of the communication channel to optimize for the desired quality metrics.

Depends on:

- *SNIR, NI power level and BER*. RAT can be chosen in response to channel characteristics such as SNIR and NI power levels and experienced BERs, in order to enable robust and low latency communication.

Affects:

- *MCS*. The chosen RAT will determine which MCS are available.
- *TX Power*. Some RATs, such as those in the ISM bands, place limitations on permissible transmit powers [51][52].

MCS

MCS describes the type of modulation used in the communication system and also the FEC scheme along with the code rate. With higher modulation rates, higher data rates are achievable, however with higher modulation rates experienced BER increases as well, leading to more retransmissions and thus to a lower effective data rate.

[A]: MCS can be chosen based on channel characteristics and is thus an adjustable parameter.

Depends on:

- *SNIR, NI power levels and BER*. Modulation and coding schemes are chosen based on experienced SNIRs, NI power levels and BERs on the channel.

Affects:

- *Data rate*. Higher modulation rates enable higher data rates. Proportionality between modulation rate and data rate is only given when the channel can be assumed to have an arbitrarily high SNIR [49], since a faster modulation requires a higher SNIR in order limit retransmissions due to errors that occur as a result of the faster modulation.
- *BER*. At a given SNIR the faster the MCS the higher the BER and thus the number of retransmissions will be.

TX Power

Knowing which transmit power was used within a communications system that reports a certain experienced SNIR or BER values is necessary in order to qualify the reported measurements and to make them comparable to those of other communication systems, which might operate with different transmit powers.

[A]: TX power is an adjustable radio setting, that can be chosen in response to the quality of the channel. Increasing the TX power at the sender will increase the SNIR at the receiver and thus lead to fewer retransmissions due to errors.

Depends on:

- *RAT*. As mentioned above under RAT, some RATs prescribe permissible TX power levels limited by regulatory standards.
- *SNIR, NI power levels and BER*. TX power can be adapted as a response to experienced SNIR and BER levels.

Affects:

- *SNIR and BER*. Increasing TX power leads to a higher SNIR and thus a lower BER.

Geographic position

The communication environment will vary with geographic position. Different geographic positions will yield different channel SNIR and BER characteristics. If the application scenario includes mobile nodes the position of those nodes will vary with time and should be reported with spectrum sensing reports. Even in case of stationary nodes the position should either be reported with sensing reports or be known to the system beforehand.

[M]: The position at which a measurement was taken can be itself be measured and fed into the

input of the reasoner directly.

[A]: Position can also be regarded as an adjustable parameter, as the position at which to communicate can be chosen in response to channel characteristics in different locations.

Depends on:

- *Measured SNIR, NI power and BERs*. As an adjustable parameter, position can be chosen in response to experienced SNIR, channel noise and BER if the application scenario allows it, as might be the case in a mobile environment. Even in a non-mobile environment, antenna might be adjustable.

Affects:

- *Measured SNIR, NI power and BERs*. Basic channel characteristics such as these are location-dependent. This is especially the case in indoor environments. Depending on where measurements are taken, sensors will report different values for these parameters when sensing the same frequency bands. By affecting these basic channel parameters, the parameter of position also indirectly affects those parameters that are affected by these basic parameters.
- *(Available) Bandwidth*: Much like with the basic channel characteristics above, available spectral bands might be location-dependent.

Packet delay variation

Packet delay variation is the difference in delay between subsequent packets sometimes also referred to as jitter [53][54]. It is an indicator of channel or link stability and reliability.

[M]: Packet delay variation can be measured by the communicating radios.

Depends on:

- *SNIR*. A variation in packet delay occurs when there is also a variation in the state of the channel. As such, a varying SNIR can have an impact on packet delay variation between communicating radios.

Propagation delay

The propagation delay is the lower bound for signal transmission, limited by the speed of light and the distance between nodes. Propagation delay is negligible for indoor scenarios. It is not negligible, however, when communication occurs over hundreds of kilometers. When choosing one of two links, the propagation delay might be a factor in determining which of the links to choose based on the quality of service requirements.

[M]: Can theoretically be measured, provided that the radios are synchronized with each other, as measuring propagation delay requires high precision timing.

[C]: If the distance between nodes is known, the propagation delay can be calculated.

Depends on:

- Depends only on the distance between communicating nodes, the speed of the signal and the characteristics of the propagation medium.

Processing and serialization delay

On the nodes, packets have to be processed and serialized and deserialized. Buffering and queuing of packets also adds some delay.

[M]: Can be measured by repeatedly taking measurements in the processing systems and statistically calculating the processing and serialization delay. In most cases this delay is negligible, however.

Depends on:

- *Packet size*. Larger packets require longer processing and serialization.

Affects:

- *Packet delay variation*. Since the processing and serialization delay is not a fixed value but a random variable, the packet delay variation is affected by the processing and serialization delay.

Following is a tabular overview of the parameter dependency analysis done in this section:

Table 4.1: Input parameters

Parameter/factor	depends on	affects
Bandwidth Utilized: [M], [A] Available: [M],[C]		SNIR Channel capacity Data rate
SNIR in-band: [M] out-of-band: [C],[I]	Spectrum occupancy Bandwidth NI power	BER MCS Channel capacity Data rate
NI power in-band: [M] out-of-band: [C],[I]	Spectrum occupancy	SNIR BER Channel capacity MCS
BER [M],[C]	SNIR NI power MCS TX power	Data rate MCS Packet size
Channel capacity [C]	Bandwidth SNIR	Data rate
Spectrum occupancy [M]		SNIR
Data rate (Throughput) [C], [M]	MCS Channel capacity Bandwidth SNIR and NI power TX power and BER	
Packet size [A]	BER	Data rate Processing and serialization delay
RAT [A]	SNIR NI Power BER	MCS TX power
MCS [A]	SNIR NI power BER	Data rate BER
TX power [A]	RAT SNIR BER	SNIR BER
Geographic Position [M], [A]	SNIR NI power BER	SNIR and NI power BER RAT Spectrum occupancy and available bandwidth
Packet delay (variation) [M]	SNIR	
Propagation delay [M]	(Distance between nodes) (Propagation medium characteristics)	
Processing and serialization [M]	Packet size (NIC/Hardware specs)	Packet delay variation

4.3 Parameter Reduction

In the previous section a list of context parameters was defined that could serve as input parameters for the context reasoner in the dynamic spectrum access reasoning scenario of this work. In this section, the list of parameters is reduced to the parameters that will be used in the proof of concept implementation of the context reasoner.

The analysis of context parameters and their interdependencies showed that the context space is highly complex, with parameters depending on each other in circular and non-linear ways. A parameter reduction can be performed by considering the parameters which affect the most other parameters. Another factor to be considered when reducing the parameters is whether a parameter will actually be obtainable in the application scenario. Following is the reduced set of the most important parameters for the context reasoner:

SNIR

The parameter analysis in the previous chapter shows that SNIR is a fundamental parameter on which many other parameters depend. Many important channel characteristics and quality metrics, such as BER or channel capacity, can be inferred from SNIR. Therefore, SNIR is used as a context parameter.

NI power

Similar to SNIR, noise and interference measurements are very basic wireless channel parameters affecting many other parameters and giving information about the quality and the state of the channel. While SNIR includes noise and interference power, it makes sense to consider these parameters separately, as that allows the context reasoner to make more informed inferences about the quality of channels, for instance, when comparing two spectral bands with similar SNIR but different noise plus interference power values.

Bandwidth

Since the context reasoner outputs facts about spectral bands, it needs to know which bands received measurements are associated with. Here, bandwidth does not only refer to the width of the band but also to its location in the radio spectrum, which can be indicated by center frequency. This information can either be used in the quality assessment of the channel or it can simply be used to associate measurements and reasoning outputs with certain spectral bands.

These three parameters are the only channel parameters that are used by the context reasoner to make channel quality assessments in this work. While many more parameters could potentially be used, the list of used parameters is intentionally kept small, as one requirement was to implement a *minimal context* reasoning engine. Additional context parameters are not needed, as the presented parameters are sufficient to make statements about the quality of a spectral band.

5 Concept and Design

In this chapter the design of the context reasoner is presented and discussed. As a first step, the requirements for the implementation are set in Section 5.1.

In Section 5.2 the high level data flow architecture of the spectrum manager environment in which the context reasoner component will be integrated is introduced.

The internal architecture and the individual modules comprising the context reasoner component are presented in the following section. Section 5.4 gives a brief overview of the protocol and interface specifications for communication between the entities in the spectrum manager environment.

The chapter concludes with Section 5.5, in which various preliminary considerations are discussed.

5.1 Requirements

In this section the requirements for an implemented solution are outlined. These requirements have been chosen with respect to the application scenario and the objectives stated in Section 1.2

- **Real-time capabilities:**

  The implemented solution should operate and process the context information in real-time or near real-time. Due to the fast-changing nature of the radio spectrum, the implemented reasoner should exhibit real-time capabilities in order to be useful in a real-world deployment scenario.

  Since the specific time limits always depend on the application and must be defined to reflect its requirements, no concrete time limit requirements are set for the implemented solution. However, the solution is to be designed and implemented with the goal of minimizing processing time in mind. To this end the following points should be considered during design and implementation:

  - I/O operations should be limited, particularly disk access, as it is much slower than main memory access.
  - As a corollary to the previous point, data should preferably be kept and processed in main memory.
  - High resolution timers should be used for finer grained control for time sensitive operations and periodic timers.

- Asynchronous operations should be limited as their timing is unpredictable.
- Interrupt-based operations should be limited as they incur processor context switches.

- **Numerical context:**

  The reasoner must be able to operate on numerical context. This requirement arises from the fact that the context reasoner will be integrated in an already existing environment, in which data, in particular measurements, are passed around in numerical form.

- **Robustness against noise:**

  The implemented solution must be able to handle noisy input data. As any measurement of real physical quantities is subject to measurement errors due to noise and other factors, the implemented reasoner must be able to perform robustly in the face of noisy and imprecise inputs.

- **C++ implementation:**

  The system should be implemented in C++. This requirement arises from the fact that preexisting spectrum manager implementations are done in C++. A C++ implementation of the solution will simplify integration of the context reasoner with the spectrum manager system.

  Another advantage of implementing the solution in C++ is that it will help meeting the real-time requirements, as C++ is a non memory managed language and thus avoids the timing penalties incurred by automatic garbage collection.

- **Concurrency:**

  The implemented solution must operate concurrently so that inputs from different sources can be processed simultaneously.

- **Modularity and flexibility:**

  The implementation must be modular so that components within it can be exchanged and modified without affecting other components. Flexibility is required so that the solution can easily be adapted to changing application requirements.

- **Documentation:**

  As already mentioned in objectives the implemented solution should be well documented, so that the system and the code can be operated and modified with minimal effort.

5.2 High Level Data Flow Architecture

This section will describe the high level data flow architecture, of the dynamic spectrum management infrastructure. The interfaces and messages are described in more detail in Section 5.4. This section only aims to introduce the spectrum management environment and give a high level overview data flow in that environment.

There are 3 entities in the dynamic spectrum management infrastructure, namely *spectrum users*, *spectrum providers* and *measurement points*. Spectrum users are those that request spectrum access rights from spectrum providers, which are responsible for granting access and withdrawing spectrum usage rights from users. Measurement points observe the spectral environment and report those measurements to spectrum providers or users depending on the use case.

These three roles can be realized in different combinations within devices or systems. For instance, a wireless network access controller could act as a spectrum provider for wireless end systems, while acting as a spectrum user requesting several spectrum portfolios from a spectrum manager. Interfaces that enable communication between these entities are described in [55]

Figure 5.1 shows the three entities of the dynamic spectrum management infrastructure in a typical scenario. The spectrum manager acts as both a spectrum provider and a context aggregator. It communicates with measurement points, realized on remote sensor devices via the `If_DeviceSensor` interface. Context aggregators receive measurement reports from and pass measurement control information to measurement points through the `If_DeviceSensor` interface.

The cell controller combines three roles, acting as spectrum user, spectrum provider and context aggregator at the same time. As a spectrum user, it sends spectrum access requests to and receives responses from the spectrum manager via the `If_SpectrumManager` interface. As a spectrum provider it can receive spectrum access requests from wireless end-systems and grant or deny those requests. As a context aggregator it receives spectrum utilization reports from wireless end-systems and sensor devices acting as measurement points.

