TESIS DOCTORAL

Contribuciones al desarrollo de sistemas centrados en la calidad de servicio a través de la estimación dinámica de propiedades no funcionales

JUAN FRANCISCO INGLÉS ROMERO

PROGRAMA DE DOCTORADO DE TECNOLOGÍAS INFORMÁTICAS (TIN)

Con la conformidad de la directora

Dra. Cristina Vicente Chicote

Esta tesis cuenta con la autorización de la directora de la misma y de la Comisión Académica del programa. Dichas autorizaciones constan en el Servicio de la Escuela Internacional de Doctorado de la Universidad de Extremadura.

2023

TESIS DOCTORAL

Contributions to the development of QoS-aware systems through the runtime estimation of non-functional properties

JUAN FRANCISCO INGLÉS ROMERO

PROGRAMA DE DOCTORADO DE TECNOLOGÍAS INFORMÁTICAS (TIN)

2023

A mi mujer y a mi pequeño

Institutional acknowledgements

I would like to acknowledge the support provided by the following institutions during the completion of this Thesis:

- To Fundación Séneca Agencia de Ciencia y Tecnología de la Región de Murcia for supporting me with a research grant, 15561/FPI/10.
- To the RoQME Integrated Technical Project funded from the European Union's H2020 Research and Innovation Programme under grant agreement No. 732410, in the form of financial support to third parties of the RobMoSys Project.
- To the MIRoN Integrated Technical Project funded from the European Union's H2020 Research and Innovation Programme under grant agreement No. 732410, in the form of financial support to third parties of the RobMoSys Project.
- To the CLARC Project funded from the European Union's Seventh Framework Programme for research, technological development, and demonstration under grant agreement No. 601116, in the form of financial support to third parties of the ECHORD++ Project.

Personal acknowledgements

I am deeply grateful to everyone who, in one way or another, played a role in making this doctoral Thesis a reality:

- To Cristina Vicente Chicote, for her great dedication, patience, unconditional support, and good advice during all these years. For shaping me as a researcher (and as a person).
- To Christian Schlegel, for giving me the opportunity to carry out several research stays at the Ulm University of Applied Sciences. And together with his team (Alex, Andreas, Matthias, and Dennis), for always making me feel welcome.
- To Benoit Baudry and Jean-Marc Jézéquel, for giving me the opportunity to carry out a research stay at Institut de Recherche at Informatique et Systèmes Aléatoires (IRISA).
- To Antonio Bandera, for giving me the opportunity to carry out a research stay at University of Málaga.
- To the members of the RoQME and MIRoN projects, for the good times we spent together doing science (and other stuff). Especially to Adri and Jesús.
- To my colleagues at the Catholic University of Murcia, for their support and encouragement.
- *A mis padres, por su cariño y apoyo constantes durante toda mi vida.*
- *A mi familia, por los ánimos y todos los momentos compartidos durante la elaboración de esta tesis. En especial a todos los que me han ayudado a sacar tiempo durante las últimas semanas de ardua escritura.*
- *A mi mujer Ester y mi peque Álvaro, por ser los pilares de mi cordura emocional.*

Resumen

A medida que se incorpora más tecnología en nuestra vida diaria, crece la demanda de servicios que puedan mejorar nuestra calidad de vida y nuestras experiencias. Sin embargo, el éxito de estos servicios no se establece únicamente en base a la funcionalidad que ofrecen sino, también, y de forma cada vez más importante, considerando su Calidad de Servicio (QoS, del inglés *Quality of Service*). La calidad de servicio de un sistema software se basa en lo bien (o mal) que éste se comporta, en relación con determinadas propiedades no funcionales, como rendimiento, fiabilidad, o satisfacción del usuario, entre otras. Ser capaces de evaluar la calidad de servicio de un sistema es crucial para poder aprovechar todo el potencial de las nuevas tecnologías y, por ello, el desarrollo de sistemas centrados en la QoS es de suma importancia.

Esta Tesis tiene como objetivo contribuir al diseño e implementación de métricas de calidad de servicio a través de la estimación dinámica de propiedades no funcionales. Se propone un enfoque formal para estimar, en base a la información contextual disponible, una serie de métricas de QoS asociadas a las propiedades no funcionales consideradas de interés en cada aplicación o escenario. Para ello, se han desarrollado un conjunto de herramientas de modelado que facilitan la aplicación del enfoque propuesto, abstrayendo al usuario de toda la complejidad de su implementación. En este sentido, el enfoque propuesto busca promover el rol del Ingeniero de QoS, guiándole en el uso de las herramientas desarrolladas para que pueda especificar (en tiempo de diseño) y estimar (en tiempo de ejecución) métricas de QoS definidas sobre las propiedades no funcionales de su interés.

Abstract

As more technology is incorporated into our daily lives, the demand for services that improve our quality of life and experiences, grows. However, the success of these services is not established solely based of their functionality, but also, and increasingly importantly, considering their Quality of Service (QoS). The quality of service of a software system is related to how well it behaves in relation to certain non-functional properties, such as performance, reliability, or user satisfaction, among others. Being able to evaluate the QoS of a system is crucial to be able to make the best of new technologies and, therefore, the development of systems focused on QoS is of the utmost importance.

This Thesis aims to contribute to the design and implementation of QoS metrics through the dynamic estimation of non-functional properties. A formal approach is proposed to estimate, based on the available contextual information, a series of QoS metrics associated with the non-functional properties considered relevant in each application or scenario. In this vein, a set of modelling tools has been developed to ease the adoption of the proposed approach, abstracting the user from all the complexity of its implementation. The proposed approach seeks to promote the role of QoS Engineers, guiding them in the use of the developed tools, so that they can specify (at design time) and estimate (at runtime) QoS metrics defined on relevant non-functional properties.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

This chapter outlines the motivation behind this Thesis and provides an overview of its objectives. It also highlights the main contributions and describes the structure of the rest of the dissertation.

1.1 Motivation

As the world becomes increasingly reliant on technology, the need for "smart" services that can improve our daily lives is more pressing than ever. Whether it is robots serving as co-workers, assistants, or caregivers, or smart cities revolutionizing healthcare, transportation, and other essential services, technology has the power to create a better future for us all. However, the true measure of these services is not just what they can do, but how well they can do it. This is where Quality of Service (QoS) comes into play.

Measuring QoS is key to ensure a high quality in non-functional properties, such as performance, reliability, and satisfaction, essential to unlock the full potential of the new technologies. This is why the development of QoS-aware systems is of paramount importance. By focusing on the development of quality-aware systems, we can create a more connected, efficient, and satisfying world for everyone. This Thesis aims to contribute to this effort by exploring the design, implementation, and evaluation of QoS-aware systems through the runtime estimation of non-functional properties.

Consider the following scenario as an example to motivate this Thesis: A social robot acts as a receptionist in a hospital, providing information to visitors and conducting health surveys for new patients. It is possible to estimate how engaged people are with the robot by observing whether they approach or avoid it. Each time the robot is ignored, the perception of its ability to engage people decreases while, when someone approaches the robot, it increases. In order

to measure the robot QoS in terms of user engagement, this perception needs to be translated into a numerical score (QoS estimate). This score may be used by QoS Engineers, e.g., to check whether the robot meets the user engagement requirements, or to compare different interaction strategies and decide which one performs better. QoS estimates may also be used by the robot itself, e.g., to adapt its behaviour in order to improve the scores when they are low.

This Thesis develops a mathematical model to express this kind of dynamics and quantify them into QoS estimates, associated with non-functional properties, such as "engagement". In addition, to make the model accessible, a high-level language has been created, providing QoS Engineers with an easy-to-use modelling tool and all the required runtime infrastructure to deal with the estimation of non-functional properties.

The idea of measuring QoS based on contextual observations is not new. It has been applied to different domains, including human-robot interaction [1][2], human-machine teams [3], smart cities [4] or edge computing [5], among others. However, most approaches address QoS measurement in a limited and ad-hoc way, with little or no reuse, disregarding separation of roles (not considering specifically QoS Engineers), and with a narrow view of non-functional properties, which are usually limited to existing taxonomies. Moreover, to the best of our knowledge, it seems to be a lack of modelling frameworks focused on QoS estimation, providing a simple language, not restricted to a limited set of non-functional properties, and independent of the application domain, the software architecture, and the purpose of the QoS estimates.

1.2 Objectives

The main intended goal of this Thesis is to contribute to the development of QoS-aware systems through the runtime estimation of non-functional properties. To achieve this, the following specific objectives have been established:

- **Objective 1**: Examine the role of causality in QoS awareness, including how to formally express and reason about causal relationships.

- **Objective 2**: Design a mathematical model to estimate QoS in terms of non-functional properties and based on contextual observations.
- **Objective 3**: Identify the main modelling concepts that could be used to hide the mathematical complexity behind QoS estimations.
- **Objective 4**: Develop techniques to automatically derive the parameters of the mathematical model (Objective 2) from the modelling concepts (Objective 3).
- **Objective 5**: Define the algorithms to estimate QoS at runtime, focusing on numerical stability and efficiency.
- **Objective 6**: Develop a set of statistics on QoS estimates to provide users with additional information they can use to improve their specifications.
- **Objective 7**: Explore how the parameters of the mathematical model can be adjusted through empirical examples illustrating different contextual situations.
- **Objective 8**: Implement a Domain Specific Modelling Language and its supporting tools for specifying and executing QoS models. This objective is based on the results of objectives 1-6.
- **Objective 9**: Evaluate the proposal in real-world and simulated scenarios.

1.3 Contributions

The contributions of this Thesis are as follows:

1. A probabilistic framework that, based on contextual observations, allows estimating QoS metrics as indicators of how well the system performs in terms of relevant non-functional properties (see Chapter 3).
2. The runtime infrastructure required to support the probabilistic framework, including an exact inference algorithm for computing QoS estimates, and some relevant statistics on them (see Chapter 5).

3. The design of a neural network architecture as an alternative to the probabilistic approach, which allows refining QoS estimates with a limited number of input-output examples (see Chapter 5).
4. An abstraction of the probabilistic framework to make the specification of QoS metrics more accessible. This includes the identification of the key modelling concepts and their relationships, and the methods for translating the resulting high-level specifications into both the topology and the parameters of the underlying probabilistic model (see Chapter 4).
5. A Domain-Specific Modelling Language, a model-to-code transformation and the runtime infrastructure allowing the estimation of the QoS metrics. This contribution builds on the previous ones (see Chapter 6).

1.4 Outline

The rest of this document is organized as follows, Chapter 2 describes the research background and the state of the art; Chapter 3 identifies the key elements of the proposal and presents a formal framework to estimate QoS metrics in terms of non-functional properties; Chapter 4 develops the basis of a high-level language to hide the complexities of the formal framework; Chapter 5 focuses on the runtime estimation of the QoS metrics, including their execution, statistics, and fine-tuning based on examples; Chapter 6 details the Domain-Specific Modelling Language built on the concepts discussed in the previous chapters; Chapter 7 presents the evaluation of the proposed approach in different scenarios; and finally, Chapter 8 concludes the Thesis, including some final remarks and discussing future work.

Chapter 2 State of the art

2.1 Quality of Service

The origin of the term "Quality of Service" (QoS) can be traced back to the telecommunication industry. It was used to describe the level of service that telecom operators could provide to their customers, such as the reliability, speed, and performance of their networks and services. According to the International Telecommunications Union (ITU), QoS is “the totality of characteristics of a telecommunications service that bear on its ability to satisfy stated and implied needs of the user of the service” [6].

In this context, the QoS has been quantified using metrics such as: throughput (the amount of data transmitted over a period), latency (the time it takes for a packet to travel from source to destination), jitter (the variation in latency over time) or packet loss (the percentage of data packets that are lost in transit), among other metrics. However, with the development of the digital age and the increasing use of data and multimedia services on the Internet, the term QoS has ended up being adopted in a wide range of domains beyond telecommunications. Just to name a few examples, QoS plays a role in cloud computing, to describe the level of service that cloud providers can offer to customers in terms of resource allocation, performance, and availability [7]; in e-learning, to assess students’ experience with online platforms [8]; or in smart homes, to provide greater comfort to homeowners through the selection of the services connected to their home devices [9].

Consequently, there is a plethora of “Quality of...” terms derived from QoS, including Quality of Experience (QoE), Quality of Data (QoD), Quality of Work (QoW), Quality of Health Care (QoHC), Quality of Information (QoI), among many others. These terms expand on the concept of quality of service and provide specific means to measure and evaluate different quality factors. That is, each “Quality of...” term answers at least: what and how to measure (metrics),

and for what purpose (quality factors). For example, Quality of Experience (QoE) [10] aims to measure the user's subjective experience with a product or service, by using a number of techniques (surveys, log analysis, eye tracking, etc.) to assess quality factors such as usability, content quality, and enjoyment. In the following sections we will define the concept of metric and quality factor.

Finally, it should be noted that, in this Thesis, we will use QoS in its broadest sense, as a generic concept that encompasses the rest of the "Quality of..." terms. Therefore, we do not consider its original definition exclusively limited to the quality of the network.

2.2 Metrics

Some definitions for metrics can be found in the literature in the context of software systems [11] [12]. Based on those we will consider that a metric is a method to quantify attributes of a system, product, or service. Metrics can be used to monitor and track changes in a system over time, and to identify areas for improvement. Note that metrics and measurements are often used interchangeably, despite being distinct. Metrics are functions, while measurements are the numbers obtained from applying metrics. For example, a website might use a web analytics tool to track the number of visitors to their site over a given period of time. The metric being applied here is the number of website visitors, and the measurement is the actual count of visitors obtained through the application of the metric. The measurement can then be used to track the performance of the website and make informed decisions for optimization and growth.

Metrics can be sorted into various categories depending on the nature of the measurements. Below, we present a possible classification based on [13].

- Objective vs. subjective: Objective metrics are based on quantifiable and observable data, such as response time or error rate. These metrics are independent of personal opinions or subjective interpretations. On the contrary, subjective metrics are based on personal opinions, such as satisfaction or user experience. These metrics are influenced by personal biases and subjective interpretations.

- Quantitative vs. qualitative: Quantitative metrics come from numerical measurements, such as response time or throughput. On the other hand, qualitative metrics come from descriptive measurements that can be expressed in terms of attributes or characteristics, such as user satisfaction.
- Static vs. dynamic: Dynamic metrics evolve with time, while static metrics do not. For example, a static metric can be the storage capacity of a disk drive, and a dynamic metric the network latency.
- Absolute vs. relative: Absolute metrics are standalone measurements that can be quantified independently, while relative metrics are only meaningful when evaluated in relation to other metrics.
- Continuous vs. discrete: Continuous metrics are expressed as continuous values, such as response time or throughput. Discrete metrics are expressed as a set of discrete values, such as the number of errors or the number of successful transactions.
- Direct vs. indirect: Direct metrics are obtained by observing the property being measured directly, while indirect metrics are calculated by evaluating and analysing other data sources. For example, the delay between sending and receiving data over a network connection is an indirect metric, as it is derived by evaluating the time taken for data to be transmitted and received.

Metrics are a valuable tool for evaluating and optimizing the quality of a system. They provide a means of measuring and tracking changes in the system over time and can be used to make informed decisions about how to optimize the system. In the following, we describe some of the capabilities that metrics provide.

1. Objectives: Metrics are used to measure specific objectives, such as response time, reliability, or efficiency. The objectives of the metrics should be clearly defined, so that the metric can be evaluated against them.
2. Data collection: Metrics are collected by monitoring the system and gathering data that reflects how the system is working. This data is then analysed to determine the value of the metric.

3. Data analysis: The data collected from the system is analysed to determine the value of the metric. The analysis may involve simple calculations, such as averaging the values, or more complex statistical models, such as regression analysis.
4. Visualization: The data collected from the system can be visualized in various forms, such as charts, graphs, or tables, to help understand the behaviour of the system and the changes in the metric over time.
5. Comparison: Metrics can be compared against each other to evaluate the performance of different systems, or against established baselines or thresholds to determine whether the system is meeting performance goals.
6. Optimization: Metrics can be used to identify areas for improvement in the system. By monitoring changes over time, it is possible to identify trends and patterns, and to make informed decisions about how to optimize the system to improve its performance.
7. Collaboration: Metrics can be shared and discussed with stakeholders, such as developers, managers, and customers, to help them understand the performance of the system and to make informed decisions about how to optimize it.

2.3 Quality factors

We will consider some software engineering standards to better understand the concept of *quality factor*. This is not because this work requires adhering to a standard, but rather because standards often provide a wealth of established knowledge and commonly accepted practices.

The term quality factor was introduced in the ISO/IEC 9126-1 standard [14] and, later, in its update, the ISO/IEC 25010 [15]. According to these standards, quality factors are defined as specific aspects of a product, service, or process that contribute to its overall quality. The standards define a quality model consisting of general characteristics that are divided into sub-characteristics, which are further divided into attributes, creating a hierarchical structure with higher levels of abstraction at the top and more specific ones at the bottom. Thus, characteristics, sub-characteristics, and attributes are quality factors ordered from abstract to more concrete. At the bottom of the hierarchy, a quality factor can be measured through a metric. Then, the results are propagated upwards to determine the level of the general

characteristics at the top of the hierarchy. Figure 1 shows an example of hierarchy of quality factors.

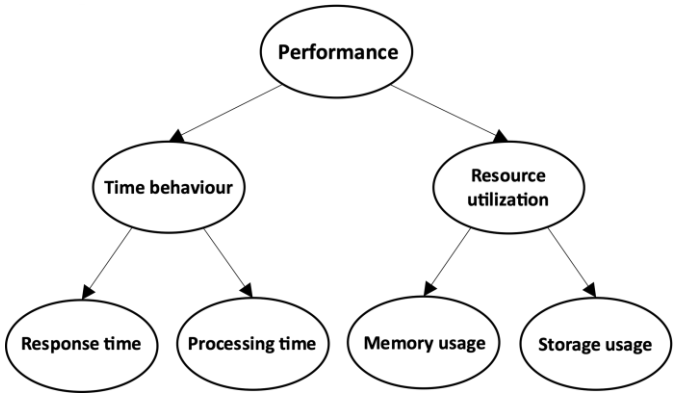


Figure 1. Example of a hierarchy of quality factors

In this Thesis we are going to use the term of *non-functional property* to refer to quality factors that describe how well the system or service performs at runtime in terms of specific criteria, such as safety, resource consumption, or usability. These properties contrast with *functional properties*, which describe what a system or service is supposed to do, in terms of specific functions or capabilities. It should be noted that, in this work, non-functional properties are quality factors of the system in operation, which should be measured at runtime under real conditions. In contrast, we will not pay attention to quality factors, such as testability and scalability, focused on the development process. In addition, contrary to its use in Requirement Engineering [16], in this context, non-functional properties are not requirements. Rather, requirements may be defined as constraints on the value of non-functional properties. Later, Section 3.2.1 will provide more details about the concept of non-functional property considered in this work.

2.4 QoS-aware systems

We define QoS awareness as the ability of a system to measure the level of quality it provides at runtime in terms of non-functional properties, such as user satisfaction, safety, or resource consumption. A QoS-aware system should at least be able to:

- *Monitor data* at runtime from the system itself, the environment, the users, or any other source of information.
- *Extract information* from the monitored data related to the quality that the system achieves and how it changes over time.
- *Estimate the level of performance* associated with high-level quality factors based on the information extracted.

QoS awareness is important in systems where the QoS requirements are subject to change, such as in systems that are deployed in dynamic or uncertain environments. By being QoS-aware at runtime, these systems can ensure that they continue to meet the needs of their users, even as those needs change. This can lead to improved user satisfaction, better system performance, and increased system reliability. In general, among the tasks of a QoS-aware system we can find:

- Assessment: Checking QoS requirements (e.g., safety rules in mobile robots [17], or safety and performance requirements in smart cities [18]). Supporting the developers' decision-making to optimize the system based on the reported quality (e.g., benchmarks for human-machine teaming [19], or evaluation of cloud computing services [20]).
- Adaptation: Dynamically adjusting the behaviour, configuration, or resources of the system in order to improve its QoS [21].
- Prediction: Anticipating future QoS requirements based on past data, and proactively adjusting the system to meet those requirements (e.g., model evolution using Bayesian estimators [22]).
- Reasoning: Using logic and knowledge representation to understand the relationship between the system behaviour and the QoS provided, and to make decisions accordingly (e.g., using Partially Observable Markov Decision Processes [23]).

2.5 Causality in QoS awareness

Establishing the causal relationships between the factors impacting QoS enables better decisions about how to control and optimize the services a system provides. In this sense,

Causality and QoS awareness are related as both are concerned with understanding the behaviour of the system. Causality can help in identifying the root cause of QoS degradation. Understanding the relationships between the different components in the system can help determine which component is causing the degradation of QoS, and then take the appropriate action to fix the problem.

However, causality may not only be relevant in the optimization of the QoS, but also in its estimation. In this Thesis, we assume that we can estimate QoS by analysing observable effects in contextual data (collected from the system, the environment, the users, etc.). Figure 2 shows a concept map that links context data with QoS.

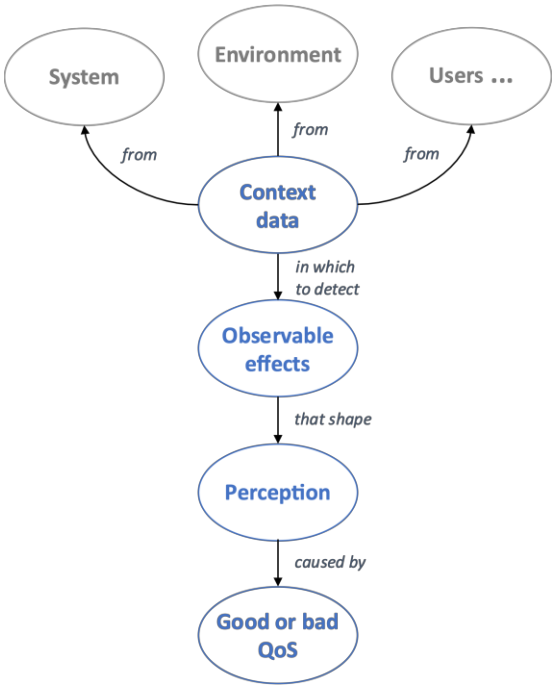


Figure 2. Link between QoS and context data

It is based on the following:

- There is a causal relationship between the QoS of a system and the perception that a domain expert has of it. That is, a poor performance of the system leads the expert to perceive it as such.
- Perception is shaped from observable effects in the context data. For example, a smoothly driving autonomous car, following speed limits and traffic signs without

accidents, will be perceived as safe. But if it suddenly drifts out of its lane into oncoming traffic, this perception will change quickly.

- The context in which a system operates is made up of data from the system itself, its environment, users, or from any other source, such as, an external web service. The actions of the system can have a direct impact on this context, and therefore on the perception of its quality. Going back to the autonomous car example, if the vehicle strays from its lane, it can raise concerns about its safety. Additionally, external factors, like weather conditions, can also affect the perception, as the speed at which a car drives is considered safe depending on whether it is raining or not.

The next section introduces formal methods to specify causal relationships on which to draw conclusions.

2.6 Causal inference

Causal inference is the process of drawing a conclusion about a causal connection based on the conditions of the occurrence of an effect. In other words, it is the process of inferring cause and effect relationships from observed data. One way to represent causality is through the use of causal inference languages. These are formal languages that are specifically designed to represent causal relationships and enable query handling. Next, we will introduce the *Structural Causal Models* (SCMs) and the *Causal Bayesian Networks* (CBNs), which are two important approaches to formally represent causal relationships and draw conclusions from the data. Note that there are other frameworks such as Pearl's do-calculus [24], Halpern's counterfactual logic [25], and Neyman-Rubin potential outcome framework [26].

2.6.1 Structural Causal Models

Structural Causal Models (SCMs) [27] are a framework for representing causal relationships using directed acyclic graphs (DAGs). In a SCM, the nodes of the graph represent variables, and the directed edges are causal relationships between those variables. The direction of the edges indicates the direction of causality, with the tail of the edge representing the cause and the head of the edge the effect.

An SCM is defined by a set of structural equations, one for each variable in the model. Each structural equation expresses the value of the variable as a function of the values of its parents (i.e., the variables that have directed edges pointing to it) and an error term. The error term represents the uncertainty or noise in the relationship, which is often assumed to be normally distributed with zero mean and a fixed variance.

SCMs can be used for a variety of tasks, such as:

- Interventions: One of the main strengths of SCMs is the ability to model the effects of interventions in a system. An intervention is a change made to one or more variables in the system. The effect of the intervention on the rest of the system can be estimated by analysing the graph structure and the causal relationships modelled by the SCM.
- Causal effect: By analysing the graph structure and the causal relationships represented by the SCM, one can estimate the causal effect of interventions on the system. This allows us to understand how changes in one variable can affect the values of other variables in the system.
- Counterfactual reasoning: SCMs can also be used to reason about counterfactuals, which are statements about what would happen in a different hypothetical scenario. For example, one can ask what would happen if a certain variable was set to a different value, or if an intervention was made to the system.
- Predictive power: By modelling the underlying mechanisms of a system, SCMs can be used to make predictions about the system's behaviour. This makes SCMs a powerful tool for analysing and predicting the behaviour of complex systems.

Some popular software packages to work with structural causal models are: Pcalg [28], Causal Explorer [29] or TETRAD [30]. There are also other libraries and packages available in widespread programming languages, such as Python [31], that provide implementations of the algorithms for working with structural causal models.

Example: Let us illustrate the above with a simple example. Figure 3 shows the DAG associated with a SCM. In this graph, nodes denote variables and edges causal relationships. Thus, we have two variables: Smoking (S) and Lung cancer (L), and a causal relationship: S

causes L. Since S has no incoming edges, it is called exogenous variable, while L is called endogenous because of the incoming edge.

Lung cancer has Smoking as a direct cause, or, in other words, the value of L depends explicitly on the values of S, that is, $L = f(S)$. Suppose that this function can be expressed with a linear model as follows: $L = \beta_0 + \beta_1 S + U$, where β_0 and β_1 are the parameters of the model and U the residual error term. In this model, the effect of being a smoker (S=1) on lung cancer is quantified by the parameters β_1, while the term β_0 would represent the baseline risk of lung cancer in the absence of smoking (S=0). Note that this is just an example, the model could be linear or not. Normally, it would be chosen based on previous experience in similar cases, trial and error, and/or the analysis of historical lung cancer data.

To estimate the parameters of the model, one could use observational or intervention data. With observational data, the parameters could be estimated using regression analysis or similar methods. With intervention data, the parameters could be estimated by manipulating S (e.g., by randomly assigning individuals to quit smoking or continue smoking) and observing the effect on L.

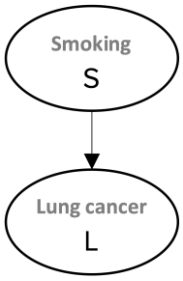


Figure 3. Relationship between smoking and lung cancer

2.6.2 Causal Bayesian Networks

Causal Bayesian Networks (CBNs) [27] are a type of Bayesian network specifically designed to represent causal relationships between variables. They are an extension of traditional Bayesian networks that allows to represent both probabilistic and causal relationships between variables.

As the SCMs, a CBN is a directed acyclic graph (DAG) where each node represents a random variable, and the edges causal relationships between these variables. Each node has a probability distribution that expresses the probability of the variable given the values of its parents in the graph. These probability distributions are represented as conditional probability tables (CPTs) or by conditional probability distributions (CPDs).