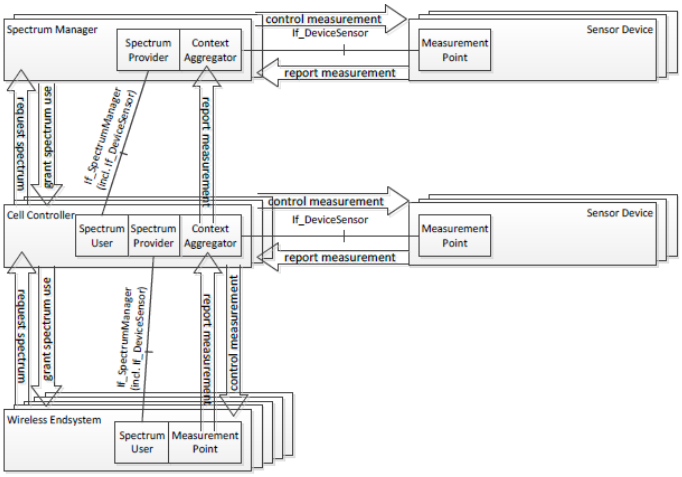

Figure 5.1: Configuration of the entities in the high level data flow architecture of the spectrum management infrastructure [55]

The point of this section is to illustrate how data can be passed between different entities in the spectrum management infrastructure. Most commonly the data is generated in an end-system in the form of measurements and then reported to a context aggregator at a higher layer.

As is suggested by Figure 5.1, context data processing can be done in a hierarchical manner. In such a hierarchy it is possible for one context aggregator to acquire and preprocess data on a relatively low layer before passing on that data to another context aggregator. In such a scenario, the first context aggregator would act as a reporting instance for the second higher layer context aggregator. One possible use case for such a hierarchical scenario would be to filter out redundancies and perform dimensionality reduction in the acquired data in the lower layer context aggregators.

While the context reasoner implemented in this work is designed to be used as a component within a spectrum manager system, it could just as well be utilized within a lower layer context aggregator, such as the cell controller in Figure 5.1. Such a hierarchical model would allow the context reasoner within the spectrum manager in the uppermost layer to work with summarized non-real-time data from several lower layer context aggregators. This hierarchy would enable the spectrum manager to observe larger geographical areas or various cells without having to deal with the high amount of data generated by the measurement points in different locations and cells.

5.3 Context Reasoner Architecture

This section presents the design decision made for the context reasoner architecture. The constituent modules are presented and described. In particular, the modular threading architecture devised to enable real time processing is outlined in this section. When designing the context reasoner architecture, emphasis was put on modularity and flexibility, so that the implementation of this architecture could be reused and modified with minimal effort.

Thread Architecture

The context reasoner implemented in this work is a component within a spectrum manager, inferring facts about the spectral environment, facilitating the decisions of the spectrum manager. To infer such facts about the spectral environment the reasoner requires context information about said spectral environment delivered by reporting instances. To handle incoming data from various reporting instances simultaneously and in real-time, a thread architecture is required.

Figure 5.2 depicts the thread architecture and all the modules that compose the context reasoner component. The architecture follows a modular design, where each module is responsible for a specific task. All modules, indicated by an *M*, run in independent threads and are connected by shared memory structures, allowing inter-thread communication. Following are descriptions of the individual modules in the thread architecture.

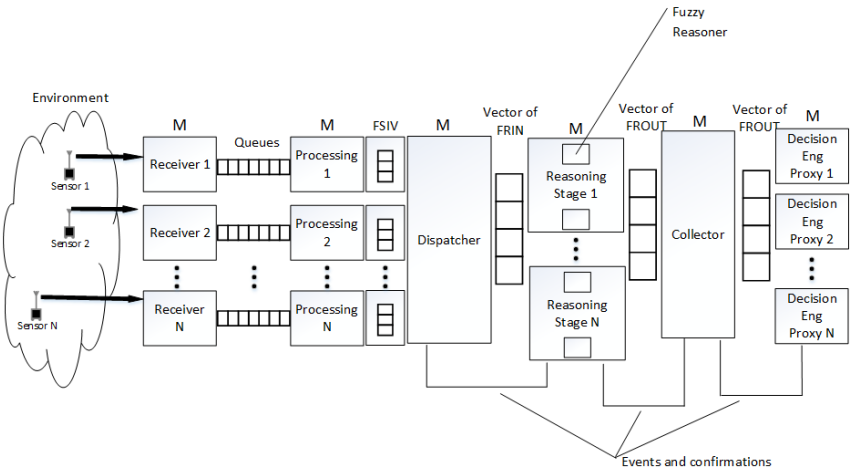

Figure 5.2: Context Reasoner modules and thread architecture

Receiver Module

The receiver module acts as the entry point of the context reasoning component. It is responsible for communicating with the entities providing the reasoning system with input values. Within the context of this work the reporting entities are assumed to be RF sensors. A connection between a reporting instance and the receiver module is established via some communications protocol, such as TCP (Transmission Control Protocol) or UDP (User Datagram Protocol).

There is one instance of the receiver module thread for each established connection. This design avoids the potential of congestion that could occur if there was only one instance of the receiver module. When integrated in the FleMMingo spectrum manager, the receiver module could implement the SAP methods for setting up a remote network connection with other entities as described in the following section. The receiver module places the received data on a queue to which it shares access with the following module in the thread architecture.

Processing Module

The processing module is responsible for any statistical preprocessing of the inputs before passing those values to the input of the reasoning stage. Simple calculations of means and variances of the sampled data as well as more complex calculations should be performed in this stage. The processing module receives raw measurement data from the receiver module, summarizes, filters and processes that data in an appropriate manner and then places it in the FSIV (Fuzzy Shared Input Vector) shared memory structure, where the dispatcher module can retrieve the data and provide it as input to the reasoning stage.

FSIV is a structure which has been defined to simplify communication between the processing and the dispatcher module. It holds a vector of FRIN (Fuzzy Reasoner Input) structures, which are data types that contain the input information for the fuzzy reasoners. Additionally, the FSIV type defines thread safe methods to allow simultaneous access of several processing modules and exclusive access of the dispatcher module to the underlying FRIN objects.

Just as is the case with the receiver module, there is one instance of a processing module for each connection to a reporting instance. Once again, this choice was made to reduce the risk of congestion and to allow fast processing of incoming data.

Dispatcher Module

The dispatcher module serves a twofold purpose. It acts as a synchronization point, temporally bringing together measurements from all the different sensors. Additionally, it delivers the outputs of all the processing modules to the input of the reasoning stage, which contains the fuzzy reasoning engines. As the sensors do not report their measurements synchronously, the measurements have to be synchronized before being delivered to the reasoning stage.

Furthermore, the dispatcher module contains an internal timer, allowing the dispatcher to periodically initiate reasoning iterations. Upon expiration of the timer period, the dispatcher retrieves the data from all the processing modules from the memory structure it shares with them and copies that data to the shared memory structure acting as the input of the reasoning stage. While the reporting instances might report their measurements with different frequencies, the reasoning stage module will receive a set of inputs from all reporting instances from

the dispatcher module at fixed periods. The decision to include a timer inside the dispatcher module rather than in a separate timer module was made to avoid the overhead that would be incurred if a separate timer module had to periodically notify the dispatcher across threads.

The dispatcher module disseminates events to the following reasoning stages, notifying them of a new reasoning iteration. The reasoning stages, or more precisely the fuzzy reasoners in those stages, send back confirmations to the dispatcher once they have finished a reasoning iteration. No new reasoning iteration can be started until the dispatcher has received confirmations from all reasoners. The same event dissemination and confirmation interaction is implemented between the reasoning stages and the collector and between the collector and the decision engine proxies.

Reasoning Stage Module

The reasoning stage module is a container enclosing the fuzzy reasoners. Different fuzzy reasoners with different inputs and rules can operate in parallel within the reasoning stage. This hybrid approach can be chosen when the number of inputs would be too large for one fuzzy reasoner or when it is desirable to separate logically unrelated rules.

There can either be one reasoning stage instance for all reporting instances or one reasoning stage instance for each reporting instance. In the former case, called the non-threaded mode, the hybrid fuzzy reasoners within the reasoning stage iteratively process the inputs of each sensor instance. In the latter case, called the threaded mode, depicted in Figure 5.2, one reasoning stage instance is instantiated per sensor instance, allowing the reasoners to process the inputs in parallel. The choice in which mode to operate the context reasoner depends on the application response time requirements, the number of sensing instances and available computing resources.

Fuzzy Reasoners

The reasoner modules are the functional centerpieces of the context reasoner component. As mentioned earlier, several separate hybrid reasoners can be operated in parallel. The fuzzy reasoners receive their inputs as well as triggering events to start a new reasoning iteration from the dispatcher module. Once the fuzzy reasoners have finished a reasoning iteration they write their outputs to a memory structure which they share with the collector module. This memory structure is a vector of FROUT (Fuzzy Reasoner Output) objects. Similar to the FRIN type, which holds relevant input information for the fuzzy reasoners, the FROUT type, which can be seen in Appendix 1, has been defined to encapsulate output information, reflecting the output that is delivered to the decision engine. The objective of processing spectrum portfolios is met by simply attaching a portfolio ID to each output message as is indicated in the FROUT structure.

Collector Module

The collector is situated after the reasoning stage in the thread architecture. Similar to the dispatcher, there is only one thread instance of the collector module. It shares memory with the preceding reasoning stage or more specifically with the fuzzy reasoners within the reasoning stage. Since the collector module must wait until all reasoners have finished their current rea-

soning iteration and have written their outputs to the shared memory location, the collector acts as a synchronizer for the fuzzy reasoners much like the dispatcher does for the processing modules. After every reasoning iteration the collector module copies the output of the reasoners to a memory location it shares with a thread representing a decision engine.

Decision Engine Proxy Module

This module represents the decision engine that receives facts about the spectral environment from the context reasoner. It is simply responsible for retrieving the output data from the memory it shares with the collector module and is then free to process that data as it sees fit.

5.4 Interfaces and Messages

This section gives an overview of the relevant interfaces between components in the spectrum manager environment. These interfaces allow interaction between the various entities in the spectrum management environment. Moreover, messages defined as part of this work to be included in the communication protocol used by the spectrum manager and the related entities are introduced.

Interfaces are realized through service access points (SAP (Service Access Point)). The most relevant interface for this work is the *If_SpectrumManager* sub-interfaces in the application service access point (ASAP (Application Service Access Point)). The *If_DeviceSensor* sub-interface implements the communication between a spectrum sensor and a spectrum provider or user.

In previous Fraunhofer FOKUS projects protocols and interfaces for exchanging messages between local and remote SAP instances were defined. Of particular interest for this work are the messages and SAP primitives. Messages are arranged in message scopes, defining messages for different purposes, which allows SAP instances to handle messages based on their scope. The two message encodings supported by the SAP implementations described in this document are a type-length-value (TLV) binary encoding and an XML (Extensible Markup Language) encoding.

If_DeviceSensor

The *If_DeviceSensor* sub-interface implements communication between spectrum sensors and spectrum users or providers. The sub-interface extends the `SCOPE_DSM` parameter dictionary of the *SpectrumManager* interface. Messages in the spectrum manager interface are logically organized into message dictionary scopes, grouping together related messages. The relevant message dictionary scope for this work is the DSM (Dynamic Spectrum Management) scope. Following are some of the message types defined for this work, along with a short description.

DS_SENSOR_CAP_IE

Describes the sensing capacities of a sensor instance. It is usually passed from a sensor instance to the client application and can contain information about several distinct sensor instances. An RF device may have more than one sensor instance, each identified by a unique sensor ID. Besides the device ID and the device location, this element contains information about the type of measurements the sensor can perform and also specifies a frequency mask

for wide-band measurements and measurements on multiple frequency bands. Furthermore, the element contains optional context information tied to the sensor instance. Context information can be either spectral, temporal, spatial or device context.

DS_MEAS_VAL_IE
This information element is used for encapsulating sensor measurements and passing those from the sensor instance to the client. The element contains a sensor ID, a sequence number, a time stamp, information about geographic position and the actual measurement data. Following are some of the relevant measurement types that have been defined for this work.

- `T_DS_MEAS_SIG_POWER_IE`: Instantaneous measurement of the received signal power in dBm.
- `T_DS_MEAS_AVG_POWER_IE`: Averaged received signal power over a certain measurement duration in dBm. Includes information about the variance, the averaging window as well as the averaging period.
- `T_DS_MEAS_SIG_SNIR_IE`: Instantaneous measurement of the received SNIR in dB.
- `T_DS_MEAS_AVG_SNIR_IE`: Averages received SNIR over a certain measurement duration in dB. Includes information about the variance, the averaging window as well as the averaging period. Also includes information about the average noise power and the variance.

Additionally, there are elements that describe the context of a sensor instance or of one or more of its measurements. These elements are either passed from the sensor instance to the client upon instantiation of a sensor instance or upon significant change of context. All messages were defined in plain C++ headers rather than with common tools such as ASN1 (Abstract Syntax Notation One) [56] or Google Protocol Buffers [57]. This was done so that the messages could easily be integrated into preexisting headers with message definition from previous projects.

The `If_DeviceSensor` is an instance of the ASAP and as such uses ASAP primitives to create and remove sensing tasks at the remote devices. The `SET` primitives are used for setting one or more values on the remote service. The `GET` primitives are used for polling sensed data. Receiving sensed data and status changes in a push mode is made possible through the `ERROR`, `NOTIFY`, `REGISTER` and `DEREGISTER` primitives. Figure 5.3 shows a message sequence chart of the *If_DeviceSensor* interface for receiving sensor instance reports. The `NOTIFY` primitive is used to receive periodic measurements from a sensor instance. As with all other messages, each measurement must come with a `DeviceID` and a `SensorID` so that it can unambiguously be assigned to a sensor instance. Device and sensor identifications are exchanged during instantiation of a sensor instance.

Output Interface and Messages

So far only the interface relating to communication and message exchange between the context reasoner and sensor has been described. However, the context reasoner has to communicate its reasoning outputs to a decision engine or any other component acting as a client of the context reasoner in the spectrum manager environment. Similar to the definition of the interface and

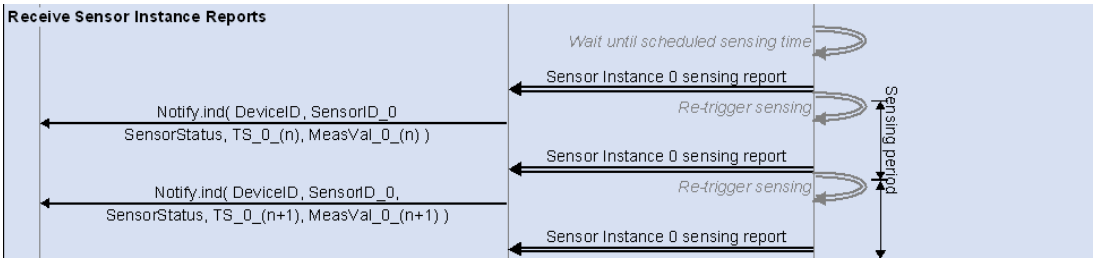

Figure 5.3: SAP interface sequence chart for receiving periodic sensor measurements [55]

messages for communication between the context reasoner and sensing instances, there should be an interface definition between the context reasoner and the decision engine or any other client interested in the context reasoner's outputs. Proposals of interface parameters to be used for communication between a context reasoner instance and a decision engine can be seen in Appendix 2.

5.5 Preliminary Considerations

This section presents some preliminary considerations that have to made before implementing the context reasoner. First uncertainty factors affecting the confidence of the reasoning outputs are discussed, after which the matter of timing is briefly presented. In the following subsection the concept of reasoning time frames is introduced. The section concludes with a brief discussion on methods for predicting future values from past observations.

5.5.1 Uncertainty

When the context reasoner infers facts about the spectral environment, it does so with context information which has some uncertainty associated with it. The output of the reasoning engine must reflect this uncertainty. Therefore, apart from stating facts about the spectral environment, the reasoning engine must also include a measure of certainty or confidence in these facts. Various factors can affect the certainty of a reasoning output. This section provides an overview of the most relevant factors of uncertainty for the scenario addressed in this work.

Measurement uncertainty

Whenever real physical quantities are measured measurement errors are committed. These errors can be random or systematic in nature. Because of these errors, every measurement is subject to uncertainties. These errors are typically caused by noise within the measurement instruments and can be expressed in terms of random variables. Measurement instruments come with margin of error statements indicating precision with which measurement can be performed. Through the method of error propagation, the uncertainties in the inputs of a computation can be forwarded to the outputs.

Variabilty of the measured quantity

The sensing instances periodically sample the spectral environment and report those samples to the context reasoner system. The sequence of reported measurements can be regarded as

a realization of a stochastic process or a time series. From such a time series a sample mean and a sample variance can be calculated. The sample variance is a measure of uncertainty. It quantifies the variability of the physical quantity over time. The higher the variability of that quantity during the observation period, the less confidently one can make statements about the current state of that quantity.

Sampling error

When sampling from a population, all estimates of population parameters such as mean and variance deviate from the true population parameters. The standard deviation of a sampling distribution is referred to as the standard error of the statistic. It indicates how much the estimate of the population parameter deviates from the true population parameter [58].

The standard error of the mean can be calculated from a sample estimate of the mean with $\frac{s}{\sqrt{n}}$, where s is the standard deviation of the sample and n is the sample size. Increasing the sample size thus decreases the standard error [21]. This means that the more values one samples from a distribution the higher the certainty with which one can make inferences about that distribution. Relating this to the situation of sensors sampling the spectral environment, means that a higher sampling rate should result in lower uncertainty of the reasoning outputs.

Jitter

Jitter is the variation of some significant quantity with time. In the case of periodic measurements it is the variation of the period. In the case of packet based communication, jitter is the variance in the delay between the arrival of subsequent packages. The latter form of jitter is also referred to as packet delay variation [53].

Both forms of jitter are sources of uncertainty in the reasoning scenario. In the former case, uncertainty is caused by the measurement period deviating from a constant reference period. Viewing the period as a random variable, the standard deviation of its distribution can be used to quantify the jitter value. Figure 5.4 illustrates how measurement jitter affects the measurement of a time-varying quantity. Instead of obtaining measurements at fixed times $T_1 \dots T_n$, jitter causes the measurements to be obtained within the intervals $T_1 \pm \Delta T \dots T_n \pm \Delta T$. When measuring a time-varying quantity, the amplitude will vary by ΔA within the interval ΔT. Since the progression of the measured quantity over time is unknown, the measurement error ΔA also remains unknown. However, a larger jitter value and thus a larger ΔT results in greater uncertainty, represented by ΔA. Furthermore, the variability of the quantity must also proportionally affect the uncertainty in the measured amplitude.

The second form of jitter, packet delay variation, affects uncertainty in the same way as the first form does. Even if the sensor could theoretically take measurements with 0 measurement jitter, any packet delay variation would obscure this fact on the receiver side, since the receiver is incapable of discerning the contribution of measurement jitter to the experienced packet delay variation.

Recency

Recency refers to how long ago a measurement serving as context information for the reasoner was taken. The longer ago a measurement was taken, the less confidently the reasoning en-

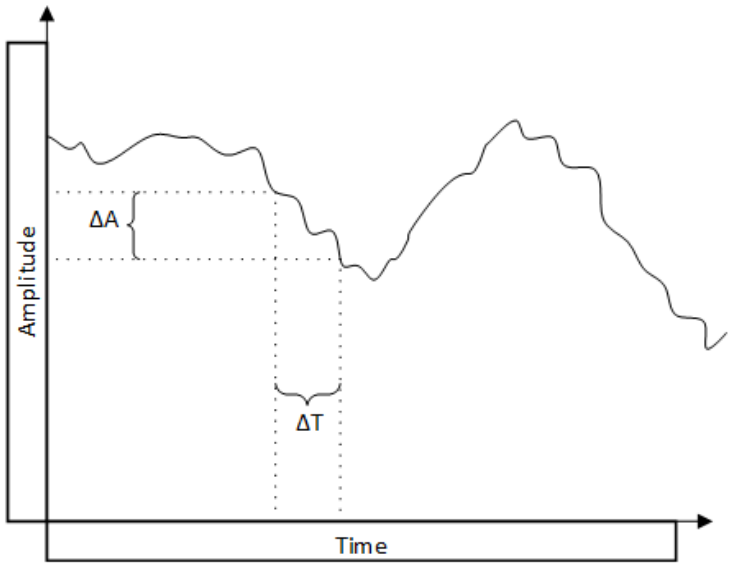

Figure 5.4: Effect of measurement jitter on a time-varying quantity

gine can infer facts about the current and future state of the spectral environment. The effect of uncertainty caused by recency is exacerbated when measuring highly variable quantities. In the reasoning scenario at hand, recency can be taken into account by evaluating timestamps of measurements and adjusting the certainty in an inference accordingly.

In Section 4.1 the reasoner outputs were defined and a confidence value was determined to be one of the outputs. The uncertainty factors presented in this section will be used to determine the confidence in the inference made by the context reasoner.

5.5.2 Timing

In the scenario at hand, sensors are instantiated and instructed to take periodic measurements of the spectral environment. Any measurement instrument takes a finite time to perform a measurement. Depending on the application and scenario, the time it takes to measure the parameters of interest might not be negligible.

Figure 5.5 illustrates the relevant time points and intervals during measuring and reporting. ΔT_M is the time interval it takes the measurement instrument to perform a measurement. ΔT_{Tx} is the time interval it takes to transmit the measurement to the context reasoner, barring transmission errors and retransmissions. $T_{M1}, T_{M2} \ldots T_{Mn}$ are the time instants at which the measurements conclude. These are the points in time to which the timestamps delivered in the measurement reports refer. At time points $T_{TX1}, T_{TX2} \ldots T_{TXn}$ the measurement values are sent from the sensor to the context reasoner. $T_{RX1}, T_{RX2} \ldots T_{RXn}$ are the time points at which the client receives the measurements. ΔT_R is the reporting interval which is simply the reciprocal of the reporting rate.

Sensor measurement durations should be known to the client of the sensor so that appropriate reporting rates can be requested, which are not in conflict with the measurement durations

of the sensors. Measurement durations are the lower bounds on reporting rate, as the sensors cannot report measurements with a faster rate than they can measure.

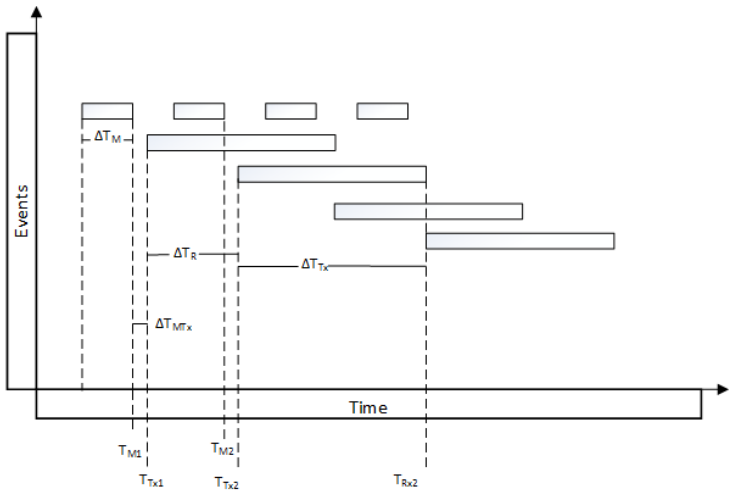

Figure 5.5: Timestamp, measurement duration, reporting period and transmission time of a measurement

Relative and Absolute Timing

When receiving measurements from a sensor, the context reasoner must relate the timestamps of the measurements to its own internal clock. In particular, when receiving measurements from several sensors which all report time according to their own internal clocks, the context reasoner must bring those times together to make them comparable.

If the timestamps reported by the sensors are relative to some previous time, it is simply a matter of adding the newly received timestamp to the previous timestamp. This requires there to have been an exchange of a reference timestamp between the sensor instance and the context reasoner, for instance during initialization of the sensor instance. If the reported timestamps by the sensors are absolute timestamps, the clock offsets between the sensor instances and the context reasoner must be determined by some form of clock synchronization such as IEEE 1588 [59], [60] before the exchange of any timestamped messages. In case the accuracy requirements are not so stringent as to require a synchronization providing the accuracy of IEEE 1588, absolute timestamps can be requested by a `gettimeofday()` call or an equivalent function implemented on the sensors.

5.5.3 Reasoning Time Frames and Sliding Windows

To be able to make more accurate inferences about the current and future state of the spectral environment, historical measurement data must be considered. To this end, the concept of reasoning time frames is introduced. A reasoning time frame is simply the time interval from which past data is taken into consideration. For instance, when using a reasoning time frame of a 100 ms, only measurement data that was received within the last 100 ms will be used by the reasoning engine.