CBNs allow to model interventions, or changes in the values of certain variables, by changing the CPTs or CPDs of the affected variables. This can be used to make counterfactual predictions, that is, predictions about what would happen if a certain intervention were performed. CBNs also allow to reason about causality, that is, to determine the cause-and-effect relationships between variables. For example, we can use the CBN to infer the probability distribution of an unobserved variable given the observations of other variables. In addition, we can determine which variables are most likely to be the cause of some effect.

CBNs are widely used in various fields such as medicine [32], biology [33], or economics [34]. They are particularly useful for modelling systems where the goal is to make predictions about future observations, and to reason about causality based on a set of observations. Some popular software packages to work with CBNs are: BayesiaLab [35], GeNIe [36] and PyMC3 [37].

Example: Let us extend the example shown in the previous section. Figure 4 shows a new DAG, where now the variables are Smoking (*S*), Lung cancer (*L*) and Gene mutation (*M*). Regarding the causal relationships, a gene mutation can cause lung cancer, while smoking can cause lung cancer and a gene mutation.

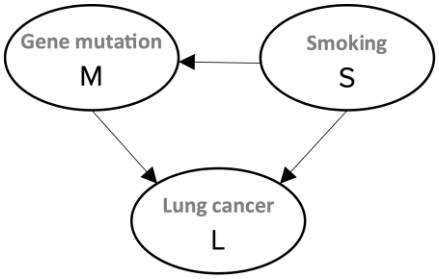


Figure 4. Relationship between smoking, gene mutation and lung cancer

Suppose that after a statistical analysis of the incidence of lung cancer in the population, we have the probabilities shown in Figure 5. Note that each of the causal relationships (edges in

the graph) is specified by a probability table, thus, $P(S)$ is the probability of smoking, $P(M|S)$ is the probability of a gene mutation conditioned by smoking, and $P(L|S,M)$ is the probability of lung cancer conditioned by smoking and a gene mutation. Variables only have two possible values, 0 (for false) and 1 (for true).

S	$P(S)$
0	0.63
1	0.37

M	S	$P(M\|S)$
0	0	0.99
0	1	0.95
1	0	0.01
1	1	0.05

L	S	M	$P(L\|S,M)$
0	0	0	0.99
0	0	1	0.97
0	1	0	0.95
0	1	1	0.8
1	0	0	0.01
1	0	1	0.03
1	1	0	0.05
1	1	1	0.2

Figure 5. Probabilities for the example

Given the observation that a person has lung cancer (L=1), we can perform inference to determine the probability of smoking (S=1). For that, by applying the chain rule of probability, we obtain the joint probability function, $P(L,S,M) = P(S)\ P(M|S)\ P(L|S,M)$. Then, we can use Bayes' theorem and marginalize to calculate $P(S=1|L=1)$, as follows:

$$P(S=1|\,L=1) = \frac{\sum_M P(L=1,S=1,M)}{\sum_S \sum_M P(L=1,S,M)} = \frac{\sum_M P(S=1)P(M|S=1)P(L=1|S=1,M)}{\sum_S \sum_M P(S)P(M|S)P(L=1|S,M)}$$

$$= \frac{0.37 \cdot 0.95 \cdot 0.05 + 0.37 \cdot 0.05 \cdot 0.2}{0.63 \cdot 0.99 \cdot 0.01 + 0.63 \cdot 0.01 \cdot 0.03 + 0.37 \cdot 0.95 \cdot 0.05 + 0.37 \cdot 0.05 \cdot 0.2} = 0.76$$

As a result, there is a 76% chance that a person with lung cancer is a smoker.

Chapter 3 Non-functional properties to measure QoS at runtime

3.1 Approach overview

The ability to observe the context and derive information about how the system performs with regard to a number of non-functional properties, such as power consumption or safety, seems to be an essential aspect for any system in a real-world environment. It is taken for granted that systems must work properly even though the environment may be complex, inherently open and show a huge number of variants and contingencies, which is not realistic unless we have systems capable of monitoring their own performance. From this information, a system could autonomously decide to take action to improve aspects in which it performs poorly, or developers can use the feedback to refine the operation of the system.

In this chapter and the next two, we discuss a probabilistic framework that allows domain experts to measure quality of service (QoS) in terms of non-functional properties. In particular, the main challenges we will face are the following:

1. Identification of the key elements involved in the estimation of non-functional properties. In regard to this Thesis, we will clarify the notion of non-functional property, QoS metric, context and observation, and how they are related to each other.
2. A formal framework to estimate QoS metrics as runtime indicators of how the system performs, according to non-functional properties and based on contextual information. We will present a probabilistic network model in which the concepts identified in 1 will be put into practice. This network will capture the causality between variables, quantified through conditional probability distributions.

3. A high-level specification to help domain experts express QoS metrics on non-functional properties, conforming to the previous probabilistic framework but without having to deal with its complexities. From this specification, a ready-to-use probabilistic network would be derived transparently.
4. The algorithms that allow the resulting probabilistic network to be executed. The estimation of the QoS metrics will have to deal with the inherent uncertainty of the context and run according to the observations detected in the environment.

3.2 Key concepts

Many concepts mentioned in this Thesis are used in a number of different research areas. Therefore, the same concept can be referred under different terms, or the same term may refer to different concepts. In this section, the definition of non-functional property, QoS metric, context and observation is clarified in the scope of this Thesis.

3.2.1 Non-functional property

Non-functional properties relate to how a system performs rather than to what it does. Examples of non-functional properties include safety, usability or resource consumption, among others. Although this concept has long been considered by the Requirement Engineering community to express the desired qualities of a system, there is still no consensus on what the term non-functional is and how can be classified and represented [38]. Consequently, there exists a plethora of varied terms (e.g., qualities, attributes, properties, characteristics or constraints) causing not only terminology, but also vast conceptual differences [38].

In this Thesis, we consider that a non-functional property is a quality factor characterized by being:

- **Abstract**. Non-functional properties cannot be assigned objective metrics (i.e., formal and precise ways to measure them) but subjective ones (i.e., ways to measure them depending on the perception of the people involved in the measurement process). For instance, if we want to measure safety or user satisfaction, there will not be a specific

method to quantify them. Rather, these non-functional properties will be formed through the abstract perception of multiple pieces of contextual evidence. The domain expert is the one who establishes what safety and user satisfaction mean and, consequently, how to interpret the contextual evidence. Therefore, through a proper modelling process, domain experts should be able to specify how contextual information influences non-functional properties based on their knowledge and experience. From this information, it should be possible to build a system capable of automatically estimating its QoS as it works, based on the context information it receives and according to the specification of the domain expert.

- **Global**. Since a non-function property is usually estimated from multiple sources of contextual information, it tends to exhibit a global and aggregating nature that presents the system as a whole.
- **Measured at runtime**. We consider that non-functional properties are quality factors of the system in operation, such as, resource consumption or effectiveness. As a result, they should be measured at runtime under real conditions. In contrast, we will not pay attention to static properties, such as testability and scalability, focused on the system development process.
- **Not limited to a catalogue**. Contrary to some efforts and standards that provide a classification, e.g. ISO/IEC 25010 [15] or the Volere taxonomy [16], in this Thesis, we consider that the specification of non-functional properties is not limited to a catalogue. While a catalogue helps establish a common ground, the idea that properties are subject to contextual evidence makes them more dependent on the application and the system ability to monitor context. As a result, the same property in two application domains could be expressed in a totally different way and have little in common. On the other hand, the absence of a catalogue provides more freedom to explore new non-functional properties.
- **Not explicitly hierarchical**. Although there may exist relationships between properties, in general, we will assume that non-functional properties are independent quality factors at the same level of abstraction. Therefore, we suppose that a property cannot be decomposed into lower-level properties that aid in its measurement. This

consideration will allow us to simplify the QoS estimation associated with non-functional properties.

- **Not focused on validating the software development process**. Quality requirements can be defined as restrictions on non-functional properties. Requirement Engineering covers all stages from the requirements elicitation to their validation and management [16], which has been shown to clearly contribute to the success of software projects. Although the concept of non-functional property has been developed in this context, in this Thesis, the main purpose of non-functional properties is not the specification of requirements but QoS awareness. In particular, non-functional properties could be considered to adapt the behaviour of the system to improve QoS, so that detecting, for example, a poor performance or a deficient resource utilization is the first step to achieve a better operation in terms of these non-functional properties.

Finally, it should be noted that throughout this Thesis we sometimes use the term "property" instead of "non-functional property" for simplicity. Unless otherwise stated, we will consider both terms to be equivalent.

3.2.2 QoS metric

We consider that a QoS metric expresses the degree of fulfilment that a system achieves with respect to a non-functional property. In other words, a QoS metric will be a score of how well the system performs according to, e.g., safety. QoS metrics are quantified as real numbers in the range [0,1] without unit of measurement, such that a value of 1 indicates that the system is optimal in terms of the property, while 0 denotes the opposite. The baseline is 0.5, that is, the default value when there is no information about the performance. Note that a QoS metric can fluctuate over time as the system operates.

Each non-functional property will have a QoS metric associated with it. The challenge will be to estimate its value through the observation of contextual information, assuming that there is a causal relationship between the perception we have of the system and the context. Therefore, by observing the effects in the context that the system and other elements of its environment produce, we should be able to quantify how well it works with respect to a certain non-

functional property. For example, if an autonomous car drives smoothly, follows speed limits and traffic signs, and has never had an accident, the system can be considered safe and the corresponding QoS metric for safety will show a high value. But the moment we see the car drifts out of the lane and invades the opposite direction, the QoS metric will drop to a much lower value. Given the abstract nature of non-functional properties, QoS metrics are subjective metrics, as they depend on how contextual information is interpreted. For this reason, we consider that they provide coarse-grained values, i.e., a small difference between estimates may not be significant. Finally, QoS estimates will be repeatable and reproducible as long as the system is exposed to the same context events.

3.2.3 Context

One of the most generally suggested definitions of context is that of Dey and Abowd [39], where they described context as "any information that can be used to characterize the situation of an entity", being an entity "a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". This definition, like others in context-awareness, revolves around the tandem user-application. Although the concept of QoS could sometimes be interpreted as an attribute of the interaction between the user and the application, the figure of the user does not always have a central role. For example, an autonomous cleaning robot can perform its tasks with a certain quality without taking into account the interaction with users. In this work, we will explicitly put the system service in the centre. As a result, we will slightly modify the above definition to "any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or object that is considered relevant in the way the system delivers its service, including the system itself".

This definition allows a wide range of possibilities regarding the types of contextual information that can be considered. On the one hand, context sources can be *external*, i.e., data may come from the physical environment (e.g., the noise level), from users (e.g., their gender and age), or from other systems and services (e.g., stock market values); or *internal*, such as the available resources or the current configuration of the system components. The level of abstraction of the context can also range from primitive data (e.g., ambient temperature) to

derived complex information (e.g., weather forecast). Finally, time is an important variable since context data often form sequences from which new insights can be drawn by looking at how they evolve over time.

3.2.4 Observation

In this work, an observation is defined as a conditional expression of one or more context variables, such that the observation occurs when the condition (aka. the pattern) is met. We specify observations to detect the effects that the system produces in the context due to a good or bad behaviour for a non-functional property. In other words, when an observation occurs it provides contextual evidence that the system is working optimally (or not) in terms of a non-functional property. Following the example of the autonomous car, we could define an observation to note when the car drifts out of the lane, so that each time it occurs, the estimate associated with safety would be negatively affected. Consequently, an observation can reinforce (increase) or undermine (decrease) the QoS estimate of a property depending on whether or not the evidence supports the idea that the system is performing optimally with respect to this property.

3.3 An example: robots in hospitals

Imagine a social robot in a huge hospital working as a receptionist to provide information and accompany visitors. One that is able to conduct health surveys to new patients or answer routine questions, such as where to buy a coffee or where visitors can park. A robot that could also help children overcome their fears by engaging in conversation and playing with them, or relieve nurses of some duties, such as delivering medicines and taking someone's temperature. Although it may sound like science fiction, these robots are becoming a reality in some hospitals [40]. Inspired by these new uses of robotics, this section presents a simple example, which will be used throughout this Thesis to illustrate the explanations.

The tasks mentioned above, i.e. providing information, conducting surveys, delivering medicines, etc, refer to the functional part of the robot, but what about the non-functional aspects? How do we know how well a robot does its job? How can we measure its service? To

answer these questions, first, we have to choose the non-functional properties under which the robot is going to be evaluated. A service can be good or bad depending on the criteria considered. For that reason, it is not the same to look at the service paying attention to power consumption or under the prism of user satisfaction. In this example, we are going to consider "user engagement" and "safety", which, like most non-functional properties, are abstract concepts with a vague interpretation. However, in this case, we will define them as the degree to which the robot is successful in producing the following desired results.

Regarding *user engagement*:

- Acceptance among patients.
- The robot manages to attract the people's interest.
- The robot achieves a high level of interaction.

Regarding *safety*:

- The robot does not bump into anyone.
- The robot is not negatively reported by hospital staff.
- The emergency stop button has not been pressed.
- The robot has not been stranded without battery.
- The robot has not received the order to stop or move away.

To quantify how "engaging" and "safe" the robot is during its operation, we will use two QoS metrics to express the degree of fulfilment that is associated with each non-functional property, represented as a number between 0 and 1. Given the subjective nature of these metrics, they should not be taken as accurate measures, but as rough indicators that help shape a global perception of the robot service. In essence, the approach adopted in this Thesis will be the following: first, we define *context variables* (e.g., battery level) and, from them, relevant *context patterns* (e.g., "the battery level drops by more than 1% per minute"). At runtime, the detection of a context pattern will trigger an *observation*, which provides contextual evidence that reinforces (or undermines) the belief that the system is optimal (or not) in terms of a *non-functional property* (e.g., power consumption). Finally, this belief will be quantified by a *QoS metric* (which, with a value of 0.93, could show a good performance). With all that, we can see how the key concepts introduced in Section 3.2 are put into practice.

Table 1 illustrates possible context variables for the hospital robot example, which are used in Table 2 and Table 3 to define some observations for *user engagement* and *safety*, respectively. For instance, the degree of engagement can be determined by observing how patients behave with the robot. In particular, the first observation in Table 2 considers a sign of acceptance when people call the robot by its name. Thus, every time this happens, that is, every time the speech recognition module produces an event of type *name* (see *speech* in Table 1), the observation will reinforce the belief that the robot is operating optimally in terms of user engagement. As for *safety*, the first observation in Table 3 establishes that the robot is not safe if it bumps into someone. So, every time a collision is detected, the observation will undermine the belief that the robot is running optimally in terms of safety. In the following sections and chapters, we will return to this example case to illustrate the explanations.

Table 1. Context variables

CONTEXT	TYPE	VALUES	SOURCE	DESCRIPTION
State	Enum.	IDLE, GREETING, SERVING, PARTING, DOCKED	Internal state of the robot	Indicates a state: (1) IDLE, the robot is not serving anyone; (2) GREETING, the robot starts talking to a person; (3) SERVING, the robot is providing information, accompanying someone, running some medical test, etc.; (4) PARTING, the person leaves; and (5) DOCKED, the robot is charging the battery
Screen	Event	--	Touch screen	Indicates the use of the touch screen, e.g. tapping a button
Speech	Event	question, name, stop, away	Based on speech recognition	Someone asks the robot, calls it by name, or orders it to stop or move away, respectively
Collision	Event	--	Based on the bumper sensors and accelerometer	Indicates that the robot has hit something or somebody
Emergency button	Event	--	Emergency button	Indicates that the emergency stop button has been pressed
Battery level	Percentage	--	Battery sensor	Indicates the robot's battery level
Report	Event	--	Reporting service	Indicates that hospital staff have negatively reported an incident with the robot
Eye contact	Boolean	--	Based on image processing	Indicates whether or not there is eye contact with the person the robot is talking to

Table 2. Observations for user engagement

#	INFLUENCE	DESCRIPTION
Acceptance among patients		
1	Reinforces	People call the robot by its name, i.e., it triggers when a *Speech* event of type *Name* is received
2	Undermines	Patients refuse to interact with the robot, i.e., it triggers when *State* transitions from GREETING to PARTING
The robot manages to attract the people's interest		
3	Undermines	The person does not hold the gaze, i.e., it triggers when the *Eye contact* value changes more than 10 times in less than 1 min while the robot is in SERVING state
The robot achieves a high level of interaction		
4	Reinforces	People use the robot's touch screen, i.e., it triggers when a *Screen* event is received
5	Reinforces	People ask questions, i.e., it triggers when a *Speech* event of type *Question* is received

Table 3. Observations for safety

#	INFLUENCE	OBSERVATION
The robot does not bump into anyone		
1	Undermines	A *Collision* event is received
The robot is not negatively reported by hospital staff		
2	Undermines	Reported negatively, i.e., it triggers when a *Report* event of type *Negative* is received
3	Reinforces	It has not been reported negatively in the last 3 days
The emergency stop button has not been pressed		
4	Undermines	An *Emergency stop* event is received
The robot has not been stranded without battery		
5	Undermines	The battery level is lower than 1% and the robot is not in the DOCKED state
The robot has not received the order to stop or move away		
6	Undermines	An event *Speech* is received of type *Stop* or *Away*

3.4 Non-functional properties through probabilistic networks

The measurement of QoS metrics involves uncertainty, given that we cannot accurately quantify how well a system works according to non-functional properties. Probabilistic networks [41] (also known as Causal Bayesian Networks, see Section 2.6.2) provide us with a formalism for expressing and reasoning with uncertain information. In this section, we will describe the probabilistic network model we propose to estimate QoS metrics on non-functional properties.

3.4.1 Design premises

The probabilistic network model has been designed in accordance with the following premises:

- We want to make explicit the causality between non-functional properties and context, in which the fact that the system behaves (or not) optimally with respect to one or more properties is partly responsible for a number of observable effects in the context.
- The process will be driven by context evidence from a number of observations, whose direction, intensity and time aspects are subject to being configured.

- The process will apply to an indefinite number of properties and support the definition of shared observations among properties, that is, observations that provide evidence for more than one property.
- The process will be independent of context variables. From the point of view of the probabilistic network, an observation simply happens or not, regardless of how this observation was defined.
- We will assume that an observation occurs independently of any other observation, that is, observations are conditionally independent of each other, the fact that one observation has occurred does not affect the probability of another observation. Since two observations could be defined from common context sources, this assumption is a simplification that might not hold. Therefore, one must be aware of this circumstance when designing the observations.
- The model will not consider explicitly the concept of counterevidence. While the occurrence of an observation can produce evidence that the system performs in a certain way, the opposite (when not occurring) does not provide counterevidence, that is, evidence that contradicts that the system is performing in that way. In fact, we assume that the absence of observations increases the uncertainty about how the system is working with regard to the considered properties.

In addition, the probabilistic network model will support observations defined with the following attributes.

- Direction of the influence. An observation can reinforce (increase) or undermine (decrease) the QoS estimate of a property depending on whether or not the evidence supports the idea that the system is performing optimally with respect to this property.
- Intensity of the influence. An observation can have a greater or lesser impact on a QoS estimate, based on whether the evidence shows more or less certainty about the performance of the system in relation to the property. For example, in terms of safety, that a robot might collide with someone seems far more serious than if it collides with a wall. As a result, it would provide stronger evidence that the robot is not performing well with regard to safety.

- Persistence time. This point refers to how an observation interacts over time. Normally, observations gradually lose their significance as time passes. As soon as an observation occurred its influence is maximum, but as time passes it weakens until it eventually disappears. In this sense, the fact that the robot used to bump into everything months ago is less important than if this were happening now.
- Repetition. Sometimes a particular condition can make an observation occurs multiple times. In that case, the influence of each occurrence will depend on the previous ones. For example, the first time a "low battery" signal is received it may not have a great effect (it could be a sensor error), but as this observation is repeated it would gain in reliability as well as in intensity, until the evidence is totally consolidated.

3.4.2 Introducing the proposed model

A probabilistic network is defined by a directed acyclic graph (DAG), in which nodes represent variables of interest and links direct causal influences between variables. The intuitive meaning of an arrow, from a node X to a node Y, is typically that X has a direct influence on Y, which suggests that causes should be parents of effects. The strength of these influences is quantified by conditional probabilities for each node given its parents in the network. The absence of a direct link indicates conditional independence. The network supports the computation of the probabilities of any subset of variables given evidence about any other subset.

Figure 6 shows the probabilistic network model we propose to estimate QoS metrics. In this network, a non-functional property, such as *User engagement*, is represented by a hidden propositional variable, denoted by $prop$ with prior probability distribution $P(prop)$. A propositional variable expresses a fact that can be true or false, in this case, the system may or may not work optimally in terms of the non-functional property, which cannot be observed directly, but rather what one has is the belief of being in a particular state based on observable pieces of evidence. The runtime quantification of this belief results in the corresponding QoS metric value. For example, a resulting value of 0.67 for *User engagement* can be understood as the probability that the system is performing optimally in terms of this property given the observed evidence.

As mentioned in Section 3.2.4, we will consider *observations* as the means to obtain contextual evidence. Using the available context variables, an observation defines a pattern whose detection supports the presence of a particular *context condition*. In the example about the use of robots in hospitals, the observation number 2 in Table 2, linked to *User engagement*, states that every time a patient leaves when the robot starts talking to him or her reinforces the "low acceptance" condition. If it has only happened once or twice, it probably does not mean anything, but as the observation occurs more times, this context condition becomes more plausible. Thus, context conditions are also hidden propositional variables (denoted by ctx_i), as they may or may not exist, which can only be inferred by looking at indirect evidence, i.e., the occurrence of observations.

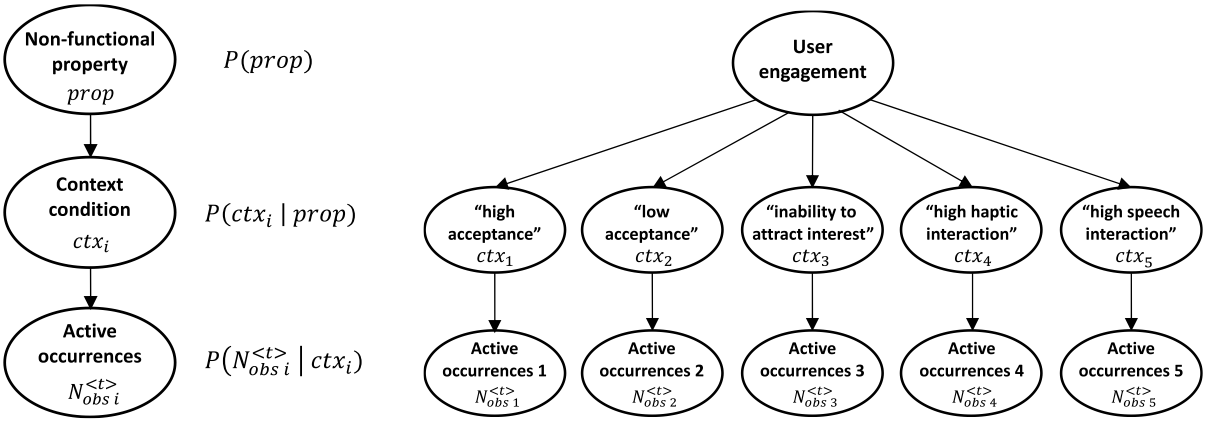


Figure 6. (Left) Proposed probabilistic network model. (Right) Example

Figure 6 shows that context conditions have as parent a non-functional property, which means that being optimal in terms of a property favours (or disfavours) certain conditions. In the example, if the robot excels at engaging with users, there will probably be "high acceptance", "high haptic interaction" and "high speech interaction", while there will be no "low acceptance" and "inability to attract interest". Put another way, if we do not know how well the robot engages with users, the presence of a context condition reinforces (or undermines) the belief that the system is acting optimally (or not) with regard to the non-functional property. The strength of this influence is quantified by the conditional probability distribution $P(ctx_i \mid prop)$.

Finally, the variable *Active occurrences* (denoted by $N_{obs\,i}^{<t>}$) indicates the average number of past occurrences (of an observation) that remain active at the time of measurement. Unlike the hidden propositional variables above, it takes a real number that is directly calculated from observable evidence. The notion "active" captures the fact that an occurrence gradually loses its relevance as time passes. To model this, we introduce the probability that a particular occurrence continues to be active after a period of time (also called survival function [42]). The moment it happens, the occurrence shows its maximum effect (i.e. this probability is 1), after that, it progressively fades until it becomes insignificant (the probability tends to 0).

The number of active occurrences provides evidence of being in a concrete context condition, so that these two nodes are linked together in the probabilistic network. See that each context condition is uniquely bound to a concrete observation pattern. Similar to what we mentioned for properties and context conditions, the number of active occurrences is an effect that appears when a certain context condition exists. Which allows us to infer, for example, the presence of the condition "high speech interaction" from the number of times that people ask the robot for information. This influence is determined by the probability distribution $P\left(N_{obs\,i}^{<t>} \mid ctx_i\right)$. It is worth noting that, although normally the notion of time is not an inherent characteristic of probabilistic networks, the fact that occurrences remain active for a period of time allows us to include timing considerations in $P(N_{obs\,i}^{<t>} \mid ctx_i)$, specifically the persistence and the repetition of the observations, as we will see in Section 4.4.4.

3.4.3 Stating the problem

Let us consider that a non-functional property is represented by the propositional variable *prop*, with values $\{PROP, \neg PROP\}$, where *PROP* means "the system performs optimally with regard to the non-functional property" and $\neg PROP$ is the negation of this statement. We want to infer *prop* given some contextual evidence, which is typified by a set of N observations, $OBS = \{obs_1, obs_2, \ldots, obs_N\}$, each one defining a particular context pattern in terms of a number of context variables. An occurrence $obs_i^{<t>}$ is produced when the pattern specified by obs_i is detected at time *t*. After that, at time $t + \tau$, $\tau \geq 0$, the occurrence may remain active with a probability expressed by $P(ACTIVE\{obs_i^{<t>}\} \mid \tau)$. Note that this probability shows its maximum value when $\tau = 0$ and decays as time passes, being $P(ACTIVE\{obs_i^{<t>}\} \mid \tau \to \infty) = 0$. The

average number of active occurrences of obs_i at time *t* is defined as the sum of the probabilities of all the occurrences, conditioned by the time elapsed since their detection, as stated below,

$$N_{obs\,i}^{<t>} = \sum_{\forall k} P(ACTIVE\{h_{obs\,i}(k)\} \mid \tau = t - t_k) \tag{3.1}$$

being $h_{obs\,i}(k)$ the current history of occurrences associated with obs_i, i.e., $h_{obs\,i}(k) = (obs_i^{<t_0>}, obs_i^{<t_1>}, \dots, obs_i^{<t_{K-1}>}, obs_i^{<t_K>})$ where $t_0 < t_1 < \cdots < t_{K-1} < t_K$.