Reasoning time frames can be implemented using sliding windows, as illustrated in Fig-

ure 5.6. The size of the sliding window must be chosen to hold the exact number of values that are reported within the desired reasoning time frame. With a sensor reporting rate R_r and a reasoning time frame of ΔT_{TF} the window size can be calculated as $S = \Delta T_{TF} \cdot R_r$.

Depending on the rate at which reasoning iterations are started, the windows might overlap. The benefit of using reasoning time frames and sliding windows becomes clear when considering the alternative, growing windows, shown in Figure 5.6. A growing window takes into consideration all values since records began. This might be undesirable in a quickly changing environment such as the one which is the spectral environment.

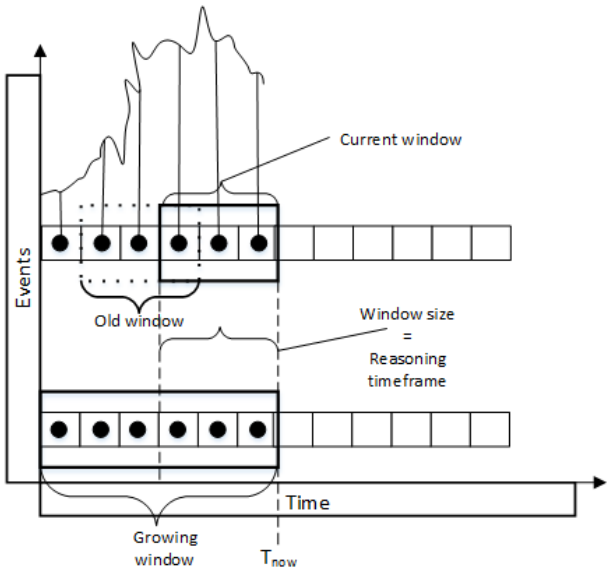

Figure 5.6: Implementing reasoning time frames with sliding windows

For a time frame of merely 10 minutes and a reporting rate of 100 Hz, which is one value every 10 ms, the window size would be 600,000. Since it might be inefficient to keep 600,000 values in memory for each sensor, the mean of n subsequent values instead of every incoming value can be stored in the sliding windows.

5.5.4 Trend and Prediction

When using the reasoning engine in a real environment, as a first step, real data of that environment should be measured and analyzed. When working with real data one usually first finds out the statistical properties of the data, such as the distribution or whether the stochastic process can be assumed to be ergodic or stationary. Based on the findings of this analysis appropriate statistical models can be chosen to model the process generating the data. Since no real data will be used to test the proof of concept implementation, assumptions about the statistical properties of the data generating processes will have to be made where necessary.

There are several ways to make a prediction about the future state of the spectral environment. Statistical models such as MA (Moving Average), EWMA (Exponentially Weighted Moving Average) [61], ARMA (Autoregressive Moving Average) or ARIMA (Autoregressive Inte-

grated Moving Average) [62], [63] can be constructed and used to make predictions. These models, however, require measuring and analyzing real data from the environment to identify the underlying stochastic processes that generate the data.

A naive method of forecasting the next value from a history of measurements is to simply predict the next value to be the same as the current value. While simple, this method works quite well for financial and economic time series according to [64]. Another method would be to predict the next value to be the average of the previous n values in some observation period. In the absence of knowledge about the underlying stochastic processes and a statistical model, these two naive methods can be applied. These two methods, however, fail to take the trend of the measurements into consideration. The drift method is a variation of naive method, taking trend into consideration by forecasting the next value to be the current value plus the average change observed in the historical data. Using this method, the forecast for a forecast horizon h and a set of historical data collected during the time interval T is given by:

$$y_t + \frac{h}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1}) = y_T + h(\frac{y_t - y_1}{T-1}) \tag{5.1}$$

Using the drift method for forecasting is equivalent to drawing a line between the first and the last observed value and extrapolating that line into the future to make a prediction [64].

The drift forecast method will be used to predict the quality of links based on the current observation the trend of past observations.

6 Implementation

In this chapter, relevant implementation details are presented and discussed. Third party software and libraries used to implement the context reasoner are presented in Section 6.1, after which the implementation of timing functionality is discussed in the next section. Section 6.3 sheds light on how threads and synchronization and signaling between the multiple threads in this implementation were managed. In Section 6.4 a proof of concept design process of a reasoner, its membership functions and rules is described.

6.1 Software and Libraries

The proof of concept implementation was developed in C++, as per the requirements, and compiled with version 4.8.4 of the GNU gcc compiler. The development operating system was Ubuntu 14.04. Following is an overview of the external software and libraries used in the implementation.

6.1.1 Boost Libraries

The boost libraries [65] are a collection of C++libraries, simplifying the implementation of common programming tasks. The version used for this implementation is version 1.60.0. The main libraries used in the implementation of the context reasoner are the following:

Boost Threading Library

The `boost/thread` library implements functionality to facilitate thread management and synchronization between threads. Apart from thread creation and destruction, the threads library was used to implement locking of critical sections, signaling and general synchronization between threads.

The boost libraries provide scoped lock types such as `boost::unique_lock` and `boost::shared_lock` which automatically release any held mutexes in case of a thread failure, making them an optimal choice for thread safe programming.

Boost Accumulators Library

The `boost/accumulator` library offers functionality for incremental statistical computation, with the concept of accumulators at its core. Accumulators are data structures, which receive data incrementally and update their internal state after every received datum. Since measure-

ments are received one at a time by the concept reasoner, the accumulator library is an ideal choice for processing those incremental measurements. The library offers functionality for various descriptive statistical computations and also supports implementation of sliding window calculations.

Boost Chrono Library

The `boost/chrono` library provides timing functionality. Clocks with various resolutions and accuracies are offered by the library. The main concepts and data types of the chrono library are durations, time points and clocks.

- **Duration:** A duration represents an interval between two time points. Apart from the predefined durations such as `minutes`, `seconds` and `nanoseconds` one can define their own custom durations, representing the number of ticks per time unit.
- **Time point:** The time point type is used for representing a specific point in time. The library defines operations on time points and durations to simplify working with timing information.
- **Clock:** A clock type is a combination of a duration and a time point. The library offers several clocks with different resolutions and other capabilities such as monotonicity. The clock used in this implementation is the `high_resolution_clock`. It is a typedef for the clock with the highest available resolution provided by the system.

Licensing

The boost libraries' software license [66] explicitly permits commercial use of the boost libraries. The license permits derivative work and modification of the original libraries for both commercial and non-commercial use without any requirement to release the source code. Furthermore, there is no obligation to reproduce any copyright messages. When releasing own code with boost code in it, the boost license only applies to the boost code and derivatives thereof. In particular, the own source code can be licensed with any other license.

6.1.2 Fuzzylite Library

The fuzzylite [67] library is a free and open source C++ and Java library for implementing fuzzy logic controllers. It was implemented to overcome the shortcoming of similar projects, which were either very costly, had restrictive licensing or overly complex implementations [68]. The version used for the implementation is version 5.0.

Some features of the library are, support for five different types of fuzzy controllers, among them the popular Mamdani and Takagi-Sugeno controllers, 20 different linguistic term membership function shapes and the ability to create custom membership functions. Furthermore, there are 15 different t-norm and s-norm operators to choose from, among them the minimum and maximum. Apart from the commonly used centroid defuzzification method, there are six more, two of which allow weighted defuzzification. The six supported hedges, any, not, extremely, seldom, somewhat and very, give the operator finer grained control of the rules.

A commercial graphical user interface named QtFuzzyLite, which simplifies designing rule engines, can be used to design fuzzy systems with fuzzylite. Fuzzy control systems designed in

the QtFuzzyLite GUI can be exported directly to C++ code. Figure 6.1 shows the GUI in which fuzzy control systems can be designed. The figure depicts a sample fuzzy control system for a washing machine, where the amount of detergent to be used and the length of the wash cycle is determined based on the load and the amount of dirt. The linguistic input variables shown on the left side along with their membership functions are *Load* and *Dirt* and the output variables and their membership functions on the right side are *Detergent* and *Cycle*. The rules specifying the logic of the controller can be seen at the bottom.

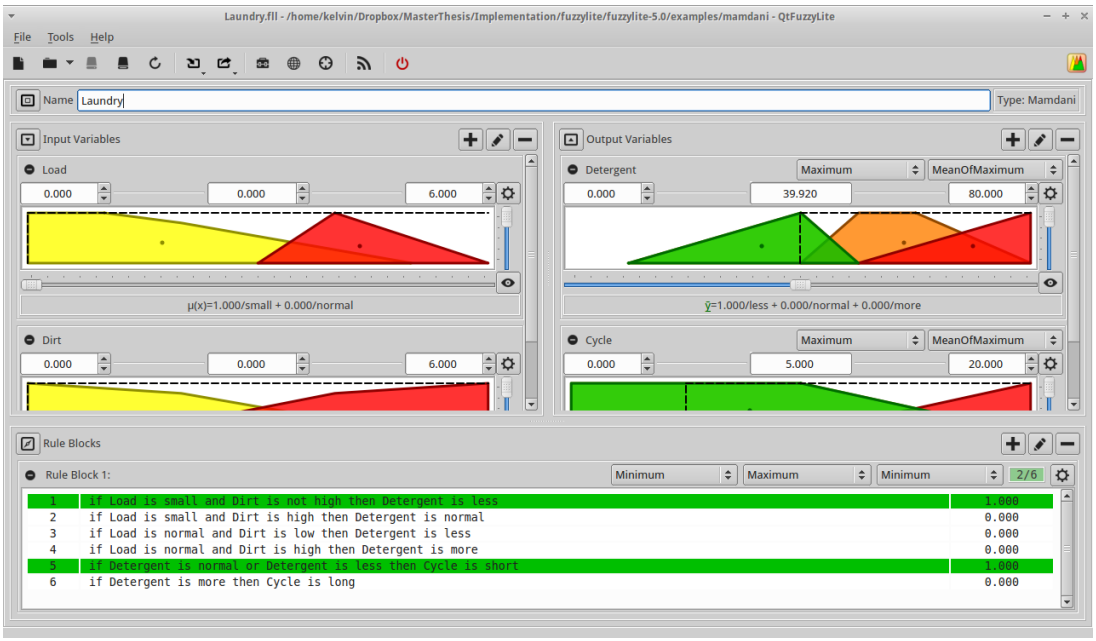

Figure 6.1: QtFuzzyLite GUI for designing fuzzy control systems

While there are other libraries for implementing fuzzy inference engines, the conclusion was reached that the fuzzylite library was the most appropriate one to use for this implementation, after researching alternative libraries.

Other libraries for implementing fuzzy systems include FuzzyClips [69], which is an extension to the CLIPS (C Language Integrated Production System) inference engine. FuzzyClips has not been maintained in over 10 years. Furthermore there is FFIS (Fast Fuzzy Inference System) [70], which has not been updated since 2013 and is poorly documented, and FisPro (Fuzzy Inference System Professional) [71] and FFLL (Free Fuzzy Logic Library) [72], which as well have not been maintained since 2013 and 2003 respectively. In [73], Cingolani et al. compare 25 different fuzzy logic libraries, many of which either did not compile, are not maintained anymore or are written in Java.

FCL (Fuzzy Control Language), FLL (Fuzzy Lite Language) and FIS (Fuzzy Inference System)

The fuzzylite library supports three different domain specific languages for designing fuzzy control systems, namely FCL, FLL and FIS. By using one of these languages, fuzzy control systems can be designed in a declarative manner and then be interpreted and compiled to C++

code by fuzzylite.

- **FCL:** FCL is a standard language for describing fuzzy control systems, published by the IEC (International Electrotechnical Commission). The syntax for FCL is specified in the document IEC 61131-7. This document is not freely available, however, and must be purchased. However, the draft of the standard is available and can be found in [74].
- **FIS**: FIS is the format implemented by Matlab used to describe fuzzy control systems. Matlabs .fis files are fully supported by fuzzylite.
- **FLL:** FLL is the default language of fuzzylite and should be used to design fuzzy systems with fuzzylite. It has been designed to be simpler and support more functionality than FCL and FIS.

Python Parser Script

The inputs and outputs, the membership functions, the rules and other settings, such as defuzzification method or inference operators, of a fuzzy inference system can all be defined in one of the languages supported by the fuzzylite software described in the above. Alternatively, the inference system can also be designed in fuzzylite's graphical user interface depicted in Figure 6.1.

Fuzzylite translates the designed fuzzy inference system to C++ code, which can then be integrated in a C++ project.

To simplify the process and the workflow of designing a fuzzy inference system in one of the fuzzy description languages and then integrating the generated code in the context reasoner project, a python script has been written. The following steps are performed by the script:

- **Parsing names:** The fuzzylite program translates the names of the input and output linguistic variables to C++ variable identifiers. Since C++ does not support reflection, these identifiers must be referred to by their names when setting inputs or extracting outputs from the reasoning engine in the C++ program.
- **Generate C++ files:** C++ code to correctly refer to the names in the context reasoner modules is generated. The generated files are header and implementation files for the fuzzy reasoner modules described in Section 5.3 as well as definitions for the FRIN and FROUT structures also mentioned in the same section.

Licensing

The fuzzylite library uses a dual-licensing model. One license is the GNU LGPL (Lesser General Public License) [GPL] and the other one is a paid commercial license. The LGPL license imposes the restriction on closed-source application of not being allowed to statically link against the library.

6.2 Timers

As mentioned in Section 5.3 the dispatcher module implements a timer to periodically initiate reasoning iterations, for which a time source is required. Furthermore, incoming measurement values have to be timestamped by the context reasoner. Since the context reasoner potentially operates in a quickly fluctuating environment and might receive measurements at high rates,

high precision timing is required to keep timing related uncertainties and imprecisions at bay.

Most modern computers offer several hardware time sources. Apart from the relatively low resolution motherboard-based clocks RTC (Real Time Clock) and the ACPI (Advanced Configuration and Power Interface) [75], [76], [77] there is the HPET (High Precision Event Timer) clock implemented in modern Intel chipsets, offering a higher resolution than the previously mentioned clocks. Querying the HPET clock is expensive, however, as it is interrupt-based [78].

The highest precision time source is TSC (Time Stamp Counter), which is a 64 bit register in each CPU core that is updated during each CPU cycle. On multiprocessor systems, the TSC should only be used as a time source if it supports a mode called invariant TSC, which guarantees that the TSC registers are synchronized across all cores and are also invariant to frequency scaling and power state changes [79].

For the implementation of the periodic timer and for timestamping incoming measurements, a type was defined that encapsulates the boost `high_resolution_clock` introduced in Section 6.1.1 and calls to read the TSC.

6.3 Threading and Synchronization

Boost Threads

As stated before, all modules run as separate threads. Threads are implemented with the help of the `boost/thread` library. Each module overloads the `()` operator, making the modules functors. When passing those module functor objects to the `boost::thread` constructor, the code defined in the overloaded `()` operator function is called by boost.

Each module thread has a while loop that checks a has boolean variable that indicates whether the thread should keep running. A signal handler is defined, capturing system signals to terminate the process. When such a signal is received, the signal handler calls the `boost/thread interrupt()` function on the threads. The `interrupt()` call throws an exception in the thread on which it is being called. The exception is handled and the boolean variable is set to false. This allows the threads to shut down gracefully rather than being stopped abruptly and do clean up work if necessary before terminating.

SharedEvent Class

To synchronize shared memory access and to signal events between modules the `SharedEvent` class was implemented. Underlying the `SharedEvent` class is a custom semaphore class implementation, facilitating synchronization between multiple threads.

The custom semaphore implementation and the `SharedEvent` class built on top of it allow one thread to act as an event dispatcher and a group of other threads as event handlers. Such an interaction is required when the dispatcher module initiates a new reasoning iteration and notifies the reasoning engines, of which there are more than one in threaded mode, as described in Section 5.3. The reasoning engines receive the notification and start a new reasoning iteration. Upon finishing, each reasoning engine must give a confirmation of having finished to the dispatcher thread. The dispatcher thread must wait for all confirmations before being able to start the next reasoning iteration.

Regarding communication between the reasoning engines and the collector module, the sit-

uation is reversed. The reasoning engines are the threads that have to notify the collector module of new data and the collector module is the thread that has to send a confirmation. The `SharedEvent` class is implemented to support this mode, where there is a group of threads that act as the dispatchers of events and single thread that acts as the handler, as well.

6.4 Proof of Concept Demonstration and Validation

This section demonstrates the proof of concept design of a fuzzy inference system for the problem addressed in this work. The rules and membership functions designed here only serve as a demonstration and should be understood as such. Following the reasoner design demonstration, the validation of the designed system is discussed shortly.

6.4.1 Proof of Concept Fuzzy Design

Designing fuzzy inference systems is not an exact science and there exists no agreed upon rigorous methodology for doing so. The human expert must encode their knowledge and expertise into the fuzzy system as they see fit and modify and adapt the rules and the membership functions by process of trial and error until the fuzzy inference system performs satisfactorily. The membership functions for the linguistic variables and fuzzy rules must be determined according to application requirements [10].

Such requirements could be an upper limit on the packet error rate or a maximum allowed packet latency. From these application requirements the output and input parameters and their membership functions and thresholds can be determined. The following output and input parameters are used in this demonstration:

Output parameters:

- **Quality:** A quality estimate taking a value between 0 and 1, 0 being the lowest quality and 1 being the highest quality.
- **Confidence:** A certainty value associated with the stated quality estimate, 0 indicating no certainty in the output and 1 indicating complete certainty.
- **Quality for a 10 second time frame:** The quality of the channel in the past 10 seconds.
- **Confidence for the 10 second time frame:** The certainty associated with the quality predictions based on the observed time frames.
- **Prediction value:** A value between 0 and 1 predicting the quality of the channel during the next time frame.

Input parameters

- **SNIR and noise:** These values are used to determine the quality of the link.
- **Percentage of values below a threshold:**. Figure 6.2 illustrates two sample distributions of measurement values. A threshold can be determined according to application require-

ments below or above which values are desirable or undesirable. The percentage of the values on the correct side of the threshold is used to determine the quality of the link in the observation period.

- **Standard deviation:** Quantifies the variability of the channel and influences the confidence of the reasoner.
- **Count:** The number of values that have been measured in the observation period.
- **Recency:** Says how old the values are.
- **Measurement uncertainty and jitter:**. Both directly affect the uncertainty of the measured values and thus the confidence of the reasoner.
- **Trend:** Indicates the trend of the values in the observation period.

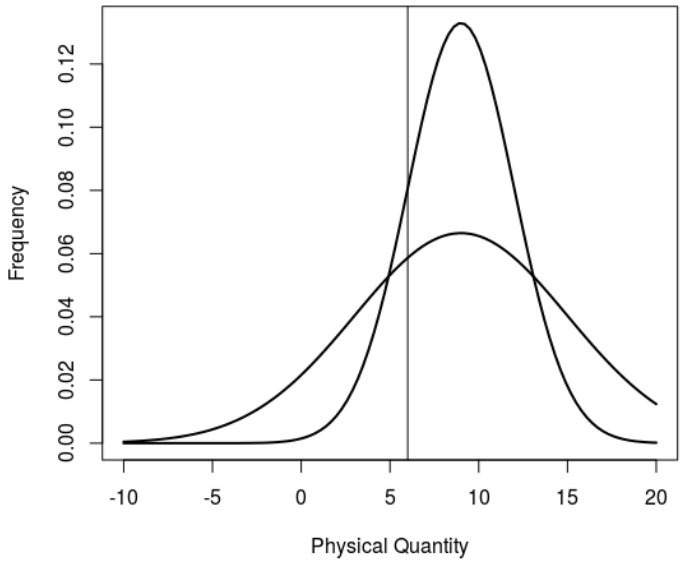

Figure 6.2: SNIR distribution and the threshold

Membership Functions and Rules

Once the requirements and the parameters have been determined, membership functions and the rules of the fuzzy inference system can be designed. The design of membership functions can be approached by first stating the domain expertise and intuitions in form of heuristics that can be applied to the problem. These same heuristics can then be used as a basis for determining the rules. Heuristics can be formulated like the following sample heuristics for the problem at hand:

- A high SNIR and low noise power should give a high quality estimate.
- A low SNIR and a high noise power should give a low quality estimate.
- A low SNIR and a low noise power value or a high SNIR and a high noise power value should give a medium quality estimate. This is because a link with a low SNIR should

not be classified as a bad link if the noise power is also low, as that link can support robust communication with the right settings.

- Unlike with the SNIR and the noise power, one low uncertainty factor should be enough to bring down the entire confidence metric of the reasoner. For instance, if measurement uncertainty is very low but there is high uncertainty associated with the recency of the values because they are all old, then the total confidence of the inference should also be low.

With the heuristics defined, the membership functions can be determined. Figure 6.3 shows sample membership functions for the linguistic variable *snir* designed in the QtFuzzyLite GUI for the proof of concept demonstration. The universe ranges from -20 to 40. The linguistic variables of this term are *low*, *medium* and *high* and correspond to the yellow, orange and red membership functions respectively. A value of around 15 dB has a membership of 0.388 in the set of the *low* linguistic term and a membership of 0.457 in the set of the *medium* term. The following assumptions were made when designing the membership functions:

- An SNIR value under 3 dB is considered low.
- An SNIR value of above 30 dB is considered high.
- An SNIR value between 10 and 30 dB is considered medium.

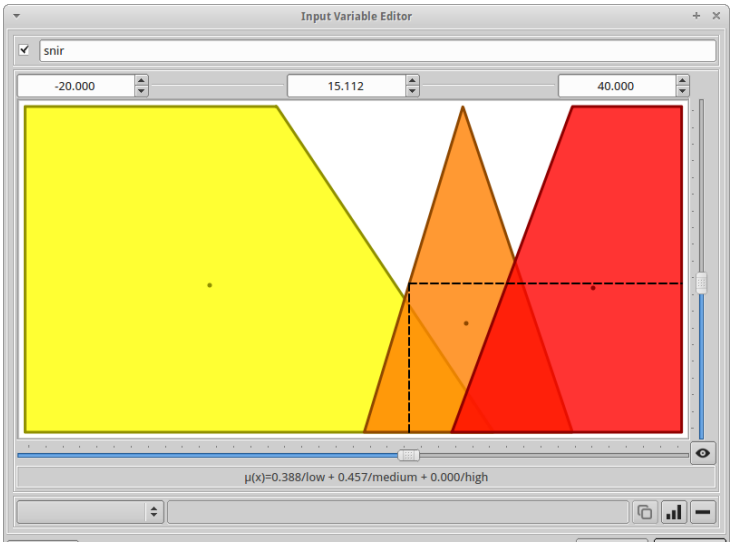

Figure 6.3: Membership functions for the *snir* linguistic variable designed in the QtFuzzyLite GUI

The rules of a fuzzy system reflect the intuition and heuristic knowledge of the system designer or domain expert. Based on the heuristics stated above, some sample rules that were determined are:

IF snir is low THEN quality is low.
IF snir is medium THEN quality is medium.
IF snir is high THEN quality is high.

IF noisepower is low THEN quality is high.

IF noisepower is medium THEN quality is medium.
IF noisepower is high THEN quality is low.

The full set of rules and the specification of the fuzzy systems can be seen in Appendix 3

6.4.2 Validation

In [80], Knauf et al. propose a framework for validation of rule-based systems. It includes generating a minimal set of test values covering the entire domain of inputs and then comparing the outputs of the rule-based system to that of a human expert. This methodology was applied to validate the proof of concept implementation.

To validate the fuzzy system, random values were generated and sent to the context reasoner as inputs. These values did not follow any particular distribution and were only used to verify that the context reasoner and the designed fuzzy inference system work correctly. Two reasoners were designed, one which delivers an output for every incoming value and one that makes inferences about a time frame of past values. The reasoners use a Mamdani type inference with the min operator as activation an conjunction and the max operator as disjunction. Defuzzification is performed with the centroid method.

Figure 6.4 depicts the generated inputs and the outputs of the fuzzy inference engine for the instant reasoner. In the first and second plot the SNIR, the noise power and the jitter inputs are shown. The third plot shows the outputs of the inference engine. The quality estimate is highest, when the SNIR is high and the noise power is low. Furthermore, the confidence in the output is low when the jitter is high and high when the jitter is low. This behavior is in accordance with the heuristics stated in Section 6.4.1.

Figure 6.5 shows the graphs for the 10 second reasoner. The outputs appear cascaded, because the engine in the 10 second reasoner repeatedly outputs a value for the same input, until its next reasoning iteration. Around the 1000th index the quality is very high, which is where both the percentage of the SNIR and the noise power are also high. The percentage value represents the amount of values that are on the desirable side of a given threshold, which is why a high noise power percentage leads to a high quality output.