The goal is to estimate the QoS metric that quantifies the belief that the system is optimal in terms of the non-functional property, which can be formally expressed by the following conditional probability: the probability of being optimal given, as contextual evidence, the average number of active occurrences for each observation.

$$QoS\ metric \ \equiv \ P(PROP \mid N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}) \tag{3.2}$$

According to the probabilistic network proposed in Figure 6 (left), the full joint probability distribution is expressed as follows,

$$P(prop, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}) = P(prop) \prod_{i=1}^{N} P(ctx_i \mid prop)\, P(N_{obs\,i}^{<t>} \mid ctx_i) \tag{3.3}$$

where ctx_i is a propositional variable with values $\{CTX_i, \neg CTX_i\}$, where CTX_i means "the *i*-th context condition is present" and $\neg CTX_i$ is its negation. To develop Equation (3.2), we need to consider the definition of conditional probability and the joint distribution. The result can be seen in Equation (3.4). Note that ctx_i has been marginalized out since its value is not conditioned.

$$P(PROP \mid N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}) = \frac{\sum_{ctx_i} P(PROP, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>})}{\sum_{prop} \sum_{ctx_i} P(prop, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>})} \tag{3.4}$$

Finally, taking Equation (3.3) into (3.4), we obtained the following expression, in which α is a normalization constant that ensures that the probabilities sum up to one.

$$P(PROP \mid N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}) = \alpha\, P(PROP) \prod_{\forall i} \left(\sum_{ctx_i} P(ctx_i \mid PROP)\, P(N_{obs\,i}^{<t>} \mid ctx_i) \right) \tag{3.5}$$

$$\alpha = \frac{1}{\sum_{prop} P(prop) \prod_{\forall i} \left(\sum_{ctx_i} P(ctx_i \mid prop)\, P\left(N_{obs\,i}^{<t>} \mid ctx_i\right) \right)} \tag{3.6}$$

Example. Let us apply Equation (3.5) to a simple example to clarify the notation of products and summations. The expanded form of this equation for the network in Figure 7 can be seen in Equation (3.7). Note that the product symbol multiplies all the contributions associated with context conditions, each of which takes into account the possible values for ctx_i (since the expression does not condition its value).

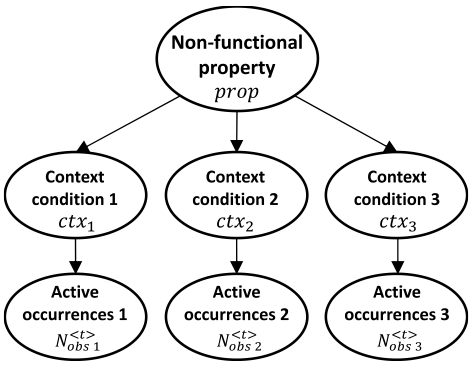


Figure 7. Probabilistic network with one property

$$\begin{aligned} QoS\ metric \equiv\ & P(PROP \mid N_{obs\,1}^{<t>}, N_{obs\,2}^{<t>}, N_{obs\,3}^{<t>}) \\ & = \alpha\, P(PROP) \cdot (P(CTX_1 \mid PROP)\, P(N_{obs\,1}^{<t>} \mid CTX_1) + P(\neg CTX_1 \mid PROP)\, P(N_{obs\,1}^{<t>} \mid \neg CTX_1)) \\ & \cdot (P(CTX_2 \mid PROP)\, P(N_{obs\,2}^{<t>} \mid CTX_2) + P(\neg CTX_2 \mid PROP)\, P(N_{obs\,2}^{<t>} \mid \neg CTX_2)) \\ & \cdot (P(CTX_3 \mid PROP)\, P(N_{obs\,3}^{<t>} \mid CTX_3) + P(\neg CTX_3 \mid PROP)\, P(N_{obs\,3}^{<t>} \mid \neg CTX_3)) \end{aligned} \tag{3.7}$$

$$\begin{aligned} \alpha = 1/[& P(PROP) \cdot (P(CTX_1 \mid PROP)\, P(N_{obs\,1}^{<t>} \mid CTX_1) + P(\neg CTX_1 \mid PROP)\, P(N_{obs\,1}^{<t>} \mid \neg CTX_1)) \\ & \cdot (P(CTX_2 \mid PROP)\, P(N_{obs\,2}^{<t>} \mid CTX_2) + P(\neg CTX_2 \mid PROP)\, P(N_{obs\,2}^{<t>} \mid \neg CTX_2)) \\ & \cdot (P(CTX_3 \mid PROP)\, P(N_{obs\,3}^{<t>} \mid CTX_3) + P(\neg CTX_3 \mid PROP)\, P(N_{obs\,3}^{<t>} \mid \neg CTX_3)) + P(\neg PROP) \\ & \cdot (P(CTX_1 \mid \neg PROP)\, P(N_{obs\,1}^{<t>} \mid CTX_1) + P(\neg CTX_1 \mid \neg PROP)\, P(N_{obs\,1}^{<t>} \mid \neg CTX_1)) \\ & \cdot (P(CTX_2 \mid \neg PROP)\, P(N_{obs\,2}^{<t>} \mid CTX_2) + P(\neg CTX_2 \mid \neg PROP)\, P(N_{obs\,2}^{<t>} \mid \neg CTX_2)) \\ & \cdot (P(CTX_3 \mid \neg PROP)\, P(N_{obs\,3}^{<t>} \mid CTX_3) + P(\neg CTX_3 \mid \neg PROP)\, P(N_{obs\,3}^{<t>} \mid \neg CTX_3))] \end{aligned} \tag{3.8}$$

3.4.4 Shared context conditions

When considering more than one property, Equation (3.5) would apply if we assume that these properties are mutually independent and have separate context conditions. In that case, the previous equation allows us to estimate the corresponding QoS metrics as if they were *M* different probabilistic networks. On the other hand, if a context condition appears as the effect of several properties, these become conditionally dependent given the shared context, which will require the introduction of some changes in the formulation.

Imagine that a context condition ctx_i produce evidence for M properties, $\{prop_1, \dots, prop_M\}$. We can conveniently see this set of properties as a single propositional variable called $props$. While we defined $prop_i \in \{PROP_i, \neg PROP_i\}$) to indicate whether the system is working optimally ($PROP_i$) or not ($\neg PROP_i$) according to a property $prop_i$, $props$ states that the system is performing optimally with respect to a specific subset of properties. Thus, the possible values of $props$ arise from all the state combinations of the properties, that is, $props$ will have 2^M possible values, $props \in \{PROPS_0, PROPS_1, \dots, PROPS_{2^M-1}\}$, where $PROPS_0$ means that the system is not running optimally for any property (i.e., $\neg PROP_M \wedge \dots \wedge \neg PROP_2 \wedge \neg PROP_1$), $PROPS_1$ means that the system is running optimally only for property $prop_1$ (i.e., $\neg PROP_M \wedge \dots \wedge \neg PROP_2 \wedge PROP_1$), and so on until $PROPS_{2^M-1}$, which means that the system is running optimally for all the properties (i.e., $PROP_M \wedge \dots \wedge PROP_2 \wedge PROP_1$). The values are ordered following the typical sequencing of a binary numeral system with symbols $\neg PROP$ (equivalent to 0) and $PROP$ (equivalent to 1), such that the index k in $PROPS_k$ is the decimal representation of the value. For instance, given 3 properties (M=3), $PROPS_5$ would correspond to $PROP_3 \wedge \neg PROP_2 \wedge PROP_1$, since the number 5 is 101 in binary.

The corresponding full joint probability distribution can be expressed as shown below. In this equation the a priori distribution of $props$ is $P(props) = \prod_{j=1}^{M} P(prop_j)$, since properties are considered independent when they are not conditioned to a shared context.

$$P(props, ctx_i, N_{obs\,i}^{<t>}) = P(props)\ P(ctx_i|props)\ P(N_{obs\,i}^{<t>}|ctx_i) \tag{3.9}$$

The QoS metric for a property $prop_j$ can be calculated by taking the full joint distribution into the definition of conditional probability, as shown in Equation (3.10). Note that the term b_i denotes the *i*-th digit of the binary representation of k.

$$QoS\ metric\ for\ prop_j \equiv P\left(PROP_j \mid N_{obs\,i}^{<t>}\right) = \frac{\sum_{\forall k \mid b_j=1} \sum_{ctx_i} P(PROPS_k, ctx_i, N_{obs\,i}^{<t>})}{\sum_k \sum_{ctx_i} P(PROPS_k, ctx_i, N_{obs\,i}^{<t>})} \tag{3.10}$$

Example. Analogously to what we did in the previous section, we will apply Equation (3.10) to the network in Figure 8 (left) to clarify the notation of products and summations. First, we will use the propositional variable *props* instead of the two property nodes, see Figure 8 (right). This variable will have 4 possible values, i.e., $props \in \{PROPS_0, PROPS_1, PROPS_2, PROPS_3\}$,

each one representing the following combinations, respectively, $(\neg PROP_2 \wedge \neg PROP_1)$, $(\neg PROP_2 \wedge PROP_1)$, $(PROP_2 \wedge \neg PROP_1)$ and $(PROP_2 \wedge PROP_1)$.

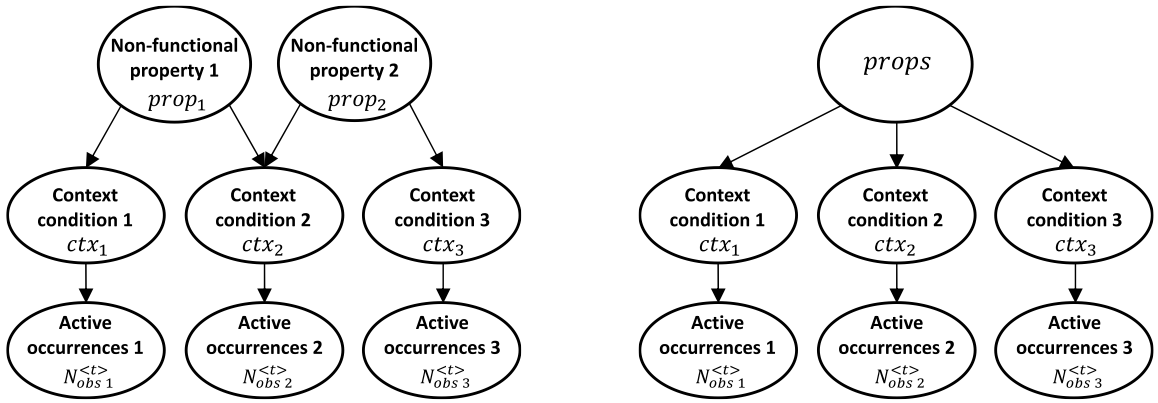


Figure 8. (Left) Two properties with shared context. (Right) Using the *props* variable

In this example we are going to calculate the QoS metric associated with the first property. Thus, being optimal in terms of $prop_1$ means that *props* must be equal to $PROPS_1$ or $PROPS_3$. Equation (3.11) shows the corresponding probability expression. Note that, in this expression, conditional probabilities that refer to non-shared context conditions can be simplified. For example, $P(ctx_1|PROPS_1) = P(ctx_1|\neg PROP_2, PROP_1) = P(ctx_1|PROP_1)$, this is so because, according to Figure 8 (left), the *Context condition 1* only depends on *Non-functional property 1*.

$$
\begin{aligned}
QoS\ metric\ for\ prop_1 &\equiv P(PROP_1 \mid N_{obs\,1}^{<t>}, N_{obs\,2}^{<t>}, N_{obs\,3}^{<t>}) \\
&= \alpha \sum_{ctx_1} \sum_{ctx_2} \sum_{ctx_3} (P(PROPS_1, ctx_1, ctx_2, ctx_3, N_{obs\,1}^{<t>}, N_{obs\,2}^{<t>}, N_{obs\,3}^{<t>}) \\
&+ P(PROPS_3, ctx_1, ctx_2, ctx_3, N_{obs\,1}^{<t>}, N_{obs\,2}^{<t>}, N_{obs\,3}^{<t>})) \\
&= \alpha\, [P(PROPS_1) \cdot (P(N_{obs\,1}^{<t>} \mid CTX_1)\, P(CTX_1 \mid PROPS_1) + P(N_{obs\,1}^{<t>} \mid \neg CTX_1)\, P(\neg CTX_1 \mid PROPS_1)) \\
&\cdot (P(N_{obs\,2}^{<t>} \mid CTX_2)\, P(CTX_2 \mid PROPS_1) + P(N_{obs\,2}^{<t>} \mid \neg CTX_2)\, P(\neg CTX_2 \mid PROPS_1)) \\
&\cdot (P(N_{obs\,3}^{<t>} \mid CTX_3)\, P(CTX_3 \mid PROPS_1) + P(N_{obs\,3}^{<t>} \mid \neg CTX_3)\, P(\neg CTX_3 \mid PROPS_1)) + P(PROPS_3) \\
&\cdot (P(N_{obs\,1}^{<t>} \mid CTX_1)\, P(CTX_1 \mid PROPS_3) + P(N_{obs\,1}^{<t>} \mid \neg CTX_1)\, P(\neg CTX_1 \mid PROPS_3)) \\
&\cdot (P(N_{obs\,2}^{<t>} \mid CTX_2)\, P(CTX_2 \mid PROPS_3) + P(N_{obs\,2}^{<t>} \mid \neg CTX_2)\, P(\neg CTX_2 \mid PROPS_3)) \\
&\cdot (P(N_{obs\,3}^{<t>} \mid CTX_3)\, P(CTX_3 \mid PROPS_3) + P(N_{obs\,3}^{<t>} \mid \neg CTX_3)\, P(\neg CTX_3 \mid PROPS_3))]
\end{aligned}
\tag{3.11}
$$

$$
\begin{aligned}
\alpha = 1/[P(PROPS_0) \cdot (P(N_{obs\,1}^{<t>} \mid CTX_1)\, P(CTX_1 \mid PROPS_0) + P(N_{obs\,1}^{<t>} \mid \neg CTX_1)\, P(\neg CTX_1 \mid PROPS_0)) \\
\cdot (P(N_{obs\,2}^{<t>} \mid CTX_2)\, P(CTX_2 \mid PROPS_0) + P(N_{obs\,2}^{<t>} \mid \neg CTX_2)\, P(\neg CTX_2 \mid PROPS_0)) \\
\cdot (P(N_{obs\,3}^{<t>} \mid CTX_3)\, P(CTX_3 \mid PROPS_0) + P(N_{obs\,3}^{<t>} \mid \neg CTX_3)\, P(\neg CTX_3 \mid PROPS_0)) + P(PROPS_1) \\
\cdot (P(N_{obs\,1}^{<t>} \mid CTX_1)\, P(CTX_1 \mid PROPS_1) + P(N_{obs\,1}^{<t>} \mid \neg CTX_1)\, P(\neg CTX_1 \mid PROPS_1)) \\
\cdot (P(N_{obs\,2}^{<t>} \mid CTX_2)\, P(CTX_2 \mid PROPS_1) + P(N_{obs\,2}^{<t>} \mid \neg CTX_2)\, P(\neg CTX_2 \mid PROPS_1)) \\
\cdot (P(N_{obs\,3}^{<t>} \mid CTX_3)\, P(CTX_3 \mid PROPS_1) + P(N_{obs\,3}^{<t>} \mid \neg CTX_3)\, P(\neg CTX_3 \mid PROPS_1)) + P(PROPS_2) \\
\cdot (P(N_{obs\,1}^{<t>} \mid CTX_1)\, P(CTX_1 \mid PROPS_2) + P(N_{obs\,1}^{<t>} \mid \neg CTX_1)\, P(\neg CTX_1 \mid PROPS_2)) \\
\cdot (P(N_{obs\,2}^{<t>} \mid CTX_2)\, P(CTX_2 \mid PROPS_2) + P(N_{obs\,2}^{<t>} \mid \neg CTX_2)\, P(\neg CTX_2 \mid PROPS_2)) \\
\cdot (P(N_{obs\,3}^{<t>} \mid CTX_3)\, P(CTX_3 \mid PROPS_2) + P(N_{obs\,3}^{<t>} \mid \neg CTX_3)\, P(\neg CTX_3 \mid PROPS_2)) + P(PROPS_3) \\
\cdot (P(N_{obs\,1}^{<t>} \mid CTX_1)\, P(CTX_1 \mid PROPS_3) + P(N_{obs\,1}^{<t>} \mid \neg CTX_1)\, P(\neg CTX_1 \mid PROPS_3)) \\
\cdot (P(N_{obs\,2}^{<t>} \mid CTX_2)\, P(CTX_2 \mid PROPS_3) + P(N_{obs\,2}^{<t>} \mid \neg CTX_2)\, P(\neg CTX_2 \mid PROPS_3)) \\
\cdot (P(N_{obs\,3}^{<t>} \mid CTX_3)\, P(CTX_3 \mid PROPS_3) + P(N_{obs\,3}^{<t>} \mid \neg CTX_3)\, P(\neg CTX_3 \mid PROPS_3))]
\end{aligned}
\tag{3.12}
$$

3.5 Summary

In this chapter, we have defined the key elements involved in the estimation of non-functional properties. Also, we presented a probabilistic framework to estimate QoS metrics as runtime indicators of how the system performs, according to non-functional properties and based on contextual information. Chapter 4 will discuss the fundamentals of a high-level specification to help domain experts express QoS metrics, conforming to the probabilistic framework but without having to deal with its complexities.

Chapter 4 Hiding probabilistic networks behind high-level descriptions

The probabilistic network proposed in Chapter 3 attempts to mimic the QoS evaluation that domain experts would subjectively perform based on their personal judgement. Thus, as domain experts specify the probabilistic network for their use case (following the model proposed in Section 3.4), they would be transferring some of their knowledge about non-functional properties to the network. Unfortunately, while it is usually easy to decide which context conditions may be relevant to some non-functional properties, specifying probabilities is challenging. Mainly because it does not fit well with the natural way people express things. It is simpler for us to give qualitative indications, such as that a certain context condition has a *strong* and *positive* influence on a property, which we would never have spontaneously defined it as a conditional probability of 0.83. To mitigate this problem, this section presents the foundations to create a high-level specification, close to our understanding, where qualitative descriptions predominate over quantitative ones. We will see how the network and their probabilities could be derived from these descriptions in a transparent way for users. Note that the notions in this chapter will form the basis of the modelling language that will be presented in Chapter 6.

4.1 Towards a high-level specification to measure quality

In accordance with the previous chapter and taking into account the design premises introduced in Section 3.4.1, Figure 9 shows the main concepts that would be included in the definition of a high-level specification to measure quality.

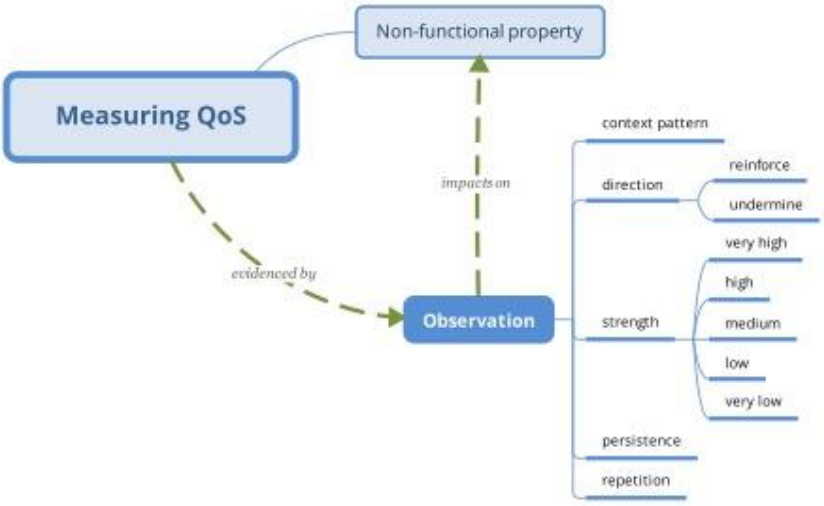


Figure 9. Concepts for the specification of QoS concerns

The elements of the above concept map are described next.

- *Non-functional property*. Measuring QoS revolves around the estimation of how well the system works in terms of one or more non-functional properties. Implicitly, each property will be associated with a QoS metric that quantifies the performance of the system for the given property.
- *Observation*, it provides evidence that the system behaviour is optimal (or not) in relation to a non-functional property. An observation can be defined considering the following aspects.
    - *Context pattern*, it identifies a condition expressed in terms of one or more context variables. The detection of this pattern produces an occurrence (an event), which influences the value of a QoS metric associated with a property.
    - *Direction*, it indicates the orientation of the evidence, it may reinforce (increase) or undermine (decrease) the belief that the system performs optimally.
    - *Strength*, it qualitatively determines the intensity of the evidence considering five different levels: from “very low” to “very high”.
    - *Persistence*, it specifies the maximum period of time in which the effect of an occurrence remains with some influence until it disappears.
    - *Repetition*, it indicates how many times an observation must be repeated to have full effect. By defining this parameter, the effect of an occurrence will depend

on the occurrences detected just before rather than taking them as independent observations. For instance, it could be useful to handle a “low battery” condition. The first time "low battery" is detected it may not have a great effect (it could be a sensor error), but as this observation is repeated it would gain in reliability as well as in intensity. The default value of this parameter is 0 (no repetitions are expected).

Note that *persistence* will require the specification of a time value, which could be provided in several ways. For example, we can opt for precision ("13 seconds"), for the convenience of expressing it linguistically ("a few seconds"), or for the simplicity of a relative value ("a short time"). Figure 10 shows the concepts we propose for the specification of time values.

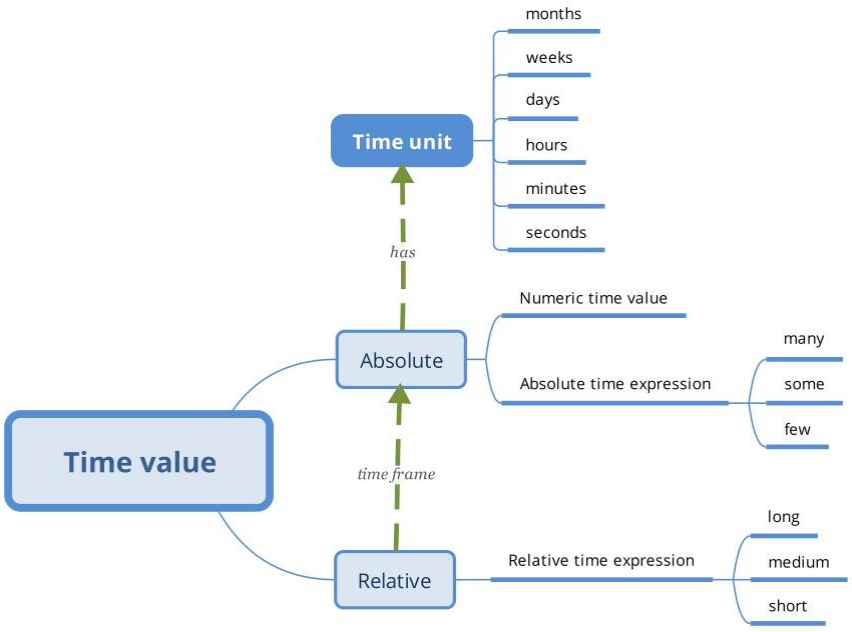


Figure 10. Concepts for the specification of time values

Next, the elements of the above concept map are described.

- *Absolute time value*, it introduces the notion of time with a certain unit. This work will take into account the following time units: seconds, minutes, hour, days, weeks and months.
- *Numeric time value*, it denotes the use of time values expressed numerically, such as “13 seconds”.

- *Absolute time expression*, it represents linguistic expressions that explicitly include the time unit, such as "a few seconds", "some minutes" or "many hours". In this work, we will consider three different quantifiers: *few*, *some* and *many*.
- *Relative time value*, it introduces the notion of time relative to a reference period (called *time frame*). The time frame should be expressed as an absolute time value.
- *Relative time expression*, it identifies relative linguistic time expressions. For example, considering a time frame of one hour, this concept allows the use of the expression "a short time" as a way of saying "a few minutes". But if the time frame had been one month, probably the meaning of "a short time" would have been closer to "a few days". In this work, we will consider three different quantifiers: *short*, *medium* and *long*.

Table 4 shows an example of high-level specification for the scenario described in Section 3.3. Note that in this example some time values were defined using relative expressions with a 7-day time frame. We selected this time frame because a week seems long enough for the robot to show all its functionality. In addition, it could be reasonable to make this time frame coincide with the robot maintenance time, so let us imagine that the robot is checked regularly every week, in which the QoS estimates collected during this period can be reviewed as part of the maintenance tasks.

Table 4. High-level specification for the hospital robot

PROPERTY	**User engagement**	
OBSERVATION	"People call the robot by its name"	
(PATTERN) The robot's name is recognized	(PERSISTENCE) Some hours	(REPETITION) 5
(PROPERTY) User engagement	(DIRECTION) Reinforce	(STRENGTH) Low
NOTE: If this observation is repeated at least 5 times, the evidence will be consolidated. Moreover, the influence is low as it is a small gesture of acceptance.		
OBSERVATION	"Patients refuse to interact with the robot"	
(PATTERN) People leaves when the robot greets them	(PERSISTENCE) Medium (7-day timeframe)	(REPETITION) 5
(PROPERTY) User engagement	(DIRECTION) Undermine	(STRENGTH) Low
NOTE: If this observation is repeated at least 5 times, the evidence will be consolidated. Moreover, the influence is low because this reaction could be caused by reasons unrelated to the robot's performance (e.g., the users' fear of the new).		
OBSERVATION	"People lose interest"	
(PATTERN) People do not hold the gaze	(PERSISTENCE) Few hours	(REPETITION) None
(PROPERTY) User engagement	(DIRECTION) Undermine	(STRENGTH) Medium
NOTE: Users can lose interest if the robot follows tedious procedures or there are long wait times.		

OBSERVATION	"People interact through the robot's touch screen"	
(PATTERN) A screen event is received	(PERSISTENCE) Short (7-day timeframe)	(REPETITION) None
(PROPERTY) User engagement	(DIRECTION) Reinforce	(STRENGTH) High
NOTE: It is a positive sign of interaction with the robot. It is considered important (high influence), and the persistence is short as it is expected to occur frequently.		

OBSERVATION	"People ask questions"	
(PATTERN) A question is recognized	(PERSISTENCE) Short (7-day timeframe)	(REPETITION) None
(PROPERTY) User engagement	(DIRECTION) Reinforce	(STRENGTH) High
NOTE: As the previous observation, it is a positive sign of interaction with the robot. It is considered important (high influence), and the persistence is short as it is expected to occur frequently.		

PROPERTY	**Safety**

OBSERVATION	"Bump into someone"	
(PATTERN) A collision event is received	(PERSISTENCE) Long (7-day timeframe)	(REPETITION) None
(PROPERTY) Safety	(DIRECTION) Undermine	(STRENGTH) Very high
NOTE: In relation to safety, a collision represents a critical circumstance that greatly undermines the belief that the robot is performing optimally. Given its importance, its effect has a long persistence.		

OBSERVATION	"Negatively reported"	
(PATTERN) A negative report event is received	(PERSISTENCE) Long (7-day timeframe)	(REPETITION) None
(PROPERTY) Safety	(DIRECTION) Undermine	(STRENGTH) High
NOTE: A negative report from the hospital staff on safety issues is considered serious and has long persistence.		

OBSERVATION	"Emergency stop pressed"	
(PATTERN) Emergency stop event is received	(PERSISTENCE) Medium (7-day timeframe)	(REPETITION) None
(PROPERTY) Safety	(DIRECTION) Undermine	(STRENGTH) Medium
NOTE: Pressing the emergency stop seems less severe than the previous observations since it can be activated for reasons other than safety.		

OBSERVATION	"Not reported"	
(PATTERN) It has not been reported negatively in the last 3 days	(PERSISTENCE) Some days	(REPETITION) None
(PROPERTY) Safety	(DIRECTION) Reinforce	(STRENGTH) High
NOTE: Persistence was set to fit the 3-day period defined by the pattern. We do not set repetition to treat occurrences separately.		