The prediction value in this proof of concept demonstration only takes into account the trend of the SNIR. The similarity of their progressions indicates that the rules work correctly.

As expressed in the heuristics in the previous subsection, one uncertainty factor is enough to bring down the confidence in an inference. In this case, the high standard deviation has a significant negative impact on the confidence value of the reasoner almost throughout the measurements. At about the 500th index, where the standard deviation dips, the confidence value of the reasoner spikes, once again showing that the reasoner operates correctly.

Performance Evaluation

Since a thorough performance evaluation outside of the real environment in which the reasoner will operate is not very meaningful, none was performed. However, while validating the rules, timestamps were taken at the initiation of every reasoning period. During these measurements the context reasoner received measurements from 4 different simulated sensors and was running in threaded mode. The dispatcher module was set to initiate a reasoning period

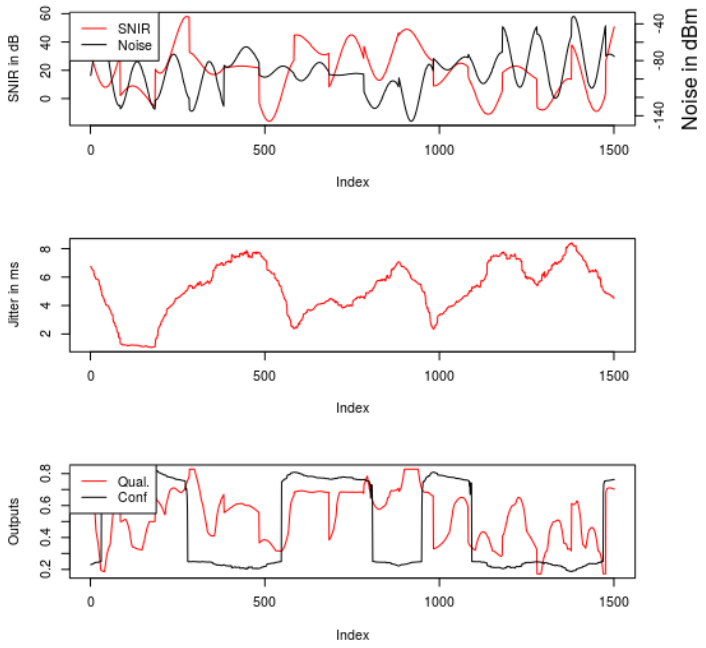

Figure 6.4: SNIR, noise power and jitter inputs and the outputs of the instant reasoner

every 10 ms. The mean achieved reasoning period was about 12.6 ms with a standard deviation of about 5 ms. Keeping in mind that these tests were run on a non-real-time operating system with scheduler preemption and other processes running, the achieved performance is satisfactory. Further tests, however, require knowledge of a target architecture on which the context reasoner shall run, to be able design sensible and meaningful test scenarios.

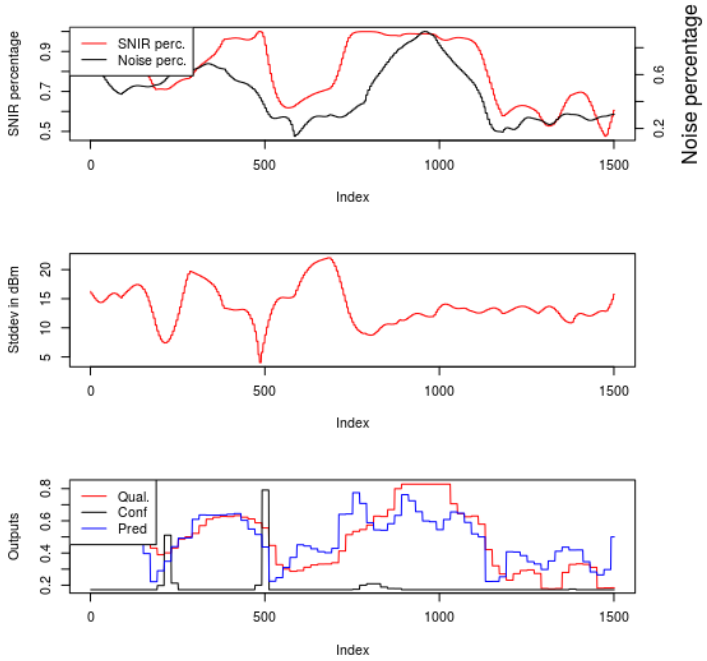

Figure 6.5: SNIR percentage, noise power percentage and standard deviation inputs and the outputs of the 10 second reasoner

7 Conclusion

This chapter concludes the thesis. Section 7.1 gives a summary of the work done and the results accomplished. Section 7.2 provides an outlook on possible future work.

7.1 Summary

In this thesis a context reasoner for dynamic spectrum access wireless communication was designed and a proof of concept implementation realized. The following work steps were taken in the course of this thesis:

- **State of the art analysis:**

  State of the art approaches to reasoning were researched and the results presented in Chapter 2. The approaches presented were rule-based reasoning, ontology-based reasoning, artificial neural networks and fuzzy reasoning. A fuzzy reasoning approach was chosen for the implementation of the context reasoner, as its capabilities were the most aligned with the objectives and requirements of this work.

- **Context space analysis:**

  A thorough analysis of the dynamic spectrum management context space was conducted, revealing complex interdependencies between the parameters, further justifying a reasoning approach rather than an algorithmic one.

- **Interface messages specified:**

  As the reasoner will operate in an environment where it receives sensing reports from radios, interface messages for communication between the radios and the reasoner were defined and integrated with existing FleMMingo interface specifications in [81].

- **Context reasoner designed:**

  The context reasoner was designed. The design was chosen to be modular and flexible, extensively making use of threads, enabling the context reasoner to handle several context sources concurrently.

- **Proof of concept implementation:**

  A proof of concept context reasoner implementation was realized. To validate the implementation, basic rule sets were demonstrated that can be used to indicate the best spectrum regarding latency and robustness and to indicate the best future spectrum based on past observations. Tests performed with synthetically generated data showed that the rules and the reasoner work correctly. Furthermore, the implementation was documented All requirements and objectives set were addressed during the design and implementa-

tion of the context reasoner.

- **Documentation:**

  The implementation was documented, enabling any future implementers to understand the inner workings of the code base.

7.2 Outlook

Implementing Interfaces and Integration in Spectrum Manager

While the interfaces and messages for communication between the context reasoner and the other entities in the spectrum manager environment were defined, they have not been implemented. As a next step, this implementation should follow, so that the context reasoner can be integrated into the spectrum manager architecture. Once this integration has occurred, the performance of the context reasoner in a real environment can be observed and tuned.

Adaptively Tuning Parameters

Furthermore, during the design of the fuzzy system, the membership functions had to be chosen as well as thresholds for where a good SNIR or noise power value starts and ends. Those thresholds are fixed and might not be optimal. The fuzzy inference system could be extended to adaptively change its membership functions to optimize the quality of its outputs. This requires some sort of feedback loop, so that the reasoning engine can learn from its previous outputs.

One way of adaptively changing the membership functions and thresholds in a fuzzy system is by combining it with an artificial neural network. One combination of artificial neural networks and fuzzy systems is ANFIS (Adaptive-Network-Based Fuzzy Inference System) [82]. ANFIS combines the learning ability of artificial neural networks with the possibility of encoding human knowledge in the reasoning engine. Apart from using a combination of neural networks and fuzzy inference to adaptively change the parameters of the fuzzy system, such a combination can also be used to increase learning speed in neural networks, by adding human knowledge to the neural network [83]. Furthermore, a combination of neural networks and a fuzzy inference system can also be used to preprocess incoming values with one of the two systems and feed the processed outputs into the second system.

Another interesting way of enhancing fuzzy inference systems is with genetic algorithms. As with neural networks, genetic algorithms can be used to tune the parameters of the fuzzy system to optimize its performance [84].

Hierarchical Fuzzy System

As the number of inputs and outputs in a fuzzy system grows, so does the number of rules. In fact, the number of possible rules in a fuzzy system increases exponentially with the number of input variables [85]. Hierarchical fuzzy systems can be used to combat the rule-explosion problem in fuzzy inference system, as the number of rules in hierarchical fuzzy systems only increase linearly with the number input variables [85]. Due to the flexible architecture of the context reasoner implemented, it could be modified to use the outputs from one fuzzy reasoner as the inputs of another one.

User Interface

So far the context reasoner is a simple console application. To make it more accessible and to simplify working with it, an HTML or some other graphical user interface could be designed and implemented. Such an interface could include the possibility of specifying and modifying rules and other aspects of the context reasoner without having to recompile and redeploy the program. Such an extension would make the context reasoner even more flexible.

.

List of Tables

4.1 Input parameters 34

1 Interface parameters and structures for reasoner output 80
2 Interface parameters and structures for setting the context reasoner settings . . . 81

List of Figures

1.1 Spectrum manager architecture within which the context reasoner is integrated . 3

2.1 Basic components of a rule-based reasoning system [12] 6
2.2 An ontology in the domain of family relationships (left) and a knowledge base of instances made from that ontology (right) [16] 8
2.3 A single neuron unit from an ANN [21] 9
2.4 Multilayer feedforward network [23] 10
2.5 Membership functions for the linguistic variable Age and the terms young, middle aged and old [30] 13
2.6 Architecture of a fuzzy inference system [31] 14
2.7 The process of fuzzy inference [32] 15

5.1 Configuration of the entities in the high level data flow architecture of the spectrum management infrastructure [55] 39
5.2 Context Reasoner modules and thread architecture 40
5.3 SAP interface sequence chart for receiving periodic sensor measurements [55] . . 45
5.4 Effect of measurement jitter on a time-varying quantity 47
5.5 Timestamp, measurement duration, reporting period and transmission time of a measurement 48
5.6 Implementing reasoning time frames with sliding windows 49

6.1 QtFuzzyLite GUI for designing fuzzy control systems 53
6.2 SNIR distribution and the threshold 57
6.3 Membership functions for the *snir* linguistic variable designed in the QtFuzzyLite GUI 58
6.4 SNIR, noise power and jitter inputs and the outputs of the instant reasoner 60
6.5 SNIR percentage, noise power percentage and standard deviation inputs and the outputs of the 10 second reasoner 61

Bibliography

[1] Nokia. *5G use cases and requirements*. White paper. Nokia. URL: http://networks.nokia.com/de/file/28771/5g-white-paper (visited on 04/11/2016).

[2] E. Dahlman et al. "5G wireless access: requirements and realization". In: *IEEE Communications Magazine* 52.12 (Dec. 2014), pp. 42–47. ISSN: 0163-6804. DOI: 10.1109/MCOM.2014.6979985.

[3] N. A. Johansson et al. "Radio access for ultra-reliable and low-latency 5G communications". In: *Communication Workshop (ICCW), 2015 IEEE International Conference on*. June 2015, pp. 1184–1189. DOI: 10.1109/ICCW.2015.7247338.

[4] R. El Hattachi. *5G White Paper*. White paper. NGMN Alliance. URL: https://www.ngmn.org/uploads/media/NGMN_5G_White_Paper_V1_0.pdf (visited on 04/11/2016).

[5] QoSMOS. *QoSMOS*. URL: http://www.ict-qosmos.eu/ (visited on 04/11/2016).

[6] Fraunhofer Fokus. *Flexible Wireless Machine to Machine Communication in Industrial Environments*. URL: https://www.fokus.fraunhofer.de/en/ngni/flemmingo_project (visited on 04/10/2016).

[7] Edmund K. Burke and Graham Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. 2nd. Springer Publishing Company, Incorporated, 2013. ISBN: 1461469392.

[8] Lui. Lam. *Nonlinear Physics for Beginners : Fractals, Chaos, Solitons, Pattern Formation, Cellular Automata, and Complex Systems*. World Scientific, 1998. ISBN: 9810201419.

[9] L. Stephanie. *QoSMOS project deliverable D2.2 – System architecture options for the QoSMOS system*. Tech. rep. Dec. 2010. URL: http://www.ict-qosmos.eu/fileadmin/documents/Dissemination/Deliverables/files/DelPub/QoSMOS_WP2_D22.pdf (visited on 04/10/2016).

[10] W. Siler. *Fuzzy Expert Systems and Fuzzy Reasoning*. Wiley, Jan. 2005. ISBN: 978-0-471-38859-3.

[11] Charles Lanny Forgy. "On the Efficient Implementation of Production Systems." AAI7919143. PhD thesis. Pittsburgh, PA, USA, 1979.

[12] Ernest Friedman Hill. *Jess in Action: Java Rule-Based Systems*. Greenwich, CT, USA: Manning Publications Co., 2003. ISBN: 1930110898.

[13] G. Jones. "Production Systems and rule-based inference". In: *Encyclopedia of cognitive science. vol.3*. 2003.

[14] Thomas R. Gruber. "A Translation Approach to Portable Ontology Specifications". In: *Knowledge Acquisition* 5.2 (June 1993), pp. 199–220. ISSN: 1042-8143. DOI: 10.1006/knac.1993.1008. URL: http://dx.doi.org/10.1006/knac.1993.1008.

[15] Natalya F. Noy. "Ontology Development 101: A Guide to Creating Your First Ontology". In: (). URL: http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html (visited on 04/03/2016).

[16] S. Cranefield. *Networked Knowledge Representation and Exchange using UML and RDF*. URL: https://journals.tdl.org/jodi/index.php/jodi/article/view/30/31 (visited on 04/06/2016).

[17] Steffen Staab and Rudi Studer. *Handbook on Ontologies*. 2nd. Springer Publishing Company, Incorporated, 2009. ISBN: 3540709991, 9783540709992.

[18] Riccardo Rosati. "On the Decidability and Complexity of Integrating Ontologies and Rules". In: *Web Semant.* 3.1 (July 2005), pp. 61–73. ISSN: 1570-8268. DOI: 10.1016/j.websem.2005.05.002. URL: http://dx.doi.org/10.1016/j.websem.2005.05.002.

[19] A. Gaignard. *Survey on semantic data stores and reasoning engines*. Paper. Sept. 14, 2010.

[20] Ian Horrocks et al. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Tech. rep. World Wide Web Consortium, May 2004.

[21] Thomas M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077, 9780070428072.

[22] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. 2nd. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN: 0132733501.

[23] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2nd ed. Pearson Education, 2003. ISBN: 0137903952.

[24] Dean A. Pomerleau. "Advances in Neural Information Processing Systems 1". In: ed. by David S. Touretzky. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989. Chap. ALVINN: An Autonomous Land Vehicle in a Neural Network, pp. 305–313. ISBN: 1-558-60015-9. URL: http://dl.acm.org/citation.cfm?id=89851.89891.

[25] R. C. Hwang et al. "The indoor positioning technique based on neural networks". In: *Signal Processing, Communications and Computing (ICSPCC), 2011 IEEE International Conference on*. Sept. 2011, pp. 1–4. DOI: 10.1109/ICSPCC.2011.6061569.

[26] A. Reda and B. Aoued. "Artificial neural network-based face recognition". In: *Control, Communications and Signal Processing, 2004. First International Symposium on*. Mar. 2004, pp. 439–442. DOI: 10.1109/ISCCSP.2004.1296323.

[27] *THE FIRST COMPUTER PROGRAM TO EVER BEAT A PROFESSIONAL PLAYER AT THE GAME OF GO*. URL: https://deepmind.com/alpha-go.html (visited on 04/06/2016).

[28] L.A. Zadeh. "Fuzzy sets". In: *Information and Control* 8.3 (1965), pp. 338–353. ISSN: 0019-9958. DOI: http://dx.doi.org/10.1016/S0019-9958(65)90241-X. URL: http://www.sciencedirect.com/science/article/pii/S001999586590241X.

[29] L. A. Zadeh. "The Concept of a Linguistic Variable and its Application to Approximate Reasoning". In: *Journal of Information Science* (1975), p. 199.

[30] *Towards machine understanding and generation of linguistic values*. URL: http://www.morenaict.com:8080/?p=709 (visited on 04/06/2016).

[31] N.M. de Reus. “Assessment of benefits and drawbacks using fuzzy logic, especially in fire control systems”. In: (Sept. 20, 1993).

[32] *Fuzzy Inference Process*. URL: http://de.mathworks.com/help/fuzzy/fuzzy-inference-process.html (visited on 04/07/2016).

[33] Bart Kosko. “Fuzzy Systems As Universal Approximators”. In: *IEEE Trans. Comput.* 43.11 (Nov. 1994), pp. 1329–1333. ISSN: 0018-9340. DOI: 10.1109/12.324566. URL: http://dx.doi.org/10.1109/12.324566.

[34] N. Serrano. “Landing Site Selection using Fuzzy Rule-Based Reasoning”. In: *IEEE International Conference on Robotics and Automation*. Ed. by N. Serrano. Apr. 10, 2007.

[35] A. Howard. *A Fuzzy Rule-Based Safety Index for Landing Site Risk Assessment*. Paper. Jet Propulsion Laboratory, 2002.

[36] A. Howard. “Multi-Sensor Terrain Classification for Safe Spacecraft Landing”. In: *IEEE TRANSACTIONS ON AEROSPACE A ELECTRONIC SYSTEMS VOL. 40, NO. 4*. Ed. by A. Howard. Oct. 2004.

[37] H. Seraji. “Behavior-Based Robot Navigation on Challenging Terrain: A Fuzzy Logic Approach”. In: *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 18, NO. 3*. Ed. by H. Seraji. June 2002.

[38] A. Howard. “A Rule-Based Fuzzy Traversability Index for Mobile Robot Navigation”. In: *Proceedings of the 2001 IEEE International Conference on Robotics 8 Automation Seoul, Korea*. Ed. by A. Howard. May 21, 2001.

[39] H. Seraji. “Terrain-Based Navigation of Planetary Rovers: A Fuzzy Logic Approach”. In: *Proceeding of the 6th International Symposium on Artificial Intelligence and Robotics and Automation in Space: i-SAIRAS 2001, Canadian Space Agency, St-Hubert, Quebec, Canada*. Ed. by H. Seraji. June 18, 2001.

[40] N. Baccour. “F-LQE: A fuzzy link quality estimator for wireless sensor networks”. In: *Proceeding EWSN’10 Proceedings of the 7th European conference on Wireless Sensor Networks Pages 240-255*. Ed. by N. Baccour. 2010.

[41] N. Baccour. “Radio Link Quality Estimation in Wireless Sensor Networks: a Survey”. In: *ACM Transactions on Sensor Networks, Vol. V, No. N, Article A*. Ed. by N. Baccour. Jan. 2011.

[42] K. Syed. “Determining Fuzzy Link Quality Membership Functions in Wireless Sensor Networks”. In: *The 9th International Conference on Future Networks and Communications (FNC’14)/The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC’14)/Affiliated Workshops, Volume 34 Pages 149-156*. Ed. by K. Syed. July 2014.

[43] Jayasri T. “Link Quality Estimation using Soft Computing Technique”. In: *Middle-East Journal of Scientific Research 21 (1): 158-168, 2014*. Ed. by Jayasri T. Jan. 21, 2014.

[44] Christian Renner et al. “Prediction Accuracy of Link-quality Estimators”. In: *Proceedings of the 8th European Conference on Wireless Sensor Networks*. EWSN’11. Bonn, Germany: Springer-Verlag, 2011, pp. 1–16. ISBN: 978-3-642-19185-5. URL: http://dl.acm.org/citation.cfm?id=1966251.1966253.

[45] Z. Jian. “Link Quality and Signal-to-Noise Ratio in 802.11 WLAN with Fading: A Time-Series Analysis”. In: (2006).

[46] D. Yeager. *What's So Special About Redline's M2M Solution?* URL: http://rdlcom.com/blog/11/422/What-s-So-Special-About-Redline-s-M2M-Solution (visited on 04/08/2016).

[47] N. Maskey, S. Horsmanheimo, and L. Tuomimäki. "Latency analysis of LTE network for M2M applications". In: *Telecommunications (ConTEL), 2015 13th International Conference on*. July 2015, pp. 1–7. DOI: 10.1109/ConTEL.2015.7231227.

[48] O. Yilmay. *5G Radio Access for Ultra-Reliable and Low-Latency Communications*. May 11, 2015. URL: http://www.ericsson.com/research-blog/5g/5g-radio-access-for-ultra-reliable-and-low-latency-communications/ (visited on 04/08/2016).

[49] C.E. Shannon. "Communication in the Presence of Noise". In: *Proc. Institute of Radio Engineers 37 (1): 10–21.* (Jan. 1949).

[50] A. Vlavianos et al. "Assessing link quality in IEEE 802.11 Wireless Networks: Which is the right metric?" In: *Personal, Indoor and Mobile Radio Communications, 2008. PIMRC 2008. IEEE 19th International Symposium on* (Sept. 15, 2008).

[51] Bluetooth SIG Regulatory Committee. *BLUETOOTH LOW ENERGY REGULATORY ASPECTS*. White paper. Bluetooth SIG Regulatory Committee, Apr. 26, 2011.

[52] ETSI, ed. *ETSI EN 300 328 V1.8.1*. June 2012.

[53] C. Demichelis and P. Chimento. *IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)*. RFC 3393. http://www.rfc-editor.org/rfc/rfc3393.txt. RFC Editor, Nov. 2002. URL: http://www.rfc-editor.org/rfc/rfc3393.txt.

[54] ITU-T. *Internet protocol data communication service – IP packet transfer and availability performance parameters*. Mar. 2011-03.

[55] B. Bochow. *Interface specification, version D0.01*. Dec. 2015-12.

[56] *ASN.1 Project*. Jan. 2, 2016. URL: http://www.itu.int/en/ITU-T/asn1/Pages/asn1_project.aspx.

[57] *Protocol Buffers*. Jan. 2, 2016. URL: https://developers.google.com/protocol-buffers/.

[58] Wonnacott and Wonnacott. *Introductory Statistics for Business and Economics, 4th Edition*. 4th. John Wiley & Sons; 4 edition (January 2, 1990), 1990.

[59] J Eidson. *IEEE-1588 Standard Version 2-A Tutorial*. Tutorial. Agilent Technologies, Oct. 2006. URL: http://www.webcitation.org/5qaJpYqCH.

[60] Texas Instruments. *AN-1728 IEEE 1588 Precision Time Protocol Time Synchronization Performance*. Report. Texas Instruments, Apr. 2013. URL: http://www.ti.com.cn/cn/lit/an/snla098a/snla098a.pdf.

[61] Zhiyuan Shi, Xueyuan Jiang, and Lianfen Huang. "A prediction approach for MAC layer sensing in cognitive radio networks". In: *Wireless, Mobile and Multimedia Networks (ICWMMN 2008), IET 2nd International Conference on*. Oct. 2008, pp. 317–320. DOI: 10.1049/cp:20081000.

[62] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990. ISBN: 0816211043.

[63] Liang Cheng and Ivan Marsic. "Lightweight Models for Prediction of Wireless Link Dynamics in Wireless/mobile Local Area Networks". In: 2002.

[64] R. Hyndman. *Forecasting: Principles and Practice*. OTexts, Oct. 17, 2013. ISBN: 0987507109.

[65] Boost C++ libraries. *Boost software license*. Accessed: 2016-04-09. 2016. URL: http://www.boost.org/.

[66] Boost C++ libraries. *Boost software license*. Accessed: 2016-03-27. 2016. URL: http://www.boost.org/users/license.html.

[67] Juan Rada-Vilela. *fuzzylite: a fuzzy logic control library*. 2014. URL: http://www.fuzzylite.com.

[68] J. Vilela. "fuzzylite: a fuzzy logic control library in C++". In: *Proceedings of the Open Source Developers Conference*. 2013.

[69] *FuzzyClips*. URL: http://awesom.eu/~quentin/archives/2010/04/22/fuzzyclips_downloads/index.html (visited on 03/31/2016).

[70] S. Rabiei. *Fast Fuzzy Inference System*. URL: https://sourceforge.net/p/fast-fis/code/ci/master/tree/ (visited on 03/31/2016).

[71] S. Guillaume. *FisPro: An open source portable software for fuzzy inference systems*. URL: https://www7.inra.fr/mia/M/fispro/fisprodocen/inline-help/fisprodocen.html (visited on 03/31/2016).

[72] S. Rabin. *Free Fuzzy Logic Library*. URL: http://ffll.sourceforge.net/ (visited on 03/31/2016).

[73] P. Cingolani. "jFuzzyLogic: A Robust and Flexible Fuzzy-Logic Inference System Language Implementation". In: *WCCI 2012 IEEE World Congress on Computational Intelligence*. June 10, 2012.

[74] International Electrotechnical Commission. *IEC 1131 - PROGRAMMABLE CONTROLLERS Part 7 - Fuzzy Control Programming*. Standard. International Electrotechnical Commission, Jan. 1997. URL: http://www.fuzzytech.com/binaries/ieccd1.pdf (visited on 03/31/2016).