OBSERVATION	"Stranded without battery"	
(PATTERN) The battery level is lower than 1% and not in DOCKED state	(PERSISTENCE) Medium (7-day timeframe)	(REPETITION) 5
(PROPERTY) Safety	(DIRECTION) Undermine	(STRENGTH) High
NOTE: Once the robot reaches this situation, the observation is repeated periodically every few minutes. The situation is consolidated after 5 repetitions of the observation.		

OBSERVATION	"Commanded to stop"	
(PATTERN) An event Speech is received of type Stop or Away	(PERSISTENCE) Short (7-day timeframe)	(REPETITION) None
(PROPERTY) Safety	(DIRECTION) Undermine	(STRENGTH) Low
NOTE: The strength is low since the speech recognition can produce false results in real noisy environments.		

In the following sections we will see how the underlying probabilistic network can be derived transparently from this high-level specification. As a result, domain experts will be able to focus on their use cases without having to deal with the complexities of probability distributions or dependencies among variables.

4.2 Deriving the network topology

By taking into account the model presented in Section 3.4, it is straightforward to derive the constituent parts of the probabilistic network from a high-level specification. In essence, the two basic rules to obtain the probabilistic network are:

1. A *Property* generates a *Non-functional property* node.
2. An *Observation* produces two nodes: A *Context condition* linked to an *Active occurrence*, where the former has as parents the *Non-functional property* nodes on which the observation impacts.

Figure 11 illustrates the application of these rules.

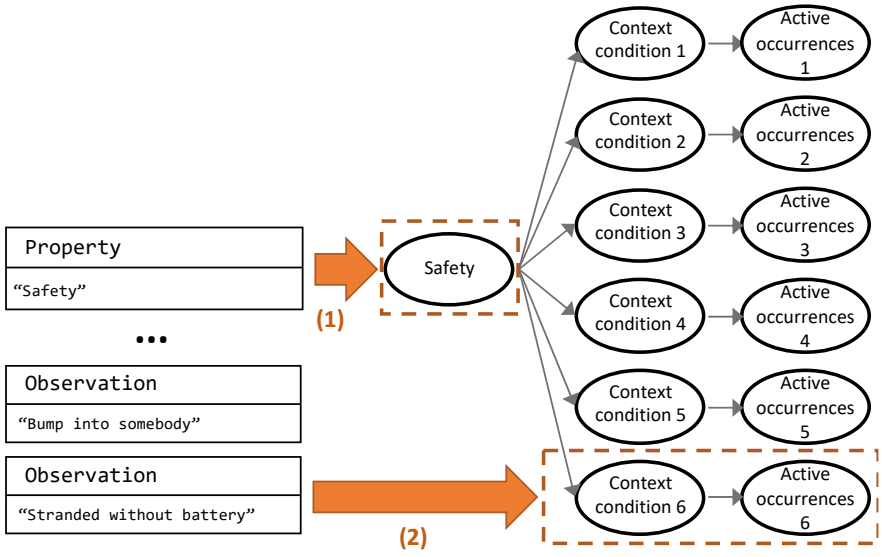


Figure 11. Illustration of how to derive of the network topology

When we have several properties, it is convenient, before applying these rules, to identify groups of dependent properties in order to segment the specification into multiple networks. Potentially, it will reduce the computational cost since the estimates would be calculated over

several smaller probabilistic networks (rather than over a single larger and more intricate network). In this sense, two properties are considered dependent if there is an observation that has an impact on both (i.e., the properties share a common context condition as we saw in Section 3.4.4).

Regarding the method we could use to identify dependencies among properties, first, let us denote C as a collection of non-empty subsets of dependent properties, where their union represents the total set of properties and the intersection between subsets is always empty (a property can only be in a single subset). To sort properties into subsets in C we could proceed as follows.

1. At first, C is empty.
2. We go through all the observations. For each one, let us consider A as the subset of properties on which the observation impacts,
    2.1 Given any subset B in C, if the intersection $A \cap B \neq \emptyset$, we update A as the union $A \cup B$ and remove B from C
    2.2 We add A as a new subset of C

Finally, once the previous method has been carried out, we will generate a separate probabilistic network from each subset of dependent properties in C, so that, if we want to obtain estimates for a given property, we will only need to process the corresponding network in which that property is defined.

4.3 Deriving time values

The concept map presented in Figure 10 lays the foundations for expressing time values using qualitative quantifiers. While these bring specifications closer to the natural way users understand things, we need a method to transparently derive numerical values from this information to apply them to the underlying probabilistic network. Next, we propose a method based on the Weber-Fechner Law.

4.3.1 The Weber-Fechner Law

The Weber-Fechner Law (WFL) [43] is an important principle in psychophysics that describes the relationship between the magnitude of a physical stimulus and the intensity perceived by a person. This relationship has been experimentally validated for a broad range of scenarios, like human vision, hearing, touching, or time perception, and is considered today as one of the most fundamental laws of the psychology of perception. According to the WFL, the differential perception *dP* is directly proportional to the relative change *dS/S* of the physical stimulus:

$$dP = k\,\frac{dS}{S} \tag{4.1}$$

with S as the current magnitude of the stimulus, while the constant k is up to experimental determination. Straightforward integration of (4.1) leads to (4.2), where P identifies the magnitude of the perception, and the constant of integration S_0 can be interpreted as the stimulus threshold (below which no sensory perception is possible at all).

$$P = k\,ln\frac{S}{S_0} \tag{4.2}$$

4.3.2 Absolute time expressions

Absolute time expressions were presented in the concept map proposed in Figure 10. According to this concept map, we need two attributes to define an absolute time expression: *quantity* and *unit*, the former is constructed using one of the following quantifiers: *few*, *some* and *many*, while the latter indicates the unit of measurement. As a result, we can produce expressions, such as “few seconds” or “some minutes”. To derive numerical values from these, we assume that when we use the term “seconds”, “minutes”, “hours”, etc., it represents a disjointed period of time, greater than or equal to 2 (since we can only form the plural with at least 2 units) and strictly less than the lower limit of the next higher unit of measurement. For example, we suppose that when we speak of seconds, whether few, some or many, we refer to a value in the interval [2, 120) (i.e., from 2s to 1min and 59s). Figure 12 illustrates the corresponding intervals for the considered units of measurement. Note that we have conveniently assumed that a month has 4 weeks.

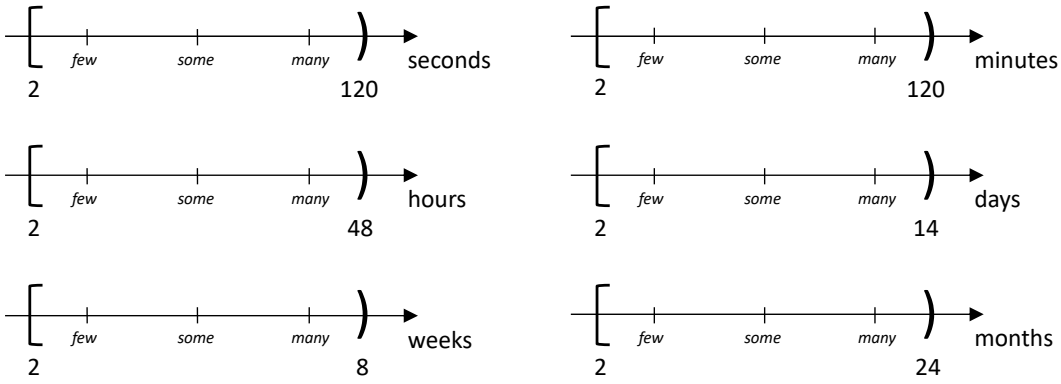


Figure 12. Time intervals for each unit

According to WFL, a person perceives the duration of a period following the logarithmic relationship shown by Equation (4.2). This equation allows us to quantify the perception *P* of a time value *S*. A zero perception ($P = 0$) would correspond to the stimulus threshold S_0, below which we cannot distinguish time values. We will assume that S_0 is 1 second, since below the second it is difficult for people to measure whether it has passed, for example, 500ms or 800ms.

By applying Equation (4.2), we know that the perception that we would have of a time value in the interval [2s, 120s) would be defined in $[k\ln(2), k\ln(120))$. In the same way, if the time value is in [2min, 120min), the perception would be in $[k\ln(2 \cdot 60), k\ln(120 \cdot 60))$. And so on with the rest of the intervals shown in Figure 12. In general, the perception of a time value in the interval $[S_1, S_2)$ will be in $[k\ln(S_1), k\ln(S_2))$, where *S* must be expressed in seconds and $k \in \mathbb{R}$ is a constant. Depending on whether we use "few", "some" or "many", the perception value can be in the lower, middle, or upper part of the interval, respectively. Considering three equal parts, we can take the midpoint of each one to represent the typical perception value for each quantifier, as shown in (4.3).

$$P_{few} = \frac{1}{6}(k\ln(S_2) - k\ln(S_1)) + k\ln(S_1)$$

$$P_{some} = \frac{1}{2}(k\ln(S_2) - k\ln(S_1)) + k\ln(S_1) \quad (4.3)$$

$$P_{many} = \frac{5}{6}(k\ln(S_2) - k\ln(S_1)) + k\ln(S_1)$$

Once we have the perception attributed to each quantifier, it is easy to obtain its respective time value by isolating *S* from Equation (4.2) and replacing *P* by the corresponding term. The result can be seen in (4.4). Note that *k* disappears from the equation, and that $(\ln(S_2) - \ln(S_1))$ has been reduced to $\ln\left(\frac{S_2}{S_1}\right)$. Finally, Table 5 presents the typical time values that represent each absolute time expression. This mapping can be used to derive time values from expressions in order to configure the underlying probabilistic network, as we will see in Section 4.4.

$$
\begin{aligned}
S_{few} &= e^{\frac{1}{6}\ln\left(\frac{S_2}{S_1}\right)+\ln(S_1)} \\
S_{some} &= e^{\frac{1}{2}\ln\left(\frac{S_2}{S_1}\right)+\ln(S_1)} \\
S_{many} &= e^{\frac{5}{6}\ln\left(\frac{S_2}{S_1}\right)+\ln(S_1)}
\end{aligned}
\tag{4.4}
$$

Table 5. Typical time values for absolute time expressions

Unit	Few	Some	Many
seconds	4s	15s	61s
minutes	3min 57s	15min 29s	60min 38s
hours	3h 23min 48s	9h 47min 52s	28h 15min 43s
days	2d 18h 23min 17s	5d 6h 59min 45s	10d 2h 56min 5s
weeks	2w 3d 15h 20min	3w 6d 23h 59min 59s	6w 2d 10h 44min
months	12w 17h 35min 14s	27w 4d 23h 45min 9s	63w 3d 3h 45s

4.3.3 Relative time expressions

Relative time expressions were also introduced in the concept map proposed in Figure 10. According to this concept map, we need two attributes to define a relative time expression: *quantity* and *reference*. The former allows one of the following quantifiers: *short*, *medium* and *long*, while the latter indicates the time frame that will be used as a reference. For example, depending on the time frame, we can use the expression “short time” as a way of saying “few minutes” if the time frame is one hour, or “few days” if the time frame is one month. We will derive numerical values from relative time expressions in a similar way to how we did it in the previous section.

First, we need to set the interval in which the resulting time value will be defined. The upper limit, which establishes the maximum time that can be attributed to a relative expression, will

be the considered time frame, expressed by S_{frame} in seconds. As for the lower limit, we have to introduce the concept of *just-noticeable difference* (JND) [43]. JND is the smallest change in stimuli that can be perceived. For instance, if a period of 10s can (only just) be distinguished from that of 12s, the JND is 2s. According to the WFL, the perception that a change produces is inversely proportional to the initial period (see Equation (4.1)), so adding 2s to a period of 10s is much more noticeable than adding the same 2s to 10 hours. In consequence, if the period is doubled, the JND also doubles (to keep the same perception). In general, the JND can be expressed as a percentage of the initial period. With all that, we will consider that the lower limit of the interval is the minimum noticeable period, that is, the JND for the given time frame, which we quantify as the 4% of the time frame. We have chosen this percentage empirically as it seems to produce reasonable results. Therefore, a relative expression will be derived to a time value in the interval $[0.04\ S_{frame},\ S_{frame}]$.

Taking into account Equation (4.2), the perception of a time value in the interval $[0.04\ S_{frame}, S_{frame}]$ is defined in $\left[k\ln\left(0.04\ S_{frame}\right), k\ln(S_{frame})\right]$. Thus, depending on whether we use "short", "middle" or "long", the perception value can be in the lower, middle, or upper part of the interval, respectively. Given three equal parts, we take the upper limit of each one to represent the typical perception value of each quantifier, as shown in (4.5). We use the upper limit instead of the midpoint, as we did in the previous section, to make the expression "long time" match the time frame.

$$\begin{aligned} P_{short} &= \frac{1}{3}\left(k\ln\left(S_{frame}\right) - k\ln\left(0.04\ S_{frame}\right)\right) + k\ln\left(0.04\ S_{frame}\right) \\ P_{medium} &= \frac{2}{3}\left(k\ln\left(S_{frame}\right) - k\ln\left(0.04\ S_{frame}\right)\right) + k\ln\left(0.04\ S_{frame}\right) \\ P_{long} &= \left(k\ln\left(S_{frame}\right) - k\ln\left(0.04\ S_{frame}\right)\right) + k\ln\left(0.04\ S_{frame}\right) \end{aligned} \tag{4.5}$$

With the perception attributed to each quantifier, we can obtain the respective time value by isolating *S* from Equation (4.2) and replacing *P* by the corresponding term. The result can be seen in (4.6). Table 6 presents the typical time values for some example expressions.

$$S_{short} = e^{\frac{1}{3}\ln(25) + \ln(0.04\, S_{frame})}$$
$$S_{some} = e^{\frac{2}{3}\ln(25) + \ln(0.04\, S_{frame})} \quad (4.6)$$
$$S_{long} = e^{\ln(25) + \ln(0.04\, S_{frame})} = S_{frame}$$

Table 6. Typical time values for some relative time expressions

Time Frame	Short	Medium	Long
10min	1min 10s	3min 25s	10min
1h	7min 1s	20min 31s	1h
5h	35min 5s	1h 42min 35s	5h
1d	2h 48min 25s	8h 12min 28s	1d
1w	19h 38min 57s	2d 9h 27min 18s	1w
4w	3d 6h 35min 51s	1w 2d 13h 49min 14s	4w

4.3.4 Random sampling of time expressions

Although we could directly use the mapping resulting from Sections 4.3.2 and 4.3.3 to convert qualitative expressions into concrete time values, sometimes it is preferable to introduce some randomness. In this section, we will assign time expressions to probability distributions instead of time expressions to values, so that an expression like "some minutes" would be translated into a random variable with values around 15 minutes. The proposal shown below is inspired by how Beta distributions are used in Subjective Logic [44].

We are going to generalize what we have introduced in the last two sections. For that, we will assume that the perception follows a beta distribution [45] whose mode corresponds to the time value that we have obtained for each expression previously. In other words, Table 5 shows the most likely value for each absolute time expression. The beta distribution will be defined as follows.

$$P \sim Beta(\alpha, \beta) \quad (4.7)$$

The parameters α and β are calculated as indicated in (4.8).

$$\alpha = \frac{2q}{v} + 1; \quad \beta = \frac{2(1-q)}{v} + 1 \quad (4.8)$$

where q is a function of the quantifier and v defines the *vagueness* as a number between 0 and 1. Table 7 presents the different values that q can adopt, which come from the intervals that were defined in the previous sections. For example, if we consider a perception normalized in the range from 0 to 1, the typical perception for "few" will be represented by $q = 1/6$ (i.e., the midpoint of the first third), $q = 1/2$ for "some" (the midpoint of the second third) and $q = 5/6$ for "many" (the midpoint of the last third). Regarding the vagueness, it controls the dispersion of the distribution, such that, as the parameter approaches 0, the more peaked the distribution is. The concept of vagueness establishes how concrete a quantifier is, so, in the extreme end, when $v \to 0$, there is no ambiguity, that is, each quantifier is associated with a single perception value, the one defined in Section 4.3.2 and 4.3.3. It is worth noting that we can select v so that a certain proportion of the samples fall within an interval. For instance, if we want the 90% of the time values generated by the quantifier "some" to have a normalized perception of $q \pm 0.1$, we must select $v = 0.031$, since it makes $P(q - 0.1 < X \leq q + 0.1) = F(q + 0.1; \alpha, \beta)$-$F(q - 0.1; \alpha, \beta) = 0.9$, being F the cumulative distribution function of $Beta(\alpha, \beta)$.

Table 7. Values of q for each quantifier

Quantifier	q
few	1/6
some	1/2
many	5/6
short	1/3
medium	2/3
long	1

In order to assign a time value to an expression, once we have modelled the perception as a random variable following a Beta distribution with parameters α' and β', we need to generate a random sample x according to this particular distribution. For that, we could use the inverse transform sampling method [46]. The resulting time value is obtained by applying Equation (4.9), where S_0, S_1 and S_2 were defined in Section 4.3.2 and 4.3.3.

$$S_x = S_0 \, e^{x \ln\left(\frac{S_2}{S_1}\right) \ln(S_1)} \tag{4.9}$$

Figure 13 shows an example of the resulting probability density for absolute (left graph) and relative (right graph) time expressions. Note that the peaks of the distributions are located on the values that were defined in the previous sections.

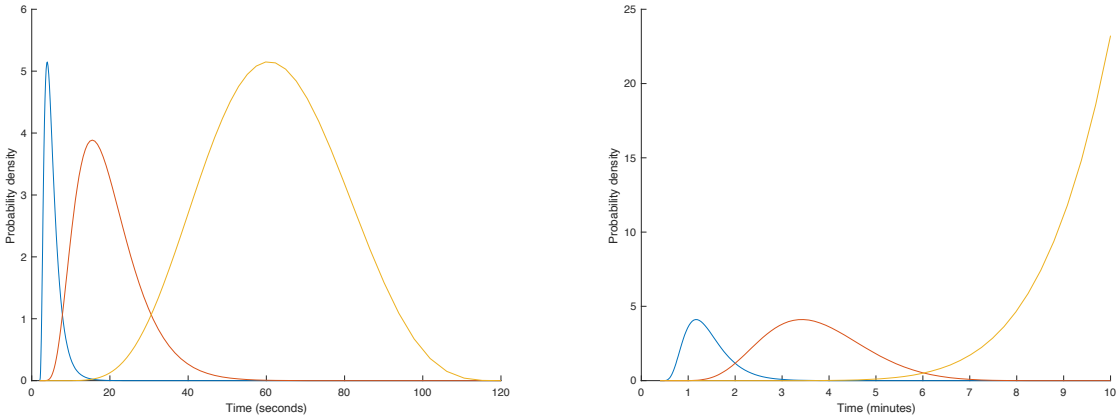


Figure 13. Example of probability densities. (Left) Probability density for "few" (blue), "some" (red) and "many" (yellow) seconds. (Right) Probability density for "short" (blue), "medium" (red) and "long" (yellow), with a time frame of 10 minutes. The vagueness has been set to 0.1

Section 4.4.4 will show how the translation of time expressions (absolute and relative) is used to derive some of the parameters of the underlying probabilistic network.

4.4 Deriving probabilities

In order to complete the probabilistic network, we need to define the probability distributions associated with the variables, mainly: $P(prop)$, $P(ctx_i \mid prop)$ and $P\left(N_{obs\,i}^{<t>} \mid ctx_i\right)$, see Figure 6 (left). To do this, it is necessary a method to transparently derive these terms from the high-level specification. It is worth noting that the process of deriving probabilities is not intended to generate precise values, but to capture the basic dynamics behind QoS metrics. As mentioned in Section 3.2.2, we regard QoS metrics as coarse-grained subjective values raised from contextual observations. Therefore, it is not significant if a probability is 0.83 or 0.82. This is also the reason why, unlike classical approaches (e.g., based on maximum likelihood [41]), we do not need to rely on data to compute probabilities.

4.4.1 Distribution P(prop)

The distribution $P(prop)$ quantifies the belief that the system behaves optimally in terms of a property before some evidence is taken into account. We will consider $P(prop)$ equal to 0.5, which expresses maximum uncertainty. That this, when there is no evidence, we do not know if the system is inclined to be optimal or not, so we make $P(PROP) = P(\neg PROP) = 0.5$. This value will represent the baseline of QoS metrics, that is, the starting point or the value to which metrics tend when there is no information.

4.4.2 Distribution P(ctx_i | prop)

The probability distribution $P(ctx_i \mid prop)$ tells us how well the presence of a context condition fits with the belief that the system is working optimally with respect to a property. To derive this distribution, we assume that the attribute *strength* of an observation indicates the Likelihood Ratio (LR) of a context condition with respect to a property, expressed as it can be seen in Equation (4.10). LR provides a measure of how discriminating a context condition is. A LR of 1 indicates that the presence of a context condition does not provide any information about the property. A LR that tends to 0 suggests a very strong positive correlation, that is, the presence of the condition implies that the system behaves optimally with respect to the property. On the contrary, if LR tends to infinity, it indicates the opposite: the system does not behave optimally. Intermediate values will exhibit a wide range of different degrees of influence. In general, values in the interval $(0,1)$ reinforce, while those in the interval $(1, +\infty)$ undermine.

Table 8 shows the assigned LR to each strength value, distinguishing whether the evidence reinforces or undermines. When reinforcing, this mapping has been designed to make *strength* and LR follow a linear relationship, so that the range of intensities of an observation is uniformly covered by *strength*. In case of undermining, the LR values are obtained as the inverse of the equivalent reinforcing LR values since we would like strength values to be symmetric and have the same intensity regardless of whether the evidence reinforces or undermines.

Table 8. Mapping strength with Likelihood Ratios

Strength	LR (Reinforcing)	LR (Undermining)
VERY HIGH	0.1	1/0.1
HIGH	0.28	1/0.28
MEDIUM	0.46	1/0.46
LOW	0.64	1/0.64
VERY LOW	0.82	1/0.82

By taking into account the definition of LR (4.10) in Equation (4.11), we can obtain (4.12), which results in (4.13) by assuming that $P(prop)$ and $P(ctx_i)$ are equal to 0.5. This final equation enables us to calculate the conditional probabilities associated with a context condition.

$$LR = \frac{P(CTX_i \mid \neg PROP)}{P(CTX_i \mid PROP)} \tag{4.10}$$

$$P(CTX_i) = P(CTX_i|PROP)P(PROP) + P(CTX_i|\neg PROP)P(\neg PROP) \tag{4.11}$$

$$P(CTX_i|PROP) = \frac{P(CTX_i)}{P(PROP) + P(\neg PROP)\,LR};\;\; P(CTX_i|\neg PROP) = LR\,P(CTX_i|PROP) \tag{4.12}$$

$$P(CTX_i|PROP) = 1/(1+LR);\;\; P(CTX_i|\neg PROP) = LR/(1+LR) \tag{4.13}$$

Example. Let us consider the observation "Bumped into somebody" specified in Table 4. The occurrence of this observation undermines safety very highly, which means a LR of 10 (i.e., 1/0.1) according to Table 8. After applying the equations in (4.13), Table 9 lists the resulting probabilities that form the distribution $P(ctx|prop)$. Note that the probabilities in each row must sum to 1 in accordance with the probability theory.

Table 9. Resulting probabilities from the observation

	¬CTX	*CTX*
¬PROP	0.0909	0.9091
PROP	0.9091	0.0909

4.4.3 Distribution P(ctx$_i$ | prop$_1$, ... , prop$_M$)

This distribution appears when an observation provides evidence for several properties, in which case it would result in a probabilistic network with a context condition shared among a number of properties, as explained in Section 3.4.4. Next, we will propose a generalization of the method presented in the previous section.

Let us consider that a context condition ctx_i produces evidence for N of the M properties defined in the network as $\{prop_1, \dots, prop_M\}$. As we did in Section 3.4.4, we can conveniently see the set of properties as a single propositional variable called $props$, which states that the system is performing optimally with respect to a specific subset of properties. Thus, the possible values of $props$ arise from all the state combinations of the properties, that is, $props$ will have 2^M possible values, $props \in \{PROPS_0, PROPS_1, \dots, PROPS_{2^M-1}\}$, where $PROPS_0$ means that the system is not running optimally for any property (i.e., $\neg PROP_M \wedge \dots \wedge \neg PROP_2 \wedge \neg PROP_1$), $PROPS_1$ means that the system is running optimally only for property $prop_1$ (i.e., $\neg PROP_M \wedge \dots \wedge \neg PROP_2 \wedge PROP_1$), and so on until $PROPS_{2^M-1}$, which means that the system is running optimally for all the properties (i.e., $PROP_M \wedge \dots \wedge PROP_2 \wedge PROP_1$). The values are ordered following the typical sequencing of a binary numeral system with symbols $\neg PROP$ (equivalent to 0) and $PROP$ (equivalent to 1), such that the index k in $PROPS_k$ is the decimal representation of the value. For instance, given 3 properties (M=3), $PROPS_5$ would correspond to $PROP_3 \wedge \neg PROP_2 \wedge PROP_1$, since the number 5 is 101 in binary.

The impact that an observation has on each property is defined individually by the attribute *strength*. We denote with the term $SE_{prop\ j} \in \{5, 4, 3, \dots, -4, -5\}$ the *evidence* associated with a property $prop_j$, where the absolute value $|SE_{prop\ j}|$ indicates the strength (5 for VERY HIGH, 4 for HIGH, 3 for MEDIUM, etc.) and the sign function $sgn(SE_{prop\ j})$ is the direction (+1 means *reinforce*, while -1 means *undermine*). By default, if not specified, $SE_{prop\ j} = 0$, which means the observation provides no evidence for property $prop_j$. Although this information is provided individually to properties, sharing a context condition makes properties conditionally dependent, so an observation will always have a global effect on them. Therefore, given the M-tuple $(SE_{prop\ 1}, \dots, SE_{prop\ M})$, Equation (4.14) allows us to quantify the degree to which an

observation supports the belief that the system acts optimally with respect to a particular subset of properties, stated by $PROPS_k$. In this equation, the term b_i is the *i*-th digit of the binary representation of *k*. The results are in the range [-5, 5] and have the same semantics that we have defined for $SE_{prop\ j}$. Moreover, the results satisfy $SE_{PROPS_k} = -SE_{PROPS_{2^M-1-k}}$, which means that $PROPS_k$ and $PROPS_{2^M-1-k}$ are opposite propositions, that is, they exhibit the same intensity but opposite directions. In general, according to Equation (4.14), the opposite of $PROPS_k$ is the one that has the state of the properties inverted (swapping $\neg PROP_i$ and $PROP_i$). For instance, as we saw previously, if $PROPS_5$ represents $PROP_3 \wedge \neg PROP_2 \wedge PROP_1$, then its opposite is $PROPS_2$, which corresponds to $\neg PROP_3 \wedge PROP_2 \wedge \neg PROP_1$.

$$SE_{PROPS_k} = \frac{1}{N} \sum_{i=1}^{M} (-1)^{(1-b_i)}\ SE_{prop\ i} \tag{4.14}$$

Example. Imagine we have an observation that produces evidence for 3 properties ($N = M$). Its occurrence will reinforce $prop_1$ highly, undermine $prop_2$ very lowly and reinforce $prop_3$ very highly, which results in $(SE_{prop\ 1}, SE_{prop\ 2}, SE_{prop\ 3}) = (4, -1, 5)$. Equation (4.14) indicates how much support the observation produces for a specific scenario in which the system is believed to be optimal (or non-optimal) with respect to some properties. Table 10 shows the results obtained after applying the equation to the possible states of the system. For instance, the observation will support (*reinforce*), with an overall intensity of 3.3333 (i.e., a high-medium intensity), the idea that the system is operating optimally in terms of $prop_1$ and $prop_3$ but poorly with respect to $prop_2$ (see row $PROPS_5$). To a lesser degree, the observation also supports, with a medium-low intensity, the belief of being optimal in terms of any property (row $PROPS_7$). On the other hand, if we think that the system is behaving in a totally deficient way (row $PROPS_0$), the observation will contradict (*undermine*) it with low-medium intensity.

Table 10. Quantification of the observation support for each possible belief

props	**Properties**	$\boldsymbol{SE_{PROPS_k}}$
$PROPS_0$	$\neg PROP_3 \wedge \neg PROP_2 \wedge \neg PROP_1$	$1/3\ (5(-1) - 1(-1) + 4(-1)) = -2.6667$
$PROPS_1$	$\neg PROP_3 \wedge \neg PROP_2 \wedge PROP_1$	$1/3\ (5(-1) - 1(-1) + 4) = 0$
$PROPS_2$	$\neg PROP_3 \wedge PROP_2 \wedge \neg PROP_1$	$1/3\ (5(-1) - 1 + 4(-1)) = -3.3333$
$PROPS_3$	$\neg PROP_3 \wedge PROP_2 \wedge PROP_1$	$1/3\ (5(-1) - 1 + 4) = -0.6667$
$PROPS_4$	$PROP_3 \wedge \neg PROP_2 \wedge \neg PROP_1$	$1/3\ (5 - 1(-1) + 4(-1)) = 0.6667$
$PROPS_5$	$PROP_3 \wedge \neg PROP_2 \wedge PROP_1$	$1/3\ (5 - 1(-1) + 4) = 3.3333$
$PROPS_6$	$PROP_3 \wedge PROP_2 \wedge \neg PROP_1$	$1/3\ (5 - 1 + 4(-1)\ ) = 0$
$PROPS_7$	$PROP_3 \wedge PROP_2 \wedge PROP_1$	$1/3\ (5 - 1 + 4) = 2.6667$

In order to derive probabilities of SE_{PROPS_k}, we will assume that SE_{PROPS_k} shows a specific Likelihood Ratio LR_k, which follows the same relationship we used in Table 8. Equation (4.15) states this mapping, where the definition of Likelihood Ratio adopted in this section is expressed by (4.16). This definition suggests that LR_k is a relative measure of how discriminating a context condition is for a certain belief with respect to its opposite. In other words, a LR_k that tends to 0 indicates that the presence of the context condition will make $PROPS_k$ much more likely than $PROPS_{2^M-1-k}$. On the other hand, if LR_k tends to infinity, it will indicate the contrary. In addition, thanks to the symmetry between opposite propositions (i.e., $SE_{PROPS_k} = -SE_{PROPS_{2^M-1-k}}$), expression (4.15) satisfies the relation $LR_k = 1/LR_{2^M-1-k}$. In the previous example, $LR_2 = \frac{1}{LR_5}$. That is, $LR_2 = (1 - \frac{0.9}{5}\ 3.3333)^{-1} = 1/0.4$, which represents $\frac{P(CTX_i \mid PROPS_5)}{P(CTX_i \mid PROPS_2)}$, and $LR_5 = (1 - \frac{0.9}{5}\ 3.3333)^{1} = 0.4$, where $LR_5 = \frac{P(CTX_i \mid PROPS_2)}{P(CTX_i \mid PROPS_5)}$.

$$LR_k = \left(1 - \frac{0.9}{5}\ |SE_{PROPS_k}|\right)^{sgn\left(SE_{PROPS_k}\right)} \tag{4.15}$$

$$LR_k = \frac{P(CTX_i \mid PROPS_{2^M-1-k})}{P(CTX_i \mid PROPS_k)} \tag{4.16}$$

To obtain the conditional distribution $P(ctx_i \mid props)$ (i.e., $P(ctx_i \mid prop_1, \ldots, prop_M)$) from the likelihood ratios, we assume that the ratio of the conditional probabilities when negating the context condition will be $1/LR_k$, as shown by Equation (4.17). In (4.18), we start from the definition of LR_k, first, we use the relation $P(CTX_i \mid PROPS_j) + P(\neg CTX_i \mid PROPS_j) = 1$ and then, we divide both sides of the equality by $P(\neg CTX_i \mid PROPS_k)$. After that, we develop the expression until we obtain the final result, which can be seen in (4.19). Note that this result is a generalization of the equations obtained in Section 4.4.2.

$$\frac{1}{LR_k} = \frac{P(\neg CTX_i \mid PROPS_{2^M-1-k})}{P(\neg CTX_i \mid PROPS_k)} \tag{4.17}$$

$$
\begin{aligned}
& LR_k\ P(CTX_i \mid PROPS_k) = P(CTX_i \mid PROPS_{2^M-1-k}) \\
& \quad \Longrightarrow LR_k\ (1 - P(\neg CTX_i \mid PROPS_k)) = 1 - P(\neg CTX_i \mid PROPS_{2^M-1-k}) \\
& \quad \Longrightarrow \frac{LR_k\ (1 - P(\neg CTX_i \mid PROPS_k))}{P(\neg CTX_i \mid PROPS_k)} = \frac{1 - P(\neg CTX_i \mid PROPS_{2^M-1-k})}{P(\neg CTX_i \mid PROPS_k)} \\
& \quad \Longrightarrow \frac{LR_k}{P(\neg CTX_i \mid PROPS_k)} - LR_k = \frac{1}{P(\neg CTX_i \mid PROPS_k)} - \frac{1}{LR_k} \\
& \quad \Longrightarrow P(\neg CTX_i \mid PROPS_k) = \frac{LR_k(LR_k - 1)}{(LR_k - 1)(LR_k + 1)} \\
& \quad \Longrightarrow P(\neg CTX_i \mid PROPS_k) = \frac{LR_k}{LR_k + 1} \\
& \quad \Longrightarrow 1 - P(CTX_i \mid PROPS_k) = \frac{LR_k}{LR_k + 1} \\
& \quad \Longrightarrow P(CTX_i \mid PROPS_k) = \frac{1}{LR_k + 1}
\end{aligned}
\tag{4.18}
$$

$$
P(CTX_i \mid PROPS_k) = \frac{1}{LR_k + 1};\ \ P(\neg CTX_i \mid PROPS_k) = \frac{LR_k}{LR_k + 1} \tag{4.19}
$$

Example. Following the previous example, we can complete the information shown in Table 10 with the conditional probabilities obtained after applying the equations in (4.19). The results can be seen in Table 11.

Table 11. Conditional probabilities for the example

props	**Likelihood ratio**	$\boldsymbol{P(CTX_i \mid props)}$	$\boldsymbol{P(\neg CTX_i \mid props)}$
$PROPS_0$	1.9231	0.3421	0.6579
$PROPS_1$	1	0.5	0.5
$PROPS_2$	2.5	0.2857	0.7143
$PROPS_3$	1.1364	0.4681	0.5319
$PROPS_4$	0.88	0.5319	0.4681
$PROPS_5$	0.4	0.7143	0.2857
$PROPS_6$	1	0.5	0.5
$PROPS_7$	0.52	0.6579	0.3421

4.4.4 Distribution $P(N^{<t>}_{obs\ i} \mid ctx_i)$

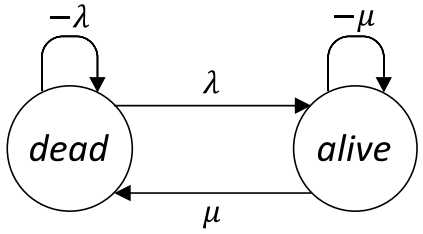
In essence, this probability distribution quantifies how well the recent history of occurrences of an observation fits with the presence of a context condition. Referring to the example on the use of robots in hospitals, this distribution may state things like, if the robot is believed to be in the "stranded without battery" condition, it is unlikely that the observation "battery less than 1% and not in the charging station" (see Table 4) would have not been occurred. In fact, the distribution will tell us that what is expected is that this observation appears several times whenever the context condition is present. In the following, we will derive the distribution $P(N^{<t>}_{obs\ i}|ctx_i)$ based on two concepts: persistence and repetition of the observations. Recall that these concepts were introduced in Section 4.1.

4.4.4.1 Persistence of the observations

In the description shown in Table 4, we annotated each observation with the maximum period of time in which the effect of an occurrence remains with some influence until it disappears. Formally, the notion of persistence appears in the term $N^{<t>}_{obs\ i}$ through the distribution $P(ACTIVE\{obs_i^{<t>}\} \mid \tau)$. According to Equation (3.1), $N^{<t>}_{obs\ i}$, the average number of past occurrences (of an observation obs_i) that are still active at time *t*, depends on $P(ACTIVE\{obs_i^{<t>}\} \mid \tau)$, the probability that a particular occurrence $obs_i^{<t>}$ continues to be active after a period of time τ. As a result, to find $P(N^{<t>}_{obs\ i}|ctx_i)$, we first need to be able to compute $N^{<t>}_{obs\ i}$, which implies that we have to derive $P(ACTIVE\{obs_i^{<t>}\} \mid \tau)$.

In order to obtain the distribution $P(ACTIVE\{obs_i^{<t>}\} \mid \tau)$, we will consider a continuous-time Markov chain [47] to describe how the state of an occurrence changes over time, from *alive*, when it is active, to *dead*, when it is not. Figure 14 shows the transition graph and the corresponding transition rate matrix (denoted by *Q*), in which all possible transitions between the two states are represented. Each element of the matrix indicates a transition depicted on the graph as an arrow, such that the elements of the first row correspond to transitions departing from *dead* and the elements of the second row those departing from *alive*. Moreover, the first column gathers the transitions ending in *dead* and the second one those ending in *alive*. The

self-transitions are on the diagonal of the matrix and have been defined so that the rows sum to zero (a necessary condition in any transition rate matrix).



$$Q = \begin{pmatrix} -\lambda & \lambda \\ \mu & -\mu \end{pmatrix}$$

Figure 14. Transition graph and transition rate matrix of an occurrence

The distribution $P(ACTIVE\{obs_i^{<t>}\} \mid \tau)$ shows its maximum value when $\tau = 0$ and decays over time, being $P(ACTIVE\{obs_i^{<t>}\} \mid \tau \to \infty) = 0$. In other words, since it happens, an occurrence moves towards the *dead* state as time passes. Therefore, as *dead* is a terminal state, λ must be equal to 0, which results in the transition rate matrix shown in (4.20).

$$Q = \begin{pmatrix} 0 & 0 \\ \mu & -\mu \end{pmatrix} \tag{4.20}$$

Thanks to the Kolmogorov forward equation, see (4.22), we can determine the transition probability matrix, denoted by $P(t)$. An element of this matrix, $P_{ij}(t)$, indicates the probability of transitioning from state i to state j at time t, assuming that $i, j \in \{0,1\}$ where 0 and 1 correspond to *dead* and *alive*, respectively. $P(ACTIVE\{obs_i^{<t>}\} \mid \tau)$ will be calculated as the probability of being in the *alive* state ($j = 1$) at time τ, that is,

$$P(ACTIVE\{obs_i^{<t>}\} \mid \tau) = P_{01}(\tau) + P_{11}(\tau) = P_{11}(\tau) \tag{4.21}$$

Note that it is reduced to $P_{11}(\tau)$ since the probability of transitioning from *dead* to *alive* is 0. To find $P_{11}(t)$, we have to solve the differential equation (4.23) obtained after developing the matrix multiplication in the Kolmogorov forward equation in (4.24).

$$\frac{dP(t)}{dt} = P(t)\, Q \tag{4.22}$$

$$\frac{dP_{11}(t)}{dt} = -\mu\, P_{11}(t) \tag{4.23}$$

$$\begin{pmatrix} dP_{00}(t)/dt & dP_{01}(t)/dt \\ dP_{10}(t)/dt & dP_{11}(t)/dt \end{pmatrix} = \begin{pmatrix} P_{00}(t) & P_{01}(t) \\ P_{10}(t) & P_{11}(t) \end{pmatrix} \begin{pmatrix} 0 & 0 \\ \mu & -\mu \end{pmatrix} \tag{4.24}$$

The solution of the equation (4.23) can be calculated as follows:

$$\begin{aligned} &\int_{P_{11}(0)}^{P_{11}(t)} \frac{dP_{11}(t)}{P_{11}(t)} = -\int_0^t \mu \, dt \\ &\Rightarrow \ln[P_{11}(t)] - \ln[P_{11}(0)] = -\mu \, t \\ &\Rightarrow P_{11}(t) = P_{11}(0) \, e^{-\mu \, t} \end{aligned} \tag{4.25}$$

Assuming that the probability of being active for an occurrence that just happened is 1, $P_{11}(0)$ must also be equal to 1, which produces the final result (4.26). To get μ, we have to consider the persistence value defined in the high-level specification. As mentioned, persistence indicates the maximum period of time that an occurrence remains. We will think that "the maximum period" is the time in which the probability of being active approaches a value close to 0, in our case, 0.01. Taking into account this and clearing μ from (4.26), we obtain Equation (4.27). Note that all time values in these expressions will be in seconds.

$$P(ACTIVE\{obs_i^{<t>}\} \mid \tau) = e^{-\mu \, \tau} \tag{4.26}$$

$$\mu = -\frac{ln(0.01)}{persistence} \tag{4.27}$$

4.4.4.2 Repetition of the observations

The presence of a context condition can make an observation occurs multiple times, where the influence of each occurrence may depend on the previous ones. To set this behaviour in the distribution $P(\mathrm{N}_{\mathrm{obs\,i}}^{<t>}|ctx_i)$, we will specify the Likelihood Ratio (LR) of the average number of active occurrences $\mathrm{N}_{\mathrm{obs\,i}}^{<t>}$ given a context condition ctx_i, as defined in Equation (4.28). Figure 15 shows two possible LR profiles. In the left graph, occurrences gradually become more discriminating as the average number of active occurrences grows, so that the first occurrences have less impact than the later ones. In other words, as an observation is repeated, the process gains confidence that the context condition is present. As for the right graph, the curve is sharper, which means that almost from the first occurrence the impact will always be the same. Note that, as the LR approaches 0, the evidence that the context condition exists becomes stronger.

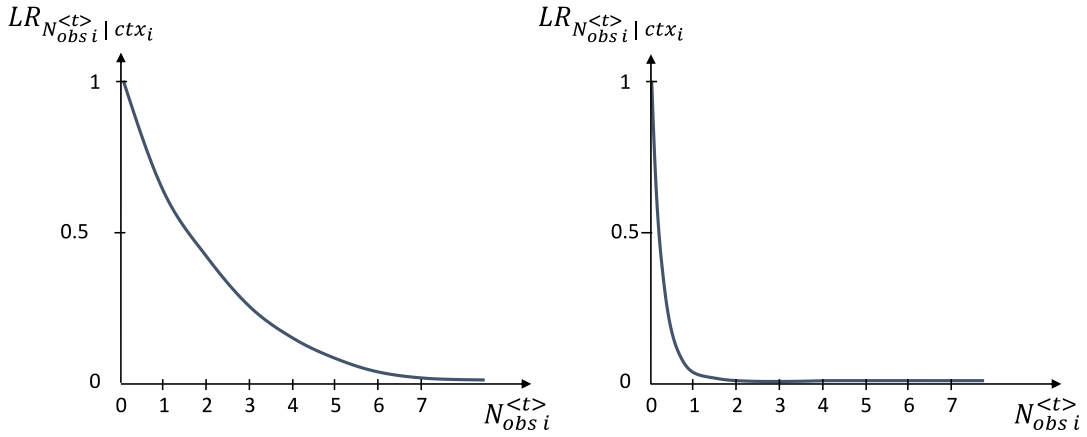


Figure 15. Possible LR profiles for $\mathrm{N}_{\mathrm{obs\,i}}^{<t>}$ given a context condition ctx_i

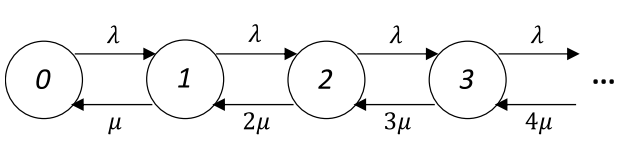
$$LR = \frac{P(\mathrm{N}_{\mathrm{obs\,i}}^{<t>} \mid \neg CTX_i)}{P(\mathrm{N}_{\mathrm{obs\,i}}^{<t>} \mid CTX_i)} \tag{4.28}$$

The LR profile function. Although we can use any function $f: \mathbb{R}_{>0} \to (0, +\infty)$ to specify the LR profile, we will consider an exponential function due to: (1) its simplicity, it is easy to operate; (2) its flexibility, it has one parameter that allows us to adjust its shape (as shown in Figure 15); and (3) its suitability, it fits well with the expected behaviour: evidence gains confidence as the observation is repeated. In particular, the function will be defined as follows.

$$LR(\mathrm{N}_{\mathrm{obs\,i}}^{<t>}) = e^{-\alpha\, \mathrm{N}_{\mathrm{obs\,i}}^{<t>}} \quad (\mathrm{N}_{\mathrm{obs\,i}}^{<t>} \geq 0) \tag{4.29}$$

The parameter α will be calculated taking into account how the attribute *repetition* was set in the high-level specification. If it has not been defined, we will select a profile like the right graph in Figure 15, in which occurrences tend to behave as independent observations. Conversely, if *repetition* has been defined, the profile may be more similar to the left graph.

M/M/∞ as an equivalent model. To delve into the problem of how to calculate the parameter α according to *repetition* and then estimate the distribution $P(\mathrm{N}_{\mathrm{obs\,i}}^{<t>} | ctx_i)$, we consider that, when a context condition ctx_i is present, the corresponding observation obs_i occurs following a Poisson process. In that situation, our model will behave as a M/M/∞ queue [48]. Figure 16 shows its transition graph and transition rate matrix.



$$Q = \begin{pmatrix} -\lambda & \lambda & 0 & 0 & 0 \\ \mu & -(\mu+\lambda) & \lambda & 0 & 0 \\ 0 & 2\mu & -(2\mu+\lambda) & \lambda & 0 \\ 0 & 0 & 3\mu & -(3\mu+\lambda) & \lambda \\ 0 & 0 & 0 & 4\mu & \ddots \end{pmatrix}$$

Figure 16. Transition graph and transition rate matrix for a M/M/∞ queue. Self-transitions have been omitted in the diagram for the sake of clarity

In queueing theory, a M/M/∞ is a stochastic model that assumes the following conditions:

- The state space is a set {0, 1, 2, 3, ...} where the value corresponds to the number of jobs in the system.
- Arrivals occur at rate λ according to a Poisson process and move the system from state i to $i+1$.
- The system has an infinite number of servers so arriving jobs are served with no waiting time.
- The service time is exponentially distributed with parameter μ. Transitions from state i to $i-1$ are at rate $i\mu$ (since i jobs are being served in parallel).

At the end of this section, we will provide a proof that our process is equivalent to a M/M/∞ queue. For now, we can intuitively see that this model fits well with our problem, such that (1) arrivals are occurrences of an observation that is repeated with rate λ, (2) the state indicates the average number of active occurrences at a given time, (3) occurrences are stochastically independent and remain active for a time that is exponentially distributed with parameter μ according to (4.27) and (4) when an occurrence dies and is no longer active, we consider it served and out of the system.

Delving into how $N_{obs\,i}^{<t>}$ changes over time in order to determine λ (and consequently α). We are going to pay attention to how the average number of active occurrences $N_{obs\,i}^{<t>}$ changes over time from t=0 until reaching stability, which, in terms of a M/M/∞ queue, means to looking at the average queue length during the transient period. According to this model, when the process starts in state 0, $N_{obs\,i}^{<0>}$ is 0. As the observation is repeated, $N_{obs\,i}^{<t>}$ gradually increases until it reaches stability at λ/μ. Equation (4.30) describes this behaviour (see [48] for more details about the derivation of this formula on a M/M/∞ queue).

$$N_{obs\ i}^{<t>} = \frac{\lambda}{\mu}\left(1 - e^{-\mu t}\right) \tag{4.30}$$

The length of the transient period coincides with the persistence time of the observation, that is, the maximum period in which an occurrence remains with some influence. However, as we limited this maximum period to the time in which the probability of being active drops to 0.01 (see the previous section), if we apply $t = persistence$ to Equation (4.30) (and the definition of μ given in (4.27)), we are not going to obtain λ/μ but a close approximation: $N_{obs\ i}^{<persistence>} = \frac{\lambda}{\mu}\left(1 - e^{\ln(0.01)\ persistence/persistence}\right) = 0.99\,\lambda/\mu$. Figure 17 illustrates how $N_{obs\ i}^{<t>}$ tends to λ/μ ($N_{obs\ i}^{<persistence>} = 4.34$ in the example), which slightly differs from the value obtained ($N_{obs\ i}^{<persistence>} = 4.30$) when applying the corresponding persistence time (t = 600s).

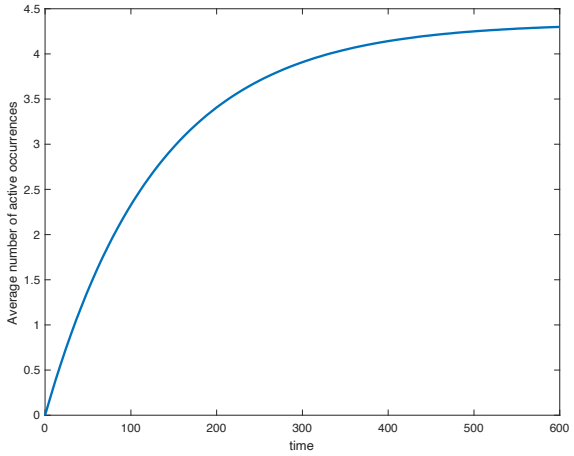


Figure 17. Example of how $N_{obs\ i}^{<t>}$ changes during the transient period

In the high-level specification, we introduced the concept of *repetition* to indicate that, during the life of an occurrence, others may happen in a way that all contribute to establishing the overall influence of the evidence. In this sense, we consider that the attribute *repetition* (how many times an observation has to be repeated to have full effect) will define the average number of active occurrences that is achieved when the observation reaches stability, that is, $N_{obs\ i}^{<stability>} = repetition + 1$ (we add 1 to take into account the first occurrence of the observation). Therefore, clearing λ from $N_{obs\ i}^{<stability>} = \lambda/\mu$, we obtain (4.31). Note that by default $repetition = 0$, which make $N_{obs\ i}^{<t>}$ tend to $\lambda/\mu = 1$, since on average only one occurrence will be expected.

$$\lambda = (\text{repetition} + 1)\ \mu \tag{4.31}$$

Calculation of α. The presence of a context condition will become apparent if the average number of active occurrences from the corresponding observation behaves as described in Figure 17. While the first occurrences may not mean anything, the evidence will get stronger as $\mathrm{N}_{\mathrm{obs\,i}}^{<t>}$ grows, reaching its maximum effect once $\mathrm{N}_{\mathrm{obs\,i}}^{<t>}$ enters the stationary phase at λ/μ ($0.99\,\lambda/\mu$, considering our approximation). Therefore, the parameter α in (4.29) will be set to show a highly discriminant LR (0.01, in our case) at $\mathrm{N}_{\mathrm{obs\,i}}^{<t>} = 0.99\ \lambda/\mu$, as seen in (4.32).

$$0.01 = e^{-\alpha\, 0.99\frac{\lambda}{\mu}} \implies \alpha = -\frac{\ln(0.01)}{0.99\,\lambda/\mu} \tag{4.32}$$

Approach to find the distribution of $\mathbf{N}_{\mathbf{obs\,i}}^{<t>}$. Now let us focus on $P\left(\mathrm{N}_{\mathrm{obs\,i}}^{<t>} \middle| CTX_i\right)$, that is, the distribution of the average number of active occurrences for the observation obs_i when the context condition ctx_i is present. In order to simplify the handling of probabilities, we are going to quantize $\mathrm{N}_{\mathrm{obs\,i}}^{<t>}$ into K+1 bins, so that we go from a continuous variable in the range $[0, +\infty)$ to a categorical variable defined in a finite set, $qn_{\mathrm{obs\,i}}^{<t>} \in \{Q_0, Q_1, Q_2 \ldots, Q_K\}$. The considered quantization process is specified in (4.33). Figure 18 shows a visual representation of the mapping between $\mathrm{N}_{\mathrm{obs\,i}}^{<t>}$ and $qn_{\mathrm{obs\,i}}^{<t>}$. To determine Δ we will consider that Q_K is reached when $\mathrm{N}_{\mathrm{obs\,i}}^{<t>}$ enters the stationary phase, that is, $\left(K - \frac{1}{2}\right)\Delta = 0.99\,\lambda/\mu$.

$$\begin{aligned}
&qn_{obs\,i}^{<t>} = Q_0 \iff \mathrm{N}_{\mathrm{obs\,i}}^{<t>} < \frac{\Delta}{2} \\
&qn_{obs\,i}^{<t>} = Q_k \iff \left(k - \frac{1}{2}\right)\Delta \le \mathrm{N}_{\mathrm{obs\,i}}^{<t>} < \left(k + \frac{1}{2}\right)\Delta \quad (k = 1,2 \ldots K-1) \\
&qn_{obs\,i}^{<t>} = Q_K \iff \mathrm{N}_{\mathrm{obs\,i}}^{<t>} \ge \left(K - \frac{1}{2}\right)\Delta \\
&where\ \Delta = \frac{0.99\,\lambda/\mu}{K - 1/2}
\end{aligned} \tag{4.33}$$

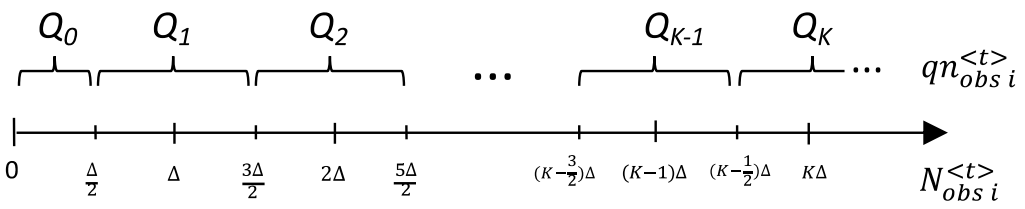


Figure 18. Visual representation of the quantization of $\mathrm{N}_{\mathrm{obs\,i}}^{<t>}$

We will distinguish between transient and stationary phase, such that the probability distribution can be decomposed as shown in (4.34).

$$\begin{aligned} P(\mathrm{N}_{\mathrm{obs\,i}}^{<t>}|CTX_i) \sim & P(qn_{\mathrm{obs\,i}}^{<t>}|CTX_i) \\ & = P(qn_{\mathrm{obs\,i}}^{<t>} \mid \neg STATIONARY, CTX_i)\, P(\neg STATIONARY \mid CTX_i) \\ & + P(qn_{\mathrm{obs\,i}}^{<t>} \mid STATIONARY, CTX_i)\, P(STATIONARY \mid CTX_i) \end{aligned} \tag{4.34}$$

where:

- $P(\mathrm{qn}_{\mathrm{obs\,i}}^{<t>} \mid \neg STATIONARY, CTX_i)$ is the distribution of $qn_{\mathrm{obs\,i}}^{<t>}$ during the transient phase.
- $P(\mathrm{qn}_{\mathrm{obs\,i}}^{<t>} \mid STATIONARY, CTX_i)$ is the distribution of $qn_{\mathrm{obs\,i}}^{<t>}$ during the stationary phase.
- $P(STATIONARY \mid CTX_i)$ is the probability of being in the stationary phase and $P(\neg STATIONARY \mid CTX_i) = 1 - P(STATIONARY \mid CTX_i)$ the probability of being in the transient phase.