[GPL] *GNU General Public License*. Version 3. Free Software Foundation, June 29, 2007. URL: http://www.gnu.org/licenses/gpl.html.

[75] NXP Semiconductor. *Enabling Timekeeping Function and Prolonging Battery Life in Low Power Systems*. NXP Semiconductor, Dec. 14, 2011. URL: https://www.maximintegrated.com/en/app-notes/index.mvp/id/1994.

[76] Maxim Integrated. *DS1375 Power Line to 60Hz Clock*. Maxim Integrated, Apr. 25, 2003. URL: https://www.maximintegrated.com/en/app-notes/index.mvp/id/1994.

[77] Daniel Bovet and Marco Cesati. *Understanding The Linux Kernel*. Oreilly & Associates Inc, 2005. ISBN: 0596005652.

[78] Intel. *IA-PC HPET (High Precision Event Timers) Specification*. Version 1.0a. Intel, Oct. 2007. URL: http://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/software-developers-hpet-spec-1-0a.pdf.

[79] Intel. *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. Manual. Intel, May 2011. URL: http://www.intel.com/Assets/en_US/PDF/manual/253668.pdf.

[80] R. Knauf, A. J. Gonzalez, and T. Abel. "A Framework for Validation of Rule-based Systems". In: *Trans. Sys. Man Cyber. Part B* 32.3 (June 2002), pp. 281–295. ISSN: 1083-4419. DOI: 10.1109/TSMCB.2002.999805. URL: http://dx.doi.org/10.1109/TSMCB.2002.999805.

[81] Bernd Bochow. *Spectrum Manager Interface specification*. Flemmingo project technical specification. Fraunhofer FOKUS, Apr. 2016.

[82] J. S. R. Jang. "ANFIS: adaptive-network-based fuzzy inference system". In: *IEEE Transactions on Systems, Man, and Cybernetics* 23.3 (May 1993), pp. 665–685. ISSN: 0018-9472. DOI: 10.1109/21.256541.

[83] H. Bothe. *Neuro-Fuzzy-Methoden: Einführung in Theorie und Anwendungen*. Springer, 1997. ISBN: 3540579664.

[84] Francisco Herrera and Luis Magdalena. *Genetic Fuzzy Systems: A Tutorial*.

[85] Li-Xin Wang. "Universal Approximation by Hierarchical Fuzzy Systems". In: *Fuzzy Sets Syst.* 93.2 (Jan. 1998), pp. 223–230. ISSN: 0165-0114. DOI: 10.1016/S0165-0114(96)00197-2. URL: http://dx.doi.org/10.1016/S0165-0114(96)00197-2.

Appendices

Appendix

1 Appendix 1: FROUT type definition

The FROUT structure type definition for the proof of concept instant reasoner and 10 second reasoner.

The fl:: namespace is the namespace of the fuzzylite library. The scalar type is simply a typedef for double or float depending on how the library was compiled. The definition of this struct reflects the output messages shown in Appendix 2

```
struct FROUT {

    uint16_t portfolioID;

    //Instant
    fl::scalar quality_value;
    fl::scalar confidence_value;
  long long timestamp_value;


    //10s
    fl::scalar quality_value_10s;
  fl::scalar confidence_value_10s;
  fl::scalar prediction_value_10s;
  long long timestamp_value_10s;



};
```

2 Appendix 2: Interface listing

The following parameters and messages structures are proposals for the interface between the context reasoner and the decision engine. Parameters such as transaction identifiers, and result codes are omitted for brevity as they can be adopted from the definitions in [81]. Primitives are assumed to be the same ones as defined in [81]. Examples are only given for SET.

Table 1: Interface parameters and structures for reasoner output

Instantaneous quality estimate based on instantaneous measurement

Name	Range	Description
Quality	[0-1]	Indicates the quality of the spectral band
Certainty value	[0-1]	Indicates the certainty of the estimate

Long term quality over a reasoning time frame. Includes prediction

Name	Range	Description
Quality	[0-1]	Indicates the quality estimate based on the observed time frame
Certainty value	[0-1]	Indicates the certainty of the inference made
Prediction value	[0-1]	Indicates the prediction estimate for the time frame

Quality output value structure

Struct {	
	Inst. quality estimate
}	Long term quality estimate [ ]

SET.rsp {	
	Quality output value structure
	Portfolio ID
}	Timestamp as defined in [81]

Much like measurement settings on the sensor instances, the context reasoner should also define adjustable settings. Following are proposed interface parameters and messages for this purpose.

Table 2: Interface parameters and structures for setting the context reasoner settings

Context reasoner settings

Name	Range	Description
Reporting period	(0-2^{32}] ms	Reporting period for the instantaneous estimates
Reasoning time frame	(0-2^{32}] ms	Length of reasoning time frame in ms
Prediction horizon	(0-2^{32}] ms	Prediction horizon in ms
Time frame reporting period	(0-2^{32}] ms	Explanation below

Time frame reporting period: When setting a reasoning time frame, a reporting period for quality estimates about that time frame should also be set. For instance, when requesting estimates about the past 10 minutes one could set the reporting period for that time frame to 2 minutes, as receiving an update about a 10 minute reasoning time frame every 10 ms would be unnecessary.

Time frame settings

Struct {	
	Reasoning time frame
	Prediction horizon
}	Time frame reporting period

Settings value structure

Struct {	
	Reporting period
}	Time frame settings [ ]

SET.rsp/req {	
}	Settings value structure

3 Appendix 3: Interface listing

FLL file for Fuzzy Reasoner for instant quality inferences:

```
1
2 Engine: FuzzyReasonerInstant
3 InputVariable: snir
4   enabled: true
5   range: -20.000 40.000
6   term: low Ramp 22.800 3.000
7   term: medium Triangle 11.000 20.000 30.000
8   term: high Ramp 19.000 30.000
9 InputVariable: noisepower
10  enabled: true
11  range: -140.000 80.000
12  term: low Ramp -90.000 -120.000
13  term: medium Triangle -126.600 -90.400 -56.400
14  term: high Ramp -67.000 20.000
15 InputVariable: meas_uncert
16  enabled: true
17  range: 0.000 1.000
18  term: low Ramp 0.300 0.050
19  term: medium Triangle 0.100 0.200 0.300
20  term: high Ramp 0.200 0.600
21 InputVariable: jitter
22  enabled: true
23  range: 0.000 10.000
24  term: low Ramp 5.100 1.000
25  term: medium Triangle 2.500 5.000 7.500
26  term: high Ramp 4.900 9.000
27 InputVariable: time_passed
28  enabled: true
29  range: 0.000 1000.000
30  term: short Ramp 510.000 100.000
31  term: medium Triangle 250.000 500.000 750.000
32  term: long Ramp 490.000 900.000
33 OutputVariable: confidence
34  enabled: true
35  range: 0.000 1.000
36  accumulation: Maximum
37  defuzzifier: Centroid 200
38  default: nan
39  lock-previous: false
40  lock-range: false
41  term: low Ramp 0.500 0.100
42  term: medium Triangle 0.250 0.500 0.750
43  term: high Ramp 0.500 0.900
44 OutputVariable: quality
45  enabled: true
46  range: 0.000 1.000
47  accumulation: Maximum
48  defuzzifier: Centroid 200
49  default: nan
```

```
  lock-previous: false
  lock-range: false
  term: low Ramp 0.500 0.100
  term: medium Triangle 0.250 0.500 0.750
  term: high Ramp 0.500 0.900
RuleBlock: snir_quality
  enabled: true
  conjunction: Minimum
  disjunction: Maximum
  activation: Minimum
  rule: if snir is low then quality is low
  rule: if snir is medium then quality is medium
  rule: if snir is high then quality is high
RuleBlock: noise_quality
  enabled: true
  conjunction: Minimum
  disjunction: Maximum
  activation: Minimum
  rule: if noisepower is low then quality is high
  rule: if noisepower is medium then quality is medium
  rule: if noisepower is high then quality is low
RuleBlock: confidence
  enabled: true
  conjunction: Minimum
  disjunction: Maximum
  activation: Minimum
  rule: if meas_uncert is high or jitter is high or time_passed is long then
      confidence is very low
  rule: if meas_uncert is low and jitter is low and time_passed is short then
      confidence is high
  rule: if meas_uncert is medium and jitter is medium and time_passed is medium
      then confidence is medium
```

FLL file for Fuzzy Reasoner for quality inferences over the last 10 seconds:

```

Engine: Reasoner10s
InputVariable: snir_percentage
  enabled: true
  range: 0.000 1.000
  term: low Ramp 0.820 0.400
  term: medium Triangle 0.560 0.710 0.860
  term: high Ramp 0.720 0.950
InputVariable: noisepower_percentage
  enabled: true
  range: 0.000 1.000
  term: low Ramp 0.750 0.170
  term: medium Triangle 0.550 0.640 0.770
  term: high Ramp 0.640 0.980
InputVariable: stddev
  enabled: true
  range: 0.000 15.000
  term: low Ramp 7.850 3.150
```

```
19   term: medium Triangle 4.900 6.950 9.600
20   term: high Ramp 7.050 11.250
21 InputVariable: time_passed
22   enabled: true
23   range: 0.000 1000.000
24   term: short Ramp 510.000 100.000
25   term: medium Triangle 250.000 500.000 750.000
26   term: long Ramp 490.000 900.000
27 InputVariable: trend
28   enabled: true
29   range: -10.000 10.000
30   term: negative Ramp 1.000 -7.000
31   term: positive Ramp -1.000 7.000
32 InputVariable: snir
33   enabled: true
34   range: -20.000 40.000
35   term: low Ramp 22.800 3.000
36   term: medium Triangle 11.000 20.000 30.000
37   term: high Ramp 19.000 30.000
38 InputVariable: cnt
39   enabled: true
40   range: 0.000 200.000
41   term: low Ramp 100.000 20.000
42   term: medium Triangle 50.000 100.000 150.000
43   term: high Ramp 100.000 180.000
44 OutputVariable: confidence
45   enabled: true
46   range: 0.000 1.000
47   accumulation: Maximum
48   defuzzifier: Centroid 200
49   default: nan
50   lock-previous: false
51   lock-range: false
52   term: low Ramp 0.500 0.100
53   term: medium Triangle 0.250 0.500 0.750
54   term: high Ramp 0.500 0.900
55 OutputVariable: quality
56   enabled: true
57   range: 0.000 1.000
58   accumulation: Maximum
59   defuzzifier: Centroid 200
60   default: nan
61   lock-previous: false
62   lock-range: false
63   term: low Ramp 0.500 0.100
64   term: medium Triangle 0.250 0.500 0.750
65   term: high Ramp 0.500 0.900
66 OutputVariable: prediction
67   enabled: true
68   range: 0.000 1.000
69   accumulation: Maximum
70   defuzzifier: Centroid 200
```

```
71   default: nan
72   lock-previous: false
73   lock-range: false
74   term: high Ramp 0.500 0.900
75   term: low Ramp 0.500 0.100
76   term: medium Triangle 0.250 0.500 0.750
77 RuleBlock: snir_quality
78   enabled: true
79   conjunction: Minimum
80   disjunction: Maximum
81   activation: Minimum
82   rule: if snir_percentage is low then quality is low
83   rule: if snir_percentage is medium then quality is medium
84   rule: if snir_percentage is high then quality is high
85 RuleBlock: noise_quality
86   enabled: true
87   conjunction: Minimum
88   disjunction: Maximum
89   activation: Minimum
90   rule: if noisepower_percentage is low then quality is low
91   rule: if noisepower_percentage is medium then quality is medium
92   rule: if noisepower_percentage is high then quality is high
93 RuleBlock: confidence
94   enabled: true
95   conjunction: Minimum
96   disjunction: Maximum
97   activation: Minimum
98   rule: if stddev is high or time_passed is long or cnt is low then confidence
       is low
99   rule: if stddev is low and time_passed is short and cnt is high then
       confidence is high
100  rule: if stddev is medium and time_passed is medium and cnt is medium then
       confidence is medium
101 RuleBlock: prediction
102  enabled: true
103  conjunction: Minimum
104  disjunction: Maximum
105  activation: Minimum
106  rule: if trend is negative and snir is low then prediction is low
107  rule: if trend is positive and snir is high then prediction is high
108  rule: if trend is positive and snir is low then prediction is medium
109  rule: if trend is negative and snir is high then prediction is medium
110  rule: if trend is positive and snir is medium then prediction is medium
```