Based on the LR profile, we will be able to obtain the probability distribution when the context condition is not present ($ctx_i = \neg CTX_i$) as seen in (4.35).

$$P(\mathrm{N}_{\mathrm{obs\,i}}^{<t>}|\neg CTX_i) \sim P(qn_{\mathrm{obs\,i}}^{<t>}|\neg CTX_i) = P(qn_{\mathrm{obs\,i}}^{<t>}|CTX_i)\, LR(k\Delta) \tag{4.35}$$

The expression $LR(k\Delta)$ is the LR associated with $\mathrm{N}_{\mathrm{obs\,i}}^{<t>} = k\Delta$, which is the middle of the bin defined by Q_k. However, according to its definition in (4.29), $LR \leq 1\ \forall qn_{\mathrm{obs\,i}}^{<t>}$, so that the sum of the distribution cannot be 1, i.e. $\sum_{k=0}^{K} P(qn_{\mathrm{obs\,i}}^{<t>} = Q_k \mid \neg CTX_i) < 1$. This fact violates the principles of probability theory because we are ignoring the concept of counterevidence.

As an observation occurs it reinforces the claim that a certain context condition is present, this is the idea behind $\mathrm{N}_{\mathrm{obs\,i}}^{<t>}$ - the more occurrences per unit of time, the greater its value and, therefore, the greater the probability that a certain condition exists. When there are no occurrences of an observation ($\mathrm{N}_{\mathrm{obs\,i}}^{<t>} = 0$) the uncertainty is maximum. However, in our model, we have not considered counterevidence, that is, evidence that contradicts the presence of the context condition. Intuitively, the concept of counterevidence helps us balance the probability distribution, so that its sum is 1. Counterevidence could be included explicitly as a new type of observation or implicitly as a function of time - the longer without positive evidence, the more likely the context condition is not present. Although this concept is not addressed, (4.36) shows

its effect on the probability distribution. In this expression, a new value for $qn_{\text{obs i}}^{<t>}$ (named Q_{ce}) represents any counterevidence.

$$P(qn_{\text{obs i}}^{<t>} = Q_{ce}|CTX_i) = 0;\;\; P(qn_{\text{obs i}}^{<t>} = Q_{ce}|\neg CTX_i) = 1 - \sum_{k=0}^{K} P(qn_{\text{obs i}}^{<t>} = Q_k \mid \neg CTX_i) \tag{4.36}$$

Distribution during the transient phase. Given a context condition ctx_i, normally, the transient phase appears twice: (1) when the condition begins to be present, and (2) once the condition clears. For example, the observation "battery less than 1% and not in the charging station" will start when the robot is entering the "stranded without battery" condition. As the observation is repeated, more evidence will be added and, eventually, $\text{N}_{\text{obs i}}^{<t>}$ will reach stability, consolidating the belief of being in that condition. After a while, the situation of the robot may change and then the condition would disappear, e.g., someone could have replaced the battery. From that moment, $\text{N}_{\text{obs i}}^{<t>}$ will gradually drop to 0. This closing period will behave as the opening one but in reverse. Although in terms of distribution of $\text{N}_{\text{obs i}}^{<t>}$ (or $\text{qn}_{\text{obs i}}^{<t>}$) both transient periods are equivalent, we focus on the opening period, which was described by the M/M/∞ queue through the Equation (4.30). It is because we are conditioned to the presence of ctx_i (i.e., $ctx_i = CTX_i$) and the transient closing only happens once this condition disappears (i.e., $ctx_i = \neg CTX_i$).

To obtain the cumulative distribution function (CDF) of $\text{N}_{\text{obs i}}^{<t>}$ during the transient period, we consider that time *t* is a uniformly distributed variable, whose values are in the interval [0, *T*], with *t* = 0 being the beginning of the transient period and *t* = *T* the end of this period. Note that *T* will correspond to the configured *persistence* time. Expression (4.37) shows the CDF of *t*. Based on this function, we can develop the CDF of $\text{N}_{\text{obs i}}^{<t>}$, denoted by $\text{F}_N(n)$, as seen in (4.38).

$$\text{F}_t(t') = P(\text{t} < t') = \frac{t'}{T} \qquad (0 \le t' \le T) \tag{4.37}$$

$$\begin{aligned} \text{F}_N(n) &= P(\text{N}_{\text{obs i}}^{<t>} < n \mid \neg STATIONARY, CTX_i) = P\left(\frac{\lambda}{\mu}(1 - e^{-\mu t}) < n \mid \ldots\right) \\ &= P\left(t < \frac{-1}{\lambda}\ln\left(1 - \frac{\mu}{\lambda}n\right) \mid \ldots\right) = \frac{-1}{\mu T}\ln\left(1 - \frac{\mu}{\lambda}n\right) \\ &where\ 0 \le n \le \frac{\lambda}{\mu}(1 - e^{-\mu T}) \end{aligned} \tag{4.38}$$

$P\left(qn_{\text{obs i}}^{<t>} \mid \neg STATIONARY, CTX_i\right)$ arises from applying the CDF above to obtain the probability that $\text{N}_{\text{obs i}}^{<t>}$ lies in each of the bins defined by $qn_{\text{obs i}}^{<t>}$. The resulting probability distribution has been developed in (4.39). Note that we have assigned zero to the probability associated with Q_K, since it is only present in the stationary phase. Also, we arrived at the final expressions after some simplifications using the definition of Δ and μ in (4.33) and (4.27) respectively, and considering T as the persistence time. Although these expressions end up not depending on μ or λ, these parameters are an essential part in the quantization of $\text{N}_{\text{obs i}}^{<t>}$ into Q_0, Q_1, Q_2 ..., so μ and λ are still present indirectly, see (4.33).

$$
\begin{aligned}
&P(qn_{\text{obs i}}^{<t>} = Q_0 \mid \neg STATIONARY, CTX_i) = \text{P}\left(\text{N}_{\text{obs i}}^{<t>} < \frac{\Delta}{2} \mid \ldots\right) = \text{F}_N\left(\frac{\Delta}{2}\right) = \frac{-1}{\mu T}\ln\left(1 - \frac{\mu}{2\lambda}\Delta\right) \\
&\quad = \frac{-1}{\ln(0.01)}\ln\left(1 - \frac{0.99}{2K-1}\right) \\
&P(qn_{\text{obs i}}^{<t>} = Q_k \mid \neg STATIONARY, CTX_i) = \text{P}\left(\left(k - \frac{1}{2}\right)\Delta \le \text{N}_{\text{obs i}}^{<t>} < \left(k + \frac{1}{2}\right)\Delta \mid \ldots\right) \\
&\quad = \text{F}_N\left(\left(k + \tfrac{1}{2}\right)\Delta\right) - \text{F}_N\left(\left(k - \tfrac{1}{2}\right)\Delta\right) = \tfrac{1}{\mu T}\left[\ln\left(1 - \tfrac{\mu}{\lambda}\left(k - \tfrac{1}{2}\right)\Delta\right) - \ln\left(1 - \tfrac{\mu}{\lambda}\left(k + \tfrac{1}{2}\right)\Delta\right)\right] \\
&\quad = \tfrac{-1}{\ln(0.01)}\left[\ln\left(1 - \tfrac{0.99\,(k-1/2)}{K-1/2}\right) - \ln\left(1 - \tfrac{0.99\,(k+1/2)}{K-1/2}\right)\right] \quad (0 < \text{k} < K) \\
&P(qn_{\text{obs i}}^{<t>} = Q_K \mid \neg STATIONARY, CTX_i) = 0
\end{aligned}
\tag{4.39}
$$

Distribution during the stationary phase. In the stationary phase, $\text{N}_{\text{obs i}}^{<t>}$ is expected to be in the interval $(0.99\,\lambda/\mu,\ \lambda/\mu)$, since at the end of the transient period $\text{N}_{\text{obs i}}^{<t>}$ is $0.99\,\lambda/\mu$ and it tends to λ/μ as time passes. However, in practice, $\text{N}_{\text{obs i}}^{<t>}$ may fluctuate due to its random nature or because the configured λ and μ may not fit well with reality, causing $\text{N}_{\text{obs i}}^{<t>}$ to fall outside this interval. For this reason, as we can see in (4.40), we have considered a very simple distribution, where Q_K represents any value of $\text{N}_{\text{obs i}}^{<t>}$ greater than $0.99\,\lambda/\mu$. This will make the model more robust.

$$
\begin{aligned}
&P(qn_{\text{obs i}}^{<t>} = Q_k \mid STATIONARY, CTX_i) = 0 \quad (0 \le \text{k} < K) \\
&P(qn_{\text{obs i}}^{<t>} = Q_K \mid STATIONARY, CTX_i) = 1
\end{aligned}
\tag{4.40}
$$

Probability of being in the stationary phase. If the context condition ctx_i is present, the term $P(STATIONARY \mid CTX_i)$ indicates the proportion of time in which $\text{N}_{\text{obs i}}^{<t>}$ is in the stationary phase. For example, regarding the context condition "stranded without battery", a probability of 0.8 would indicate that if on average this condition lasts 10 minutes, the robot will spend 2 minutes consolidating the evidence (the transient phase), and 8 minutes in which the evidence will have full effect on the *safety* estimate (the stationary phase). Since we do not have a priori information, we will start by considering $P(STATIONARY \mid CTX_i) = 0.5$ (i.e., both phases are weighted equally in (4.34)). Although we could adjust this assumption at runtime based on the data, this would have no effect on the results since it does not change the likelihood ratio used to build the probability distribution in (4.34).

A proof that our model is equivalent to a M/M/∞ queue. To show the equivalence between a M/M/∞ queue and our model when observations follow a Poisson process, we will check whether both approaches provide the same expected value for $\text{N}_{\text{obs i}}^{<t>}$. In a M/M/∞ queue, $\text{N}_{\text{obs i}}^{<t>}$ is the average queue length, whose expected value is λ/μ [48]. In our model, $\text{N}_{\text{obs i}}^{<t>} = \sum_{\forall t_i} e^{-\mu(t-t_i)}$, that is, the average number of active occurrences of obs_i at time t was defined as the sum of the probabilities that any past occurrence (detected at time t_i) remains active (see this definition in (3.1) and how we derived the probability in Section 4.4.4.1). As the expected value of the sum of several random variables is equal to the sum of their expectations, we can reduce the problem as follows.

$$E(\text{N}_{\text{obs i}}^{<t>}) = E\left(\sum_{\forall t_i} e^{-\mu(t-t_i)}\right) = \sum_{\forall t_i} E\left(e^{-\mu(t-t_i)}\right) = \sum_{\forall t_i} E(N_i) \tag{4.41}$$

In order to calculate $E(N_i)$, first, we have to determine the cumulative distribution function (CDF) of N_i. As we did before when we presented the Equation (4.37), we consider that time t is a uniformly distributed variable, whose values are in the interval [0, T). The resulting CDF can be seen in (4.42).

$$F(N_i) = P(N_i \leq n) = P\left(e^{-\mu(t-t_i)} \leq n\right) = 1 - P\left(t \leq \frac{-1}{\mu}\ln(\text{n}) + t_i\right) = \frac{1}{\mu T}\ln(\text{n}) + \frac{t_i}{T} + 1$$
$$where\ t_i \in \mathbb{R} \mid 0 \leq t_i < T\ \ and\ \ n \in \mathbb{R} \mid 0 \leq n \leq 1 \tag{4.42}$$

Once we have the CDF, we can compute the probability density function (PDF) of N_i, and from that, the expected value $E(N_i)$. This process is shown in (4.43) and (4.44) respectively.

$$f(N_i) = \frac{dF(n)}{dn} = \frac{d}{dn}\left(\frac{1}{\mu T}\ln(\mathrm{n}) + \frac{t_i}{T} + 1\right) = \frac{1}{\mu T n} \tag{4.43}$$

$$E(N_i) = \int_0^1 n\, f(n)\, dn = \int_0^1 \frac{1}{\mu T}\, dn = \frac{1}{\mu T}[1-0] = \frac{1}{\mu T} \tag{4.44}$$

The term $E(N_i)$ is the contribution to $E\left(\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}>}\right)$ of a past occurrence detected at time t_i. Given that this term is independent of t_i, the summation $\sum_{\forall t_i} E(N_i)$ can be reduced to multiply $E(N_i)$ by the average number of occurrences that appear in a period T. For that, as we have assumed a Poisson process to describe how often an observation occurs when a context condition is present, if an observation is repeated with rate λ, the average number of occurrences in a period T will be λT. Thus, the resulting expected value of $\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}>}$ is finally obtained in (4.45). Note that this value is the same as that prescribed by the M/M/∞ queue.

$$E(\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}>}) = \sum_{\forall t_i} E(N_i) = \lambda T\, \frac{1}{\mu T} = \frac{\lambda}{\mu} \tag{4.45}$$

Example. Let us consider the observation "stranded without battery" specified in Table 4. This observation must be repeated at least 5 times for the evidence to have full effect. Moreover, its persistence is defined as "medium" (relative to a 7-day time frame). According to Section 4.3.3, it will result in a *persistence* of 2d 9h 27min 18s (i.e., 206838s). From this information, we can calculate the following parameters.

$$\begin{aligned} \mu &= -\frac{\ln(0.01)}{persistence} = -\frac{\ln(0.01)}{206838} = 2.2265 \cdot 10^{-5} \\ \lambda &= (repetition + 1)\,\mu = (5+1)\,2.2265 \cdot 10^{-5} = 1.3359 \cdot 10^{-4} \\ \alpha &= -\frac{\ln(0.01)}{0.99\,\lambda/\mu} = -\frac{\ln(0.01)}{0.99 \cdot 1.3359 \cdot 10^{-4}/2.2265 \cdot 10^{-5}} = 0.7753 \end{aligned} \tag{4.46}$$

Taking into account 11 levels of quantization and applying the equations obtained in this section, Table 12 presents the resulting probabilities.

Table 12. Probabilities derived from the observation

$N_{obs\ i}^{<t>}$	$qn_{obs\ i}^{<t>}$	$P(qn_{obs\ i}^{<t>} \mid CTX_i)$	$P(qn_{obs\ i}^{<t>} \mid \neg CTX_i)$
[0, 9.87)	Q_0	0.0058	0.0058
[9.87, 29.62)	Q_1	0.0126	0.0078
[29.62, 49.37)	Q_2	0.0143	0.0054
[49.37, 69.12)	Q_3	0.0165	0.0039
[69.12, 88.87)	Q_4	0.0195	0.0028
[88.87, 108.62)	Q_5	0.0237	0.0021
[108.62, 128.37)	Q_6	0.0304	0.0017
[128.37, 148.12)	Q_7	0.0424	0.0014
[148.12, 167.87)	Q_8	0.0704	0.0015
[167.87, 187.62)	Q_9	0.2644	0.0034
[187.62, +∞)	Q_{10}	0.5	0.0039

4.5 Summary

In this chapter, we have presented a high-level specification to help domain experts express QoS metrics on non-functional properties, conforming to the probabilistic framework introduced in Chapter 3. From this specification, a ready-to-use probabilistic network is derived transparently without having to deal with its complexities. Next chapter will show how this probabilistic network can be put into action when running QoS metrics.

Chapter 5 Running QoS metrics

This chapter covers the main aspects of running and using QoS metrics, including the algorithms for calculating QoS estimates, the application of statistics, and how to tune QoS metrics.

5.1 Process for calculating QoS metrics

Although Section 3.4 introduced the mathematics that allows us to obtain QoS estimates, the classical formulation of a probabilistic network is not practical. Mainly because it makes extensive use of multiplications, so that the more observations the more multiplications will appear, and precision can end up being an issue since the operands are small numbers (probabilities). This section shows the complete process to calculate QoS metrics: (1) detect the occurrence of observations, (2) update the accumulated evidence, and (3) estimate QoS metric. For the second point we will present a robust method to address the update without the need to store the full history of occurrences. Finally, we will introduce a more practical formulation as an alternative to the classical one.

5.1.1 First step: detect the occurrence of observations

Observations can be thought of as event-driven monitors that react to specific changes in a set of context variables. Each observation defines a condition, so that, when it is detected, the corresponding observation produces an event, which, as a result, updates the input evidence of the probabilistic network. In a later chapter we will see more details of how this mechanism can be implemented.

5.1.2 Second step: update the accumulated evidence

Whenever an observation obs_i occurs, its average number of active occurrences $N_{obs\,i}^{<t>}$ will need to be updated. This term represents the accumulated evidence associated with an observation, which, according to equations (3.1) and (4.16), is defined as follows.

$$\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}>} = \sum_{\forall k} e^{-\mu\tau_k} \tag{5.1}$$

Being τ_k the period of time since the *k*-th occurrence of the observation, i.e., $\tau_k = t - t_k$, where t is the current time and t_k the time the *k*-th occurrence happened. To recalculate $N_{obs\,i}^{<t>}$ with each new occurrence without having to store the entire history of occurrences, we can rewrite the previous expression as shown in (5.2). Note that the last summation can be easily updated as new occurrences arrive. This summation can be represented by the accumulator variable $acc_{obs\,i}$, such that, when we have a new occurrence at time t_n, we just need to add its contribution to the accumulator (i.e., $acc_{obs\,i} \leftarrow acc_{obs\,i} + e^{\mu t_n}$).

$$\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}>} = \sum_{\forall k} e^{-\mu(t-t_k)} = e^{-\mu t} \sum_{\forall k} e^{\mu t_k} = e^{-\mu t}\, acc_{obs\,i} \tag{5.2}$$

However, although this approach allows us to ignore the previous history of occurrences, $e^{\mu t_n}$ will tend to get larger and larger as t_n increases, so the accumulator would not take long to have an arithmetic overflow. To solve it, below we have expressed *N* when a new occurrence arrives at t_{n+1} as a function of the previous value at t_n.

$$\begin{aligned} \mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}_{n+1}>} &= \sum_{k=0}^{n+1} e^{-\mu(t_{n+1}-t_k)} = e^{-\mu t_{n+1}} \sum_{k=0}^{n+1} e^{\mu t_k} = e^{-\mu t_{n+1}} \left(e^{\mu t_{n+1}} + \sum_{k=0}^{n} e^{\mu t_k} \right) \\ &= 1 + e^{-\mu t_{n+1}} \sum_{k=0}^{n} e^{\mu t_k} = 1 + e^{-\mu t_{n+1}}\, e^{\mu t_n}\, e^{-\mu t_n} \sum_{k=0}^{n} e^{\mu t_k} \end{aligned} \tag{5.3}$$

which finally results in

$$\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}_{n+1}>} = 1 + e^{-\mu(t_{n+1}-t_n)}\, \mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}_n>} \tag{5.4}$$

Equation (5.4) shows how *N* can be recursively updated each time a new occurrence arrives, depending only on the previous update ($\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}_n>}$) and the arrival time of the new (t_{n+1}) and the previous occurrence (t_n). Note that, since *N* is initially equal to 0, the first occurrence at t_0 will

have $\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}_0>} = 1$. In the same way, if we wanted to calculate $N\,\tau$ seconds after the arrival of an occurrence at t_n, being $0 < \tau < t_{n+1}$, we should apply the following equation.

$$\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}_n+\tau>} = \sum_{k=0}^{n} e^{-\mu(t_n+\tau-t_k)} = e^{-\mu(t_n+\tau)} \sum_{k=0}^{n} e^{\mu t_k} = e^{-\mu\tau}\, e^{-\mu t_n} \sum_{k=0}^{n} e^{\mu t_k} \tag{5.5}$$

which results in

$$\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}_n+\tau>} = e^{-\mu\tau}\, \mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}_n>} \tag{5.6}$$

The method we propose consists of two steps. First, the accumulated evidence associated with each observation is continuously updated every time a new occurrence arrives. Based on Equation (5.4), Listing 1 shows the algorithm to update *N* with each new occurrence. Second, if we want to compute a new QoS estimate at time t, we will apply Equation (5.6) to obtain the current value of $\mathrm{N}_{\mathrm{obs\,i}}^{<\mathrm{t}>}$ for each observation. Listing 2 shows the algorithm. This approach offers us an efficient and robust way of handling the accumulated evidence from the observations.

algorithm update-evidence-with-new-occurrence-obs_i **is**
const: $\mu_{obs\,i} \leftarrow -\ln(0.01)/persistence_{obs\,i}$ *($persistence_{obs\,i}$ in seconds see Equation (4.27))*
global: $N_{obs\,i}^{<t_{prev}>}$, update of N for the previous occurrence *(initialized to 0)*
t_{prev}, time of the previous occurrence in seconds *(initialized to 0)*
input: t_{new}, time of the new occurrence in seconds
output: $N_{obs\,i}^{<t_{new}>}$, updated value of N for the observation
precondition: $t_{new} > t_{prev}$
$N_{obs\,i}^{<t_{new}>} \leftarrow 1 + exp\left(-\mu_{obs\,i} \times \left(t_{new} - t_{prev}\right)\right) \times N_{obs\,i}^{<t_{prev}>}$
$t_{prev} \leftarrow t_{new}$
$N_{obs\,i}^{<t_{prev}>} \leftarrow N_{obs\,i}^{<t_{new}>}$
return $N_{obs\,i}^{<t_{new}>}$

Listing 1. Algorithm for updating the accumulated evidence

algorithm calculate-N-obs_i **is**
const: $\mu_{obs\,i} \leftarrow -\ln(0.01)/persistence_{obs\,i}$ *($persistence_{obs\,i}$ in seconds see Equation (4.27))*
input: $N_{obs\,i}^{<t_{last}>}$, N of the last occurrence *(initialized to 0) (calculated in Listing 1)*
t_{last}, time of the last occurrence in seconds *(initialized to 0)*
t, current time in seconds
output: $N_{obs\,i}^{<t>}$, N for the observation *i* at time *t*
precondition: $t > t_{last}$
return $exp(-\mu_{obs\,i} \times (t - t_{last})) \times N_{obs\,i}^{<t_{last}>}$

Listing 2. Algorithm for calculating N

5.1.3 Third step: estimate QoS metrics

In the case that properties do not share contexts, the QoS metric associated with a property *prop* can be expressed, according to Section 3.4, as shown in (5.7). The nomenclature will be the same as what we have defined in that section.

$$QoS\ metric\ \equiv\ P(PROP \mid N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}) = \frac{\sum_{ctx_i} P(PROP, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>})}{\sum_{prop} \sum_{ctx_i} P(prop, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>})} \tag{5.7}$$

If we divide both parts of the fraction by the numerator (we will consider that there are no indeterminacies due to null probabilities), we obtain

$$P(PROP \mid N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}) = \left(1 + \frac{\sum_{ctx_i} P(\neg PROP, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>})}{\sum_{ctx_i} P(PROP, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>})}\right)^{-1} \tag{5.8}$$

Applying the definition of $P(prop, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>})$ shown in Equation (3.3) and assuming that $P(PROP) = P(\neg PROP)$, we can get (5.9). $LR_{obs\,i}$ is expressed in (5.10), where we have changed $P\left(N_{obs\,i}^{<t>} \left| CTX_i\right.\right)$ for $\left(qn_{obs\,i}^{<t>} \left| CTX_i\right.\right)$, that is, we have introduced the quantization of $N_{obs\,i}^{<t>}$ according to what we saw in Section 4.4.4.2.

$$P(PROP \mid N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}) = \frac{1}{1 + \prod_i LR_{obs\,i}} \tag{5.9}$$

$$LR_{obs\,i} = \frac{P(CTX_i \mid \neg PROP)\, P\left(qn_{obs\,i}^{<t>} \left| CTX_i\right.\right) + P(\neg CTX_i \mid \neg PROP)\, P\left(qn_{obs\,i}^{<t>} \left| \neg CTX_i\right.\right)}{P(CTX_i \mid PROP)\, P\left(qn_{obs\,i}^{<t>} \left| CTX_i\right.\right) + P(\neg CTX_i \mid PROP)\, P\left(qn_{obs\,i}^{<t>} \left| \neg CTX_i\right.\right)} \tag{5.10}$$

Finally, operating with logarithms, we can improve the numerical stability by converting products into sums.

$$P(PROP \mid N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}) = \left(1 + e^{\sum_i \ln\,(LR_{obs\,i})}\right)^{-1} \tag{5.11}$$

In the general case, where properties could have shared contexts, we saw in Section 3.4 that the QoS metric associated with a property $prop_j$ can be expressed as shown in (5.12).

$$\begin{aligned} QoS\ metric\ of\ prop_j\ \equiv\ P\left(PROP_j \left| N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}\right.\right) &= \sum_{\forall k \mid b_j=1} P(PROPS_k \mid N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}) \\ &= \frac{\sum_{\forall k \mid b_j=1} \sum_{ctx_i} P(PROPS_k, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>})}{\sum_k \sum_{ctx_i} P(PROPS_k, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>})} \end{aligned} \tag{5.12}$$

Note that this formulation is a generalization of (5.7), so that both are equivalent if $props \in \{PROPS_0, PROPS_1\}$, where $PROPS_0$ and $PROPS_1$ represent $\neg PROP$ and $PROP$ respectively. Analogous to what we did previously, if we divide by the numerator, expand $P(props, ctx_1, \dots, ctx_N, N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>})$ and apply logarithms, we end up with the expression in (5.13).

$$P\left(PROP_j \mid N_{obs\,1}^{<t>}, \dots, N_{obs\,N}^{<t>}\right) = \sum_{\forall k \mid b_j = 1} \left(1 + \sum_{l \neq k} e^{\sum_i \ln\left(LR_{obs\,i}^{<l,k>}\right)}\right)^{-1} \tag{5.13}$$

$$LR_{obs\,i}^{<l,k>} = \frac{P(CTX_i \mid PROPS_l)\, P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right) + P(\neg CTX_i \mid PROPS_l)\, P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)}{P(CTX_i \mid PROPS_k)\, P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right) + P(\neg CTX_i \mid PROPS_k)\, P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)} \tag{5.14}$$

The term $LR_{obs\,i}^{<l,k>}$ provides an instantaneous and relative measure of how likely it is that the evidence about a context condition, provided by an observation, can be explained by the optimal (or non-optimal) performance of the system for some properties. Specifically, the term contrasts two possible configurations of *props*: (1) $PROPS_k$, which includes the assumption that the system behaves optimally with respect to the property under consideration, versus (2) $PROPS_l$, which assumes that the system is not optimal with respect to that property. A $LR_{obs\,i}^{<l,k>}$ approaching 0 suggests a strong support for the presence of $PROPS_k$. On the contrary, if $LR_{obs\,i}^{<l,k>}$ tends to infinity, it greatly favours the presence of $PROPS_l$. Finally, a value equals to 1 has a neutral connotation, which implies that the current evidence provides no information about the property. See that, according to Equation (5.13), if all $LR_{obs\,i}^{<l,k>}$ were 1, the QoS estimate would be the baseline value, i.e. 0.5.

As seen in (5.15), we can further develop (5.14), firstly by multiplying and dividing by $P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right)$ and, secondly by using $N_{obs\,i}^{<t>}$ in place of $P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)/P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right)$ in accordance with the LR profile defined in (4.29) and the relationship in (4.35). The result has the advantage of depending on $N_{obs\,i}^{<t>}$ instead of $qn_{obs\,i}^{<t>}$.

$$\begin{aligned} LR_{obs\,i}^{<l,k>} &= \frac{P(CTX_i \mid PROPS_l) + P(\neg CTX_i \mid PROPS_l)\, P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)/P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right)}{P(CTX_i \mid PROPS_k) + P(\neg CTX_i \mid PROPS_k)\, P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)/P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right)} \\ &= \frac{P(CTX_i \mid PROPS_l) + P(\neg CTX_i \mid PROPS_l)\, e^{-\alpha N_{obs\,i}^{<t>}}}{P(CTX_i \mid PROPS_k) + P(\neg CTX_i \mid PROPS_k)\, e^{-\alpha N_{obs\,i}^{<t>}}} \end{aligned} \tag{5.15}$$

Let us elaborate $LR_{obs\,i}^{<l,k>}$ in a different way to see more clearly how this term behaves depending on $N_{obs\,i}^{<t>}$. As shown in (5.16), we can express Equation (5.14) in terms of a logistic function, denoted by $f(z)$, with z defined in (5.17).

$$\begin{aligned}
LR_{obs\,i}^{<l,k>} &= \frac{P(CTX_i \mid PROPS_l)\,P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right) + P(\neg CTX_i \mid PROPS_l)\,P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)}{P(CTX_i \mid PROPS_k)\,P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right) + P(\neg CTX_i \mid PROPS_k)\,P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)} \\
&= \left(\frac{P(CTX_i \mid PROPS_k)}{P(CTX_i \mid PROPS_l)} + \frac{P(\neg CTX_i \mid PROPS_k)}{P(CTX_i \mid PROPS_l)}\,\frac{P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)}{P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right)}\right)^{-1} \\
&+ \left(\frac{P(\neg CTX_i \mid PROPS_k)}{P(\neg CTX_i \mid PROPS_l)} + \frac{P(CTX_i \mid PROPS_k)}{P(\neg CTX_i \mid PROPS_l)}\,\frac{P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right)}{P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)}\right)^{-1} \\
&= \frac{P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_k)}\left(1 + \frac{P(\neg CTX_i \mid PROPS_k)}{P(CTX_i \mid PROPS_k)}\,\frac{P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)}{P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right)}\right)^{-1} \\
&+ \frac{P(\neg CTX_i \mid PROPS_l)}{P(\neg CTX_i \mid PROPS_k)}\left(1 + \frac{P(CTX_i \mid PROPS_k)}{P(\neg CTX_i \mid PROPS_k)}\,\frac{P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right)}{P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)}\right)^{-1} \\
&= \frac{P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_k)}\,\frac{1}{1+e^{-z}} + \frac{P(\neg CTX_i \mid PROPS_l)}{P(\neg CTX_i \mid PROPS_k)}\,\frac{1}{1+e^{z}} \\
&= \frac{P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_k)}\,f(z) + \frac{P(\neg CTX_i \mid PROPS_l)}{P(\neg CTX_i \mid PROPS_k)}\,(1 - f(z))
\end{aligned} \tag{5.16}$$

$$\begin{aligned}
z &= -\ln\left(\frac{P(\neg CTX_i \mid PROPS_k)}{P(CTX_i \mid PROPS_k)}\right) - \ln\left(\frac{P\left(qn_{obs\,i}^{<t>} \mid \neg CTX_i\right)}{P\left(qn_{obs\,i}^{<t>} \mid CTX_i\right)}\right) \\
&= -\ln\left(\frac{P(\neg CTX_i \mid PROPS_k)}{P(CTX_i \mid PROPS_k)}\right) - \ln\left(e^{-\alpha_{obs\,i}\,N_{obs\,i}^{<t>}}\right) \\
&= -\ln\left(\frac{P(\neg CTX_i \mid PROPS_k)}{P(CTX_i \mid PROPS_k)}\right) + \alpha_{obs\,i}\,N_{obs\,i}^{<t>}
\end{aligned} \tag{5.17}$$

Logistic functions have a sigmoid curve in the domain of all real numbers with a return value (y axis) that transitions from 0 to 1 monotonically. In our case, we have the sum of a logistic function and its reflection on the vertical axis (i.e., $1 - f(z)$), which represents two forces pulling in opposite directions: (1) evidence (in favour of the presence of the context condition) and (2) counterevidence (that contradicts the presence of the context condition). The first one is introduced by the term $f(z)$, which becomes more predominant as $N_{obs\,i}^{<t>}$ grows. Note that, as we accumulate more evidence, $LR_{obs\,i}^{<l,k>}$ will approach $P(CTX_i \mid PROPS_l)/P(CTX_i \mid PROPS_k)$, which will be less than 1 if the observation reinforces or greater than 1 if it undermines. On the other hand, the counterevidence is introduced by the term $1 - f(z)$, which reaches its maximum as $N_{obs\,i}^{<t>}$ decreases taking negative values. In the end, this term would make $LR_{obs\,i}^{<l,k>}$ approach $P(\neg CTX_i \mid PROPS_l)/P(\neg CTX_i \mid PROPS_k)$. However, as $N_{obs\,i}^{<t>} \geq 0$, $LR_{obs\,i}^{<l,k>}$ will never take this value, conversely it will be bounded between 1 (when $N_{obs\,i}^{<t>} = 0$) and $P(CTX_i \mid PROPS_l)/P(CTX_i \mid PROPS_k)$. Finally, mention that this alternative formulation will be

especially useful in a later section to address the estimation of QoS through neural networks. Listing 3 shows the proposed algorithm to estimate a QoS metric based on Equation (5.13).

algorithm estimate-qos **is**
  const: parameter α for each observation *(see Section 4.4.4),*
    a table of probabilities $P(ctx_i|props)$ for each observation *(see Section 4.4.3)*
  input: the property $prop$ for which we want the QoS estimate,
    the updated value of $N_{obs\,i}^{<t>}$ for each observation *(calculated with the algorithm in Listing 2)*
  output: *estimate*, QoS value for property $prop$
  $PROPS_{pos} \leftarrow (empty\ list)$
  $PROPS \leftarrow (empty\ list)$
  for each $props_j$ defined in $P(CTX_i|props)$ **do**
    if $props_j \mid prop = PROP$ **then**
      $PROPS_{pos}.insert(props_j)$
    end if
    $PROPS.insert(props_j)$
  end for
  $len_{pos} \leftarrow PROPS_{pos}.size$
  $len \leftarrow PROPS.size - 1$
  $LLR \leftarrow (new\ len_{pos} \times len\ matrix\ initialized\ to\ 0)$
  for each obs_i **do**
    $r \leftarrow 0$
    for each $props_k$ **in** $PROPS_{pos}$ **do**
      $c \leftarrow 0$
      for each $props_l$ **in** $PROPS$ **do**
        if $props_l \neq props_k$ **do**
          $LLR[r,c] \leftarrow LLR[r,c] + ln\left(LR_{obj\,i}^{<l,k>}\right)$ *($LR_{obj\,i}^{<l,k>}$ calculated according to Equation 5.15 or 5.16)*
          $c \leftarrow c + 1$
        end if
      end for
      $r \leftarrow r + 1$
    end for
  end for
  $estimate \leftarrow 0$
  for $r = 0$ to len_{pos} **do**
    $partial \leftarrow 0$
    for $c = 0$ to len **do**
      $partial \leftarrow partial + exp(LLR[r,c])$
    end for
    $estimate \leftarrow estimate + 1/(1 + partial)$
  end for
  return *estimate*

Listing 3. Algorithm for estimating QoS metrics

5.1.4 Example simulations

Next, we show some simulations based on the hospital robot example introduced in Section 3.3. They were executed with a Java implementation of the algorithms presented in the previous sections. The simulations cover a period of 7 days with estimates every 2 minutes, resulting in a total of 5040 estimates per simulation processed in ~600ms on an average computer.

5.1.4.1 Example 1: Bumping into someone

As we explained in Section 3.3, among other reasons, the robot would be considered safe if it does not bump into anyone. So, it makes sense to have an observation that is triggered every time a collision is detected, which will greatly undermine the belief that the robot is operating optimally in terms of safety. Table 13 outlines the details of this observation.

Table 13. Observation "bump into someone"

(PATTERN) A collision event is received	(PERSISTENCE) Long (7-day timeframe)	(REPETITION) None
(PROPERTY) Safety	(DIRECTION) Undermine	(STRENGTH) Very high

Figure 19 presents the results of the simulation, in which we can see how the QoS metric associated with safety changes after the robot collides with someone. The horizontal axis of the graph corresponds to the time elapsed since the start of the simulation and the vertical one to the QoS estimate. At the beginning, the QoS estimate is 0.5, which represents the metric baseline. The observation occurs 6 hours from the start and produces a sharp drop in the estimate. Since there are no more occurrences, the metric will tend to return to 0.5, but very slowly as this observation has a long persistence.

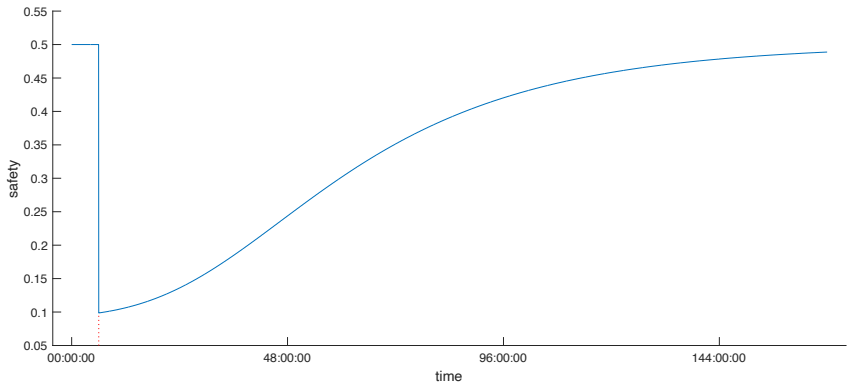


Figure 19. Safety when the robot collides with someone

5.1.4.2 Example 2: Acceptance among patients

The degree of engagement can be determined by observing how patients behave with the robot. In particular, we consider that it is a sign of acceptance if people call the robot by its name. Thus, every time this happens, an observation will reinforce the belief that the robot is operating optimally in terms of user engagement. Table 14 outlines the details of this observation.

Table 14. Observation "the robot is called by its name"

(PATTERN) People call the robot by its name	(PERSISTENCE) Some hours	(REPETITION) 5
(PROPERTY) Engagement	(DIRECTION) Reinforce	(STRENGTH) Low

We have run the simulation with 200 random occurrences of the observation generated uniformly throughout the 7 days that have been simulated. Figure 20 shows 10 hours of the simulation, in which we can see the fluctuations in the QoS estimate produced by the occurrences (marked on the graph with dotted lines). The evidence is consolidated after 5 consecutive repetitions, at which point the observation will reach its maximum effect on the estimate, whose value will be around 0.6. This observation provides weak evidence since STRENGH is low.

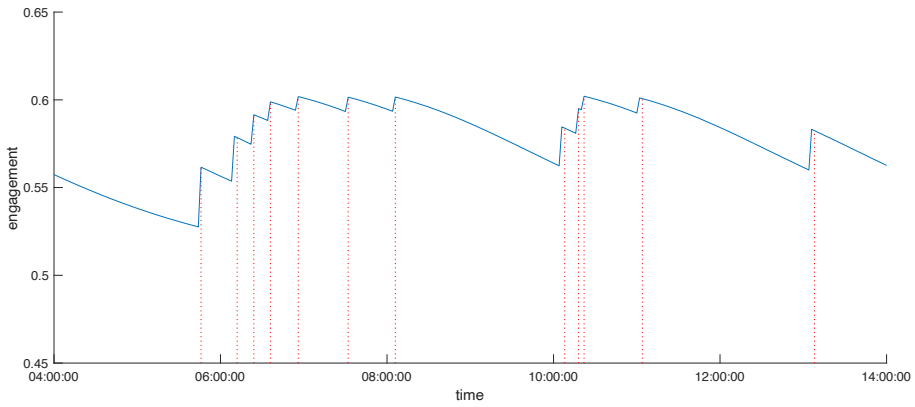


Figure 20. User engagement by observing the acceptance among patients

5.1.4.3 Example 3: Stranded without battery

It is a safety concern to have the robot stranded without battery in the middle of a hospital corridor. Thus, an observation will be triggered whenever the battery is below 1% and the robot is not in the docking station. Once the robot reaches this situation, the observation is repeated

periodically every few minutes until the situation ceases, either because the robot itself manages to reach the charging station or because the technical staff intervenes. As in the previous example, the evidence will not be consolidated until 5 consecutive repetitions are detected, at which point the observation achieves its maximum effect. Table 15 outlines the details of this observation.

Table 15. Observation "stranded without battery"

(PATTERN) The battery level is lower than 1% and not in DOCKED state	(PERSISTENCE) Medium (7-day timeframe)	(REPETITION) 5
(PROPERTY) Safety	(DIRECTION) Undermine	(STRENGTH) High

Figure 21 shows the first 18 hours of the simulation. The observation occurs at 6:00:00 and five more times in 15-minute intervals. These occurrences have been marked on the graph with dotted lines. See that the estimate gradually decreases to 0.22 as the occurrences arrive, and only once there are not more occurrences, the estimate returns to the reference value after about 60 hours.

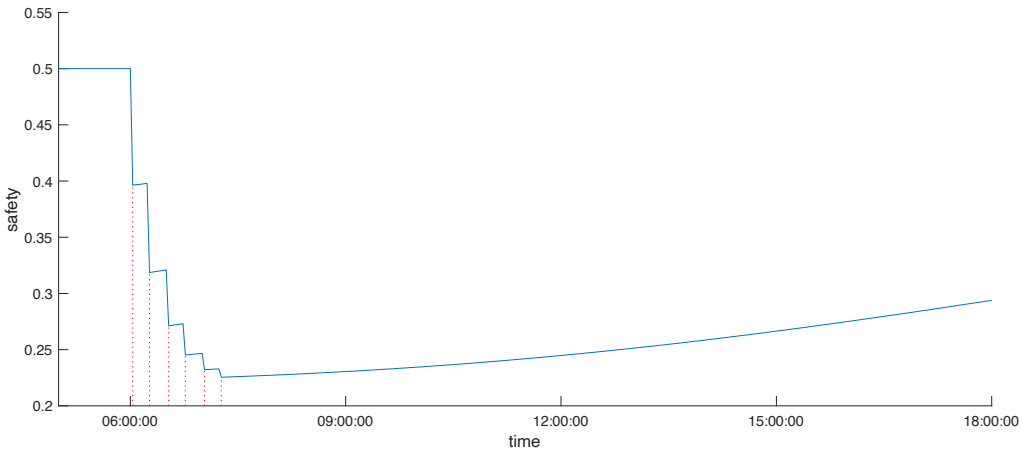


Figure 21. Safety when the robot is stranded without battery

In the examples we have seen, all the observations had an impact on individual properties. Now, let us imagine that the observation "stranded without battery" does not only impact on safety but also on a new property called "power management", which will help us to illustrate the effect of shared contexts. This property shows how optimally the system manages its energy, so that, if the robot gets stranded without battery, the observation would provide evidence that the system is not performing well in terms of power management (as well as

safety). The uncertainty will be greater as the observation no longer points to a single cause but may instead be the visible effect of various scenarios: the system performing poorly in safety, in power management, or in both. Due to this uncertainty, the evidence will lose strength. Figure 22 presents the new QoS estimates for safety, where the dashed line represents the previous results to facilitate comparison. The simulation has been carried out considering that the observation has an impact on power management in the same way as on safety (same direction and strength). Because of this, the QoS estimates for power management are identical to those for safety.

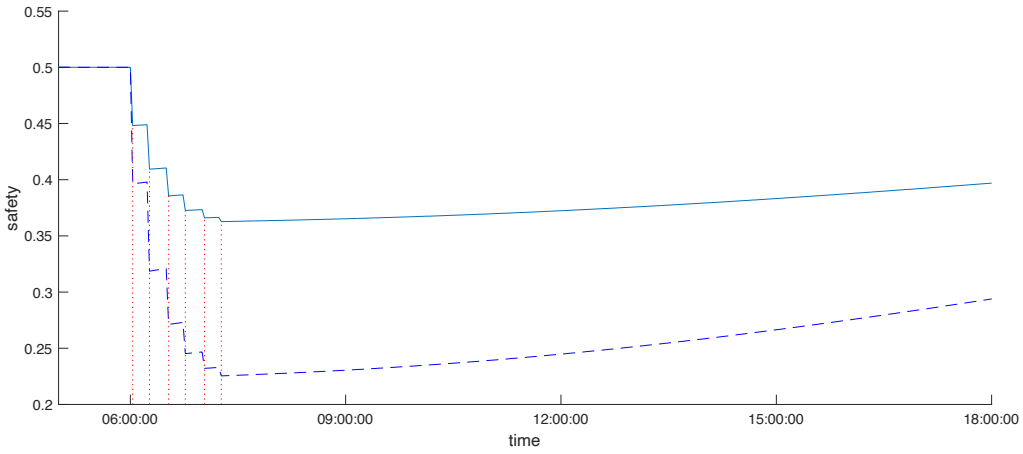


Figure 22. Safety when the observation affects two properties. The dashed line indicates the result shown previously

5.2 Statistics on QoS metrics

Once we have QoS estimates, we can treat them like any other type of data, which includes applying statistics to extract valuable information for the use case. The purpose of this section is twofold. First, we will develop some useful measures to exploit the potential of QoS estimates and, second, we will provide designers with some advice on how to check and improve their QoS specifications.

5.2.1 Central tendency and variability

When it comes to variables that produce a sequence of samples over time, it is common to measure their central tendency to obtain a single representative value. Thus, we can compute

the mode, the median or the mean of the QoS estimates in the period that has being considered to examine the system. Another option is to apply a moving average [49], to smooth out short-term fluctuations and highlight longer-term trends. With these methods, we will be able to obtain a general picture of how the system works in a period of time for the specified non-functional properties.

Aside from the central tendency, we should also take into account variability, especially if we want to compare QoS metrics, for example, to identify the non-functional property in which the system performs worst. In this sense, by simply measuring the range (that is, the highest and the lowest value), we can find flaws in the specification. Let us think we have designed our specification with two properties, where the first one only has observations that reinforce it and the second one only has observations that undermine it. According to the proposed probabilistic model, the QoS metric that will result from the first property would be defined in the range [0.5, 1], while the second metric would be in [0, 0.5]. As a result, it would not be fair to compare the two QoS metrics directly to decide on which property the system should focus in order to optimize its performance. In general, the distribution of the values of a metric does not only depend on the evidence that is produced due to a good or bad behaviour of the system, but a poor design of the observations can introduce a significant bias.

As for the application of any statistics to QoS metrics, it should be noted that users are the ones who decide when to produce a new estimate (through the execution of the algorithm in Listing 3), so they should choose the monitoring strategy that best suits their requirements (e.g., polling every two minutes). Since statistics are often linked to the monitoring, our approach is that, with the data obtained, users are responsible for implementing the statistics that they consider relevant to their use case. Nevertheless, to make things easier, we are going to provide two basic measurements to capture the central tendency and the variability of each QoS metric. Moreover, note that the statistics discussed in the following sections are also considered because they cannot be derived directly from QoS estimates.

Next, we are going to pay attention to the range and the Time-Exponential Moving Average (TEMA) [50] of each QoS metric. The latter is the version of the Exponential Moving Average for unevenly spaced time series. As QoS estimates may or may not be regular in time,

depending on the monitoring strategy implemented by the user, TEMA seems to be flexible enough to allow measuring the central tendency regardless of time changes between samples. Even so, we must keep in mind that the monitoring strategy of the user can also impact on the precision of the indicators. For instance, it is not the same to produce statistics from 10 samples than from 100 for the same period. Equation (5.18) shows the definition of TEMA considered in this work, where X_{t_i} is the sample at time t_i (in our case a QoS estimate produced at t_i) and $1/\beta$ (with $\beta > 0$) establishes the average duration over which a sample contributes, such that the higher the term $1/\beta$, the smoother the moving average. Listing 4 extends the algorithm in Section 5.1.3 to add the computation of the range and the TEMA of a QoS metric.

$$TEMA_t = \begin{cases} X_{t_0} & if\ t = t_0 \\ e^{-\beta\,(t_i - t_{i-1})}\, TEMA_{t_{i-1}} + \left(1 - e^{-\beta\,(t_i - t_{i-1})}\right) X_{t_i} & if\ t = t_i > t_0 \end{cases} \tag{5.18}$$

algorithm estimate-qos **is**
  const: β, parameter to stablish the average duration of the samples for the TEMA
  ...
  global: *max_range_stat*, maximum QoS value recorded *(initialized to 0.5)*,
  min_range_stat, minimum QoS value recorded *(initialized to 0.5)*,
  tema_stat, moving average of the QoS estimates *(initialized to 0.5)*,
  t_{prev}, time previous update *(initialized to -1)*
  input: t, current time, ...
  output: QoS estimate for property $prop$,

  (calculation of the estimate according to Listing 3)

  if *max_range_stat* < *estimate* **then**
    max_range_stat ← *estimate*
  end if
  if *min_range_stat* > *estimate* **then**
    min_range_stat ← *estimate*
  end if
  if t_{prev} = -1 **then**
    tema_stat ← *estimate*
  else
    $tema\_stat \leftarrow exp\left(-\beta\,(t - t_{prev})\right) \times (tema\_stat - estimate) + estimate$ *(derived from Equation 5.18)*
  end if
  $t_{prev} \leftarrow t$
  return *estimate*

Listing 4. Extension of the algorithm for calculating QoS estimates

5.2.2 Temporal mean of $N_{obs\,i}^{<t>}$

Measuring the real environment of the system can provide us with valuable information to improve our QoS specification. For instance, as a general rule, observations that are very frequent can saturate a metric if their occurrences are too strong. In this sense, watching the frequency of the observations can help us to check if we have assigned appropriate attributes to the observations (i.e., persistence, repetition and strength). Note that these attributes set the overall influence of an occurrence, in such a way that the impact of an observation grows as we increase its persistence and strength while reducing the number of repetitions.

In the proposed probabilistic network, the environment is represented through the term $N_{obs\,i}^{<t>}$, that is, the instantaneous average number of active occurrences of each observation, which was introduced in Section 3.4 and developed in Section 5.1.2. This measurement is interesting because it captures the dynamics behind the estimation of QoS metrics. In essence,

- $N_{obs\,i}^{<t>}$ is a counter that increases by one every time a new occurrence arrives;
- after an occurrence, the counter gradually loses one unit over time. The higher the persistence, the longer it takes to decrease;
- the maximum effect of an occurrence is reached when $N_{obs\,i}^{<t>}$ is greater than or equal to the number of repetitions + 1.

Examples of the implications that we could learn from $N_{obs\,i}^{<t>}$ are:

- Observations that are very persistent and frequent can show a high $N_{obs\,i}^{<t>}$.
- Those observations whose $N_{obs\,i}^{<t>}$ never reaches the number of repetitions + 1 could be underestimated.
- If an observation has a time between arrivals greater than its persistence, $N_{obs\,i}^{<t>}$ will not be greater than 1 and, therefore, the number of repetitions should be set to 0.

In addition, we can also use $N_{obs\,i}^{<t>}$ to discover dependencies between observations. According to the premises considered in the design of the proposed probabilistic network (see Section 3.4.1), an observation is assumed to occur independently of any other. However, since the observations could have common context sources, this assumption might not hold. In order

to check it, we could calculate the Pearson correlation coefficient [49] on the values of $N_{obs\,i}^{<t>}$ belonging to two different observations to see if they are correlated. If two observations were strongly correlated, we could consider reducing them to just one. Note that observations with very different persistence values could hide possible correlations, so we recommend using the same persistence when performing this test.

In this section, we propose the use of the temporal mean of $N_{obs\,i}^{<t>}$ as a more representative measure to analyse the effect of the environment. To develop a method to obtain this temporal mean, let us consider a sequence of values $N_{obs\,i}^{<t>}$ covering a period of time. Our approach will be to divide this period into a set of smaller periods, such that the mean could be formed by the weighted sum of the expected values in each of these fragments of time. Specifically, we will take into account the period between two adjacent occurrences of the observation, so each time a new occurrence arrives, (1) we have to calculate the expected value of $N_{obs\,i}^{<t>}$ between the previous occurrence and the new one, and (2) we have to update the global temporal mean with this result. Next, we will delve into the formulation of the first point.

Let $E_T\left[N_{obs\,i}^{<t>}\right]$ be the expected value of $N_{obs\,i}^{<t>}$ over a period of time T between two occurrences. According to Equation (5.6), the range of values for $N_{obs\,i}^{<t>}$ would be defined in the interval $(e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}, N_{obs\,i}^{<t_{occ}>}]$, being t_{occ} the time of the first occurrence. $N_{obs\,i}^{<t>}$ is a decreasing monotonic function in that interval, it will start with $N_{obs\,i}^{<t_{occ}>}$ with the first occurrence and end with $e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}$ just before the second. Note that we have not included the second occurrence in the interval, which is why we considered Equation (5.6) instead of (5.4).

To calculate $E_T\left[N_{obs\,i}^{<t>}\right]$, we must solve the following integral, where $f_N(n)$ is the probability density function (PDF) of $N_{obs\,i}^{<t>}$. The limits of the integral are defined by the previous interval.

$$E_T[N_{obs\,i}^{<t>}] = \int_{e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} n\,f_N(n)dn \tag{5.19}$$

In order to obtain $f_N(n)$, first, we have to determine the cumulative distribution function (CDF) of $N_{obs\,i}^{<t>}$. For that, we assume that the time elapsed from the first occurrence is a uniformly

distributed variable, denoted by τ, with values in the interval [0, T] and CDF as seen in (5.20). Taking into account (5.6) and (5.20), we can develop the CDF of $N_{obs\,i}^{<t>}$ as shown in (5.21).

$$\mathrm{F}_{\tau}(t') = P(\tau \leq t') = \frac{t'}{T} \tag{5.20}$$

$$F_N(n) = P\left(N_{obs\,i}^{<t_{occ}+\tau>} \leq n\right) = P\left(e^{-\mu\tau}\,\mathrm{N}_{\mathrm{obs\,i}}^{<t_{occ}>} \leq n\right) = 1 - P\left(\tau \leq \frac{-1}{\mu}\ln\left(\frac{\mathrm{n}}{\mathrm{N}_{\mathrm{obs\,i}}^{<t_{occ}>}}\right)\right) = \frac{1}{\mu T}\ln\left(\frac{\mathrm{n}}{\mathrm{N}_{\mathrm{obs\,i}}^{<t_{occ}>}}\right) + 1 \tag{5.21}$$

$$where\ n \in \mathbb{R} \mid \left(e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}\right) \leq n \leq N_{obs\,i}^{<t_{occ}>}$$

Once we have the CDF, we can compute the PDF of $N_{obs\,i}^{<t>}$, and finally, from that, the expected value $E_T\left[N_{obs\,i}^{<t>}\right]$. This process is shown in (5.22) and (5.23) respectively.

$$f_N(n) = \frac{dF(n)}{dn} = \frac{d}{dn}\left(\frac{1}{\mu T}\ln\left(\frac{\mathrm{n}}{\mathrm{N}_{\mathrm{obs\,i}}^{<t_{occ}>}}\right) + 1\right) = \frac{1}{\mu T n} \tag{5.22}$$

$$E_T[N_{obs\,i}^{<t>}] = \int_{e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} n\,f_N(n)dn = \frac{1}{\mu T}\left[N_{obs\,i}^{<t_{occ}>} - e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}\right] = \frac{N_{obs\,i}^{<t_{occ}>}}{\mu T}(1 - e^{-\mu T}) \tag{5.23}$$

By applying Equation (5.23) each time a new occurrence arrives, we will be able to calculate the expected value of the period between the previous occurrence and just before the new one. Thus, if we run this approach repeatedly over time with each occurrence, we will get a time series of expected values, which will probably be unevenly spaced since occurrences are not usually regular in time. The final temporal mean is the result of the Time-Exponential Moving Average (TEMA) [50] on the expected values. See its definition in Equation (5.18).

Finally, the following listings show the implementation of this statistic. Listing 5 extends the first algorithm presented in Section 5.1.2 to include the recurrent update of the temporal mean of $N_{obs\,i}^{<t>}$ every time a new occurrence arrives, while Listing 6 only applies whenever we want to obtain an updated value of the statistic regardless of whether or not there has been an occurrence.

algorithm update-evidence-with-new-occurrence-obs_i **is**

const: $\mu_{obs\,i} \leftarrow -\ln(0.01)/persistence_{obs\,i}$ *($persistence_{obs\,i}$ in seconds see Equation (4.27))*,
β, parameter to stablish the average duration of the samples for the TEMA

global: t_{prev}, time of the previous occurrence in seconds *(initialized to 0)*,
$N_{obs\,i}^{<t_{prev}>}$, update of N for the previous occurrence *(initialized to 0)*,
$tema\_stat\_N_{obs\,i}$, temporal mean of N for the observation *i (initialized to NULL)*

input: t_{new}, time of the new occurrence in seconds

output: $N_{obs\,i}^{<t_{new}>}$, updated value of N for the observation,

precondition: $t_{new} > t_{prev}$

$N_{obs\,i}^{<t_{new}>} \leftarrow 1 + exp\left(-\mu_{obs\,i} \times \left(t_{new} - t_{prev}\right)\right) \times N_{obs\,i}^{<t_{prev}>}$

$expected \leftarrow N_{obs\,i}^{<t_{prev}>} / \left(\mu_{obs\,i} \times \left(t_{new} - t_{prev}\right)\right) \times (1 - exp\left(-\mu_{obs\,i} \times \left(t_{new} - t_{prev}\right)\right)$

if $tema\_stat\_N_{obs\,i} = NULL$ **then**
  $tema\_stat\_N_{obs\,i} \leftarrow expected$
else
  $tema\_stat\_N_{obs\,i} \leftarrow exp\left(-\beta\left(t_{new} - t_{prev}\right)\right) \times (tema\_stat\_N_{obs\,i} - expected) + expected$
end if

$t_{prev} \leftarrow t_{new}$

$N_{obs\,i}^{<t_{prev}>} \leftarrow N_{obs\,i}^{<t_{new}>}$

return $N_{obs\,i}^{<t_{new}>}$

Listing 5. Extended algorithm for updating the evidence with a new occurrence

algorithm get-average-N-obs_i **is**

const: $\mu_{obs\,i} \leftarrow -\ln(0.01)/persistence_{obs\,i}$ *($persistence_{obs\,i}$ in seconds see Equation (4.27))*
β, parameter to stablish the average duration of the samples for the TEMA

global: $tema\_stat\_N_{obs\,i}$, temporal mean of N for the last occurrence *(updated in Listing 5)*
$N_{obs\,i}^{<t_{prev}>}$, N for the last occurrence *(updated in Listing 5)*
t_{prev}, time of the last occurrence in seconds *(updated in Listing 5)*

input: t, current time in seconds

output: temporal mean of N at time *t*

precondition: $t > t_{last}$

$expected \leftarrow N_{obs\,i}^{<t_{prev}>} / \left(\mu_{obs\,i} \times \left(t - t_{prev}\right)\right) \times (1 - exp\left(-\mu_{obs\,i} \times \left(t - t_{prev}\right)\right)$

if $tema\_stat\_N_{obs\,i} = NULL$ **then**
  $result \leftarrow expected$
else
  $result \leftarrow exp\left(-\beta\,(t - t_{last})\right) \times (tema\_stat\_N_{obs\,i} - expected) + expected$
end if

return $result$

Listing 6. Algorithm that returns an updated value of the statistic

5.2.3 Contribution of the observations

When analysing the resulting QoS estimates, it is practical to know the influence that each of the observations has had on them. Unlike the active number of occurrences shown in the previous section, where the information is limited to the individual scope of an observation, in this section we will see how to contextualize the effect of an observation along with the rest. For example, after a period of operation, we will be able to identify if an observation predominated over the rest or if, on the contrary, all contributed more or less equally. Next, we will develop a method to quantify the average contribution of each observation to a QoS estimate. Note that this method will also allow us to establish a ranking of the most influential observations for a QoS metric over a period of time.

In Section 5.1.3, we derived Equation (5.13) to compute QoS estimates. In that expression, the term $LR_{obs\,i}^{<l,k>}$ provides a relative measure of how discriminating a context condition associated with an observation is. The more discriminating the greater the contribution of an observation. Since the contribution of $LR_{obs\,i}^{<l,k>}$ becomes more important as $N_{obs\,i}^{<t>}$ increases, its maximum effect will be reached when it equals $P(CTX_i \mid PROPS_l)/P(CTX_i \mid PROPS_k)$, as we saw in Section 5.1.3. Note that this value will be in the range (0, 1) if the observation reinforces or (1, +∞) if it undermines. To avoid this range asymmetry, it is convenient to use a logarithmic scale, so that if the observation reinforces the value is in (-∞, 0) or (0, +∞) if it undermines. On the other hand, the least significant value for $LR_{obs\,i}^{<l,k>}$ is 1 (0 in a logarithmic scale), which appears when $N_{obs\,i}^{<t>}$ is 0.

Taking into account the above, we propose Equation (5.24) to define the instantaneous contribution that an observation produces (in relation to $PROPS_k$ and $PROPS_l$). Note that $C_{obs\,i}^{<l,k>}$ is expressed in terms of absolute value since it does not distinguish between evidence that reinforces or undermines, but only the extent of the influence.

$$C_{obs\,i}^{<l,k>} = \left|ln\left(LR_{obs\,i}^{<l,k>}\right)\right| \tag{5.24}$$

$C_{obs\,i}^{<l,k>}$ provides a partial measure relative to $PROPS_k$ and $PROPS_l$. In order to obtain the overall influence of an observation on a property $prop_j$, we need to add all the partial contributions that have some effect on the property, as shown in (5.25).

$$C_{obs\ i}^{<prop_j>} = \sum_{\forall k | b_j = 1} \sum_{l \neq k} C_{obs\ i}^{<l,k>} \tag{5.25}$$

However, $C_{obs\ i}^{<prop_j>}$ continues to present the contribution at a specific point in time, so we require a temporal mean to provide a more representative measure over a period. In particular, we are going to follow an approach similar to that applied in the previous section, in which we build this temporal mean from the expected values of $C_{obs\ i}^{<prop_j>}$ produced during any two consecutive occurrences of the observation. Therefore, each time a new occurrence arrives, (1) we calculate the expected value of $C_{obs\ i}^{<prop_j>}$ between the previous occurrence and the new one, and (2) we update the temporal mean with this result. Next, we will delve into the formulation of the first point.

Given the "linearity of expectation", the expected value of $C_{obs\ i}^{<prop_j>}$ over a period T can be reduced to the sum of the expectations of $C_{obs\ i}^{<l,k>}$ as displayed in (5.26).

$$E_T\left[C_{obs\ i}^{<prop_j>}\right] = \sum_{\forall k | b_j = 1} \sum_{l \neq k} E_T\left[C_{obs\ i}^{<l,k>}\right] \tag{5.26}$$

In addition, thanks to the law of the unconscious statistician (LOTUS), we can express the expected value $E_T\left[C_{obs\ i}^{<l,k>}\right]$ in terms of the probability distribution of the variable $N_{obs\ i}^{<t>}$. Recall that $LR_{obs\ i}^{<l,k>}$ is a function of $N_{obs\ i}^{<t>}$ as stated by Equation (5.15). So, to calculate $E_T\left[C_{obs\ i}^{<l,k>}\right]$ based on the distribution of $N_{obs\ i}^{<t>}$ during T, we should solve the following integral, where $f_N(n)$ is the probability density function of $N_{obs\ i}^{<t>}$, derived in (5.22). The limits of the integral were specified in the previous section based on the time of the first occurrence t_{occ}.

$$E_T\left[C_{obs\ i}^{<l,k>}\right] = \int_{e^{-\mu T}\ N_{obs\ i}^{<t_{occ}>}}^{N_{obs\ i}^{<t_{occ}>}} \left|ln\left(LR_{obs\ i}^{<l,k>}\right)\right| f_N(n)\ dn \tag{5.27}$$

In (5.28), we take the absolute value out of the integral since the term $ln\left(LR_{obs\ i}^{<l,k>}\right)$ only shows negative values with reinforcing observations, in which case the values are always less than or equal to 0 for the entire range of $N_{obs\ i}^{<t>}$. Besides, $f_N(n)$ only returns positive values.

$$\int_{e^{-\mu T}\ N_{obs\ i}^{<t_{occ}>}}^{N_{obs\ i}^{<t_{occ}>}} \left|ln\left(LR_{obs\ i}^{<l,k>}\right)\right| f_N(n)\ dn = \left| \int_{e^{-\mu T}\ N_{obs\ i}^{<t_{occ}>}}^{N_{obs\ i}^{<t_{occ}>}} ln\left(LR_{obs\ i}^{<l,k>}\right)\ f_N(n)\ dn \right| \tag{5.28}$$

By taking into account Equation (5.15), below we state $ln\left(LR_{obs\,i}^{<l,k>}\right)$ as the difference of two functions in terms of the integration variable n.

$$ln\left(LR_{obs\,i}^{<l,k>}\right) = llr_l(n) - llr_k(n)$$

$$\begin{aligned} &\text{where } llr_l(n) = ln(P(CTX_i \mid PROPS_l) \; + P(\neg CTX_i \mid PROPS_l)\, e^{-\alpha n}) \\ &\text{and } llr_k(n) = ln(P(CTX_i \mid PROPS_k) \; + P(\neg CTX_i \mid PROPS_k)\, e^{-\alpha n}) \end{aligned} \tag{5.29}$$

Then the integral in (5.28) can be written as follows,

$$\int_{e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} ln\left(LR_{obs\,i}^{<l,k>}\right)\, f_N(n)\, dn = \int_{e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} llr_l(n)\, f_N(n)\, dn - \int_{e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} llr_k(n)\, f_N(n)\, dn \tag{5.30}$$

Since $llr_l(n)$ and $llr_k(n)$ are the same function, one expressed in terms of $PROPS_l$ and the other in terms of $PROPS_k$, next we are going to develop one of them and the same will apply to the other. Note that we will consider all probabilities to be strictly greater than 0 and less than 1 to avoid indeterminacies.

$$\begin{aligned} llr_l(n) = \; & ln(P(CTX_i \mid PROPS_l) \; + P(\neg CTX_i \mid PROPS_l)\, e^{-\alpha n}) \\ &= ln(P(CTX_i \mid PROPS_l) \; + (1 - P(CTX_i \mid PROPS_l))\, e^{-\alpha n}) \\ &= P(CTX_i \mid PROPS_l)\left(1 + \frac{1 - P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)} e^{-\alpha n}\right) \\ &= P(CTX_i \mid PROPS_l)\left(1 + e^{ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right) - \alpha n}\right) \\ &= ln(P(CTX_i \mid PROPS_l)) + ln\left(1 + e^{ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right) - \alpha n}\right) \end{aligned} \tag{5.31}$$

By using the result in (5.31) and substituting $f_N(n)$ by the expression in (5.22), we can decompose the following integral in two parts.

$$\begin{aligned} &\int_{e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} llr_l(n)\, f_N(n)\, dn \\ &\qquad = \int_{e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} ln(P(CTX_i \mid PROPS_l))\, \frac{1}{\mu\, T\, n}\, dn \\ &\qquad + \int_{e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} ln\left(1 + e^{ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right) - \alpha n}\right) \frac{1}{\mu\, T\, n}\, dn \end{aligned} \tag{5.32}$$

The solution of the first part is

$$\begin{aligned}\int_{e^{-\mu T} N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} ln(P(CTX_i \mid PROPS_l))\,\frac{1}{\mu\,T\,n}\,dn &= \frac{ln(P(CTX_i \mid PROPS_l))}{\mu\,T}\left(ln\left(N_{obs\,i}^{<t_{occ}>}\right) - ln\left(e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}\right)\right)\\ &= \frac{ln(P(CTX_i \mid PROPS_l))}{\mu\,T}\left(ln\left(\frac{N_{obs\,i}^{<t_{occ}>}}{e^{-\mu T}\,N_{obs\,i}^{<t_{occ}>}}\right)\right) = \frac{ln(P(CTX_i \mid PROPS_l))}{\mu\,T}ln(e^{\mu T})\\ &= ln(P(CTX_i \mid PROPS_l))\end{aligned} \tag{5.33}$$

As for the second part, we will have to make the approximation shown in (5.34) to be able to approach it analytically.

$$\begin{aligned}ln(1+e^{u}) &= \max(u,0) + ln\left(1+e^{-|u|}\right), \qquad u \in \mathbb{R}\\ ln\left(1+e^{-|u|}\right) &\approx ln(2)\,e^{-0.85\,|u|}\end{aligned} \tag{5.34}$$

This approximation will require case decision depending on whether the value of u is less than or greater than 0, being $u = ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right) - \alpha n$. Since it is more convenient to express the cases in terms of the integration variable n, the value of n corresponding to the breakpoint $(u = 0)$ is

$$0 = ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right) - \alpha\,n_b \Longrightarrow n_b = \frac{1}{\alpha}ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right) \tag{5.35}$$

Let us solve the integral case by case:

- Case 1: if $n \le n_b$, or equivalently $n_1 \le n_b$,

$$\begin{aligned}I_{n\le n_b}(n_0,n_1) &= \int_{n_0}^{n_1} ln\left(1+e^{ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)-\alpha n}\right)\frac{1}{\mu\,T\,n}dn\\ &\approx \int_{n_0}^{n_1} ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)\frac{1}{\mu\,T\,n}dn - \int_{n_0}^{n_1}\frac{\alpha}{\mu\,T}dn\\ &+ \int_{n_0}^{n_1}\frac{ln(2)}{\mu\,T\,n}e^{-0.85\left(ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)-\alpha n\right)}dn\\ &= ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)\frac{\left(ln(n_1)-ln(n_0)\right)}{\mu\,T} - \frac{\alpha}{\mu\,T}(n_1-n_0)\\ &+ \frac{ln(2)}{\mu\,T}\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)^{-0.85}\left(E_i(0.85\,\alpha\,n_1) - E_i(0.85\,\alpha\,n_0)\right)\end{aligned} \tag{5.36}$$

- Case 2: if $n \geq n_b$, or equivalently $n_0 \geq n_b$,

$$I_{n\geq n_b}(n_0, n_1) = \int_{n_0}^{n_1} ln\left(1 + e^{ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)-\alpha n}\right)\frac{1}{\mu\, T\, n}dn$$
$$\approx \int_{n_0}^{n_1} \frac{ln(2)}{\mu\, T\, n} e^{0.85\left(ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)-\alpha n\right)}dn \tag{5.37}$$
$$= \frac{ln(2)}{\mu\, T}\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)^{0.85}\left(E_1(0.85\,\alpha\, n_0) - E_1(0.85\,\alpha\, n_1)\right)$$

- Case3: if $n_0 < n_b < n_1$,

$$I(n_0, n_1) = I_{n\leq n_b}(n_0, n_b) + I_{n\geq n_b}(n_b, n_1) \tag{5.38}$$

The limits of integration in (5.36), (5.37) and (5.38) are the ones we have already defined previously, i.e., $n_0 = e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>}$ and $n_1 = N_{obs\,i}^{<t_{occ}>}$. Moreover, the functions $E_1(x)$ and $E_i(x)$ in (5.36) and (5.37) are exponential integrals [51].

Putting it all together, the integral in (5.32) will look like this:

- Case 1: if $N_{obs\,i}^{<t_{occ}>} \leq n_b$,

$$\int_{e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} llr_l(n)\, f_N(n)\, dn \approx ln(P(CTX_i \mid PROPS_l)) + ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right) - \frac{\alpha\left(1-e^{-\mu T}\right) N_{obs\,i}^{<t_{occ}>}}{\mu\, T} +$$
$$\frac{ln(2)}{\mu\, T}\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)^{-0.85}\left(E_i\left(0.85\,\alpha\, N_{obs\,i}^{<t_{occ}>}\right) - E_i\left(0.85\,\alpha\, e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>}\right)\right) \tag{5.39}$$

- Case 2: if $e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>} \geq n_b$,

$$\int_{e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} llr_l(n)\, f_N(n)\, dn \approx ln(P(CTX_i \mid PROPS_l)) +$$
$$\frac{ln(2)}{\mu\, T}\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)^{0.85}\left(E_1\left(0.85\,\alpha\, e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>}\right) - E_1\left(0.85\,\alpha\, N_{obs\,i}^{<t_{occ}>}\right)\right) \tag{5.40}$$

- Case 3: if $e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>} < n_b < N_{obs\,i}^{<t_{occ}>}$,

$$\int_{e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>}}^{N_{obs\,i}^{<t_{occ}>}} llr_l(n)\, f_N(n)\, dn \approx ln(P(CTX_i \mid PROPS_l)) +$$
$$ln\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)\frac{\left(ln(n_b) - ln\left(e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>}\right)\right)}{\mu\, T} - \frac{\alpha\left(1-e^{-\mu T}\right) N_{obs\,i}^{<t_{occ}>}}{\mu\, T} +$$
$$\frac{ln(2)}{\mu\, T}\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)^{-0.85}\left(E_i(0.85\,\alpha\, n_b) - E_i\left(0.85\,\alpha\, e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>}\right)\right) +$$
$$\frac{ln(2)}{\mu\, T}\left(\frac{1-P(CTX_i \mid PROPS_l)}{P(CTX_i \mid PROPS_l)}\right)^{0.85}\left(E_1(0.85\,\alpha\, n_b) - E_1\left(0.85\,\alpha\, N_{obs\,i}^{<t_{occ}>}\right)\right) \tag{5.41}$$

Since the functions $E_1(x)$ and $E_i(x)$ present a singularity at $x = 0$, the expressions above can only be evaluated if $e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>} > 0$. Note that we could substitute $e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>}$ for

$max\left(e^{-\mu T}\, N_{obs\,i}^{<t_{occ}>}, \delta\right)$ to avoid problems, where δ is a small positive number (e.g. 0.01). Regarding the implementation of $E_1(x)$ and $E_i(x)$, these functions does not have a closed-form solution, so we will use an approximation. For $E_1(x)$ we consider the method proposed by Barry et al. [52]. As for $E_i(x)$, we opt for a lookup table, because the range of values over which this function will be evaluated is limited. More specifically, according to the cases in (5.36) and (5.38), we will need to compute $E_i(x)$ if $0 < n \leq n_b$, which, in terms of x, corresponds to $0 < x \leq 0.85\alpha n_b$, or $0 < x \leq 0.85\ ln\left(\frac{1-P(CTX_i \mid PROPS_j)}{P(CTX_i \mid PROPS_j)}\right)$ by substituting n_b. This interval will show its maximum width when the probability $P(CTX_i \mid PROPS_j)$ is minimum, that is, 0.0909 considering Section 4.4.3. As a result, we get the interval $0 < x \leq 1.9573$. Empirically, we have verified that a lookup table with 200 values of $E_i(x)$ evenly covering this interval provides a mean relative error around 1.5%, which seems acceptable.

Finally, the expected value $E_T\left[C_{obs\,i}^{<l,k>}\right]$ can be obtained by computing the partial integrals in (5.30) considering (5.39), (5.40) and (5.41) applied to $P(CTX_i \mid PROPS_l)$ and $P(CTX_i \mid PROPS_k)$. Thus, each time a new occurrence arrives, we will be able to determine $E_T\left[C_{obs\,i}^{<l,k>}\right]$ and from that $E_T\left[C_{obs\,i}^{<prop_j>}\right]$. By running this approach repeatedly over time with each occurrence of the observation, we will get a time series of expected values, which will probably be unevenly spaced since occurrences are not usually regular in time. The temporal mean of $E_T\left[C_{obs\,i}^{<prop_j>}\right]$ will be the result of applying the Time-Exponential Moving Average (TEMA) [50]. See its definition in Equation (5.18).

Once we have the temporal mean of $E_T\left[C_{obs\,i}^{<prop_j>}\right]$ for each observation, we need to normalize the values to obtain a relative measure of the contribution with respect to all the observations.

$$\hat{C}_{obs\,i}^{<prop_j>} = tema\left(E_T\left[C_{obs\,i}^{<prop_j>}\right]\right) \Big/ \sum_{\forall k} tema\left(E_T\left[C_{obs\,k}^{<prop_j>}\right]\right) \tag{5.42}$$

To conclude, the following listings show the implementation of this statistic. Listing 7 extends the first algorithm presented in Section 5.1.2 to include the recurrent update of the temporal mean of $E_T\left[C_{obs\,i}^{<prop_j>}\right]$. This algorithm runs with each new occurrence and is responsible for keeping the statistic updated, while the method in Listing 8 only applies at the moment of computing the resulting normalized contribution $\hat{C}_{obs\,i}^{<prop_j>}$.

algorithm update-evidence-with-new-occurrence-obs_i **is**

const: $\mu_{obs\,i} \leftarrow -\ln(0.01)/persistence_{obs\,i}$ *($persistence_{obs\,i}$ in seconds see Equation (4.27))*
β, parameter to stablish the average duration of the samples for the TEMA

global: $N_{obs\,i}^{<t_{prev}>}$, update of N for the previous occurrence *(initialized to 0)*
t_{prev}, time of the previous occurrence in seconds *(initialized to 0)*
$C_{obs\,i}^{<prop_j>}$, temporal mean of the contribution of observation *i* to property *j (initialized to NULL)*

input: t_{new}, time of the new occurrence in seconds

output: $N_{obs\,i}^{<t_{new}>}$, updated value of N for the observation

precondition: $t_{new} > t_{prev}$

$N_{obs\,i}^{<t_{new}>} \leftarrow 1 + exp\left(-\mu_{obs\,i} \times \left(t_{new} - t_{prev}\right)\right) \times N_{obs\,i}^{<t_{prev}>}$

for each $prop_j$ **do**
  $expected \leftarrow$ calculate-expectation-obs_i($prop_j$, $N_{obs\,i}^{<t_{prev}>}$, $t - t_{prev}$)
  if $C_{obs\,i}^{<prop_j>} = NULL$ **then**
    $C_{obs\,i}^{<prop_j>} \leftarrow expected$
  else
    $C_{obs\,i}^{<prop_j>} \leftarrow exp\left(-\beta\left(t - t_{prev}\right)\right) \times (C_{obs\,i}^{<prop_j>} - expected) + expected$
  end if
end for
$t_{prev} \leftarrow t_{new}$
$N_{obs\,i}^{<t_{prev}>} \leftarrow N_{obs\,i}^{<t_{new}>}$
return $N_{obs\,i}^{<t_{new}>}$

Listing 7. Extended algorithm for updating the accumulated evidence

algorithm calculate-contribution-obs_i-to-prop_j **is**

const: $\mu_{obs\,i} \leftarrow -\ln(0.01)/persistence_{obs\,i}$ *($persistence_{obs\,i}$ in seconds see Equation (4.27))*
β, parameter to stablish the average duration of the samples for the TEMA

global: $C_{obs\,i}^{<prop_j>}$, temporal mean of the contribution of observation *i* to property *j (calculated in Listing 7)*
$N_{obs\,i}^{<t_{prev}>}$, N for the last occurrence *(updated in Listing 7)*
t_{prev}, time of the last occurrence in seconds *(updated in Listing 7)*

input: t, current time in seconds
$prop_j$, property to which the contribution is calculated

output: $\hat{C}_{obs\,i}^{<prop_j>}$, normalized contribution of observation *i* to property *j*

precondition: $t > t_{prev}$

$sum \leftarrow 0$
for each obs_i **do**
  $expected \leftarrow$ calculate-expectation-obs_i($prop_j$, $N_{obs\,i}^{<t_{prev}>}$, $t - t_{prev}$)
  if $C_{obs\,i}^{<prop_j>} = NULL$ **then**
    $C\_updated_{obs\,i}^{<prop_j>} \leftarrow expected$
  else
    $C\_updated_{obs\,i}^{<prop_j>} \leftarrow exp\left(-\beta\left(t - t_{prev}\right)\right) \times (C_{obs\,i}^{<prop_j>} - expected) + expected$
  end if

$sum \leftarrow sum + C\_updated_{obs\ i}^{<prop_j>}$
end for
for each obs_i **do**
$\hat{C}_{obs\ i}^{<prop_j>} \leftarrow C\_updated_{obs\ i}^{<prop_j>} / sum$
end for
return $\hat{C}_{obs\ i}^{<prop_j>}$

Listing 8. Algorithm to compute the contribution of an observation to a property

algorithm calculate-expectation-obs_i **is**
const: a table of probabilities $P(ctx_i|props)$ for observation *i (see Section 4.4.3)*
parameter μ for the observation *i (see Equation (4.27))*
input: $prop$, property for which we want the contribution of the observation
T, time between the last two occurrences in seconds
$N_{obs\ i}^{<t_{occ}>}$, value of N at t_{occ} (previous occurrence of the observation)
output: Expected value $E_T[C_{obs\ i}^{<prop>}]$

$result \leftarrow 0$
$n_1 \leftarrow N_{obs\ i}^{<t_{occ}>}$
$n_0 \leftarrow max(\exp(\mu \times T) \times N_{obs\ i}^{<t_{occ}>}, 0.01)$
$PROPS_{pos} \leftarrow (empty\ list)$
$PROPS \leftarrow (empty\ list)$
for each $props_j$ defined in $P(CTX_i|props)$ **do**
if $props_j \mid prop = PROP$ **then**
$PROPS_{pos}.insert(props_j)$
end if
$PROPS.insert(props_j)$
end for
for each $props_k$ **in** $PROPS_{pos}$ **do**
for each $props_l$ **in** $PROPS$ **do**
if $props_k \neq props_l$ **do**
$result \leftarrow result$ + abs (calculate-partial-expectation-obs_i(n_0, n_1, T, $P(CTX_i|props_l)$)
- calculate-partial-expectation-obs_i(n_0, n_1, T, $P(CTX_i|props_k)$))
end if
end for
end for
return $result$

Listing 9. Algorithm to calculate the expected contribution of an observation to a property

algorithm calculate-partial-expectation-obs_i **is**
const: parameter α for observation *i (see Section 4.4.4)*
parameter μ for observation *i (see Equation (4.27))*
input: n_0, lower limit of $N_{obs\,i}^{<t>}$
n_1, upper limit of $N_{obs\,i}^{<t>}$
T, time between occurrences in seconds
pr, conditional probability $P(CTX_i \mid PROPS_j)$
output: result for the integral in (5.32)

$result \leftarrow ln(pr)$
$n_b \leftarrow ln((1-pr)/pr)/\alpha$
if $n_b \leq n_0$ **then**
$result \leftarrow result + ln(2) \times \left(E_1(0.85 \times \alpha \times n_0) - E_1(0.85 \times \alpha \times n_1)\right) \times ((1-pr)/pr)^{0.85}/(\mu \times T)$
else if $n_b \geq n_1$ **then**
$result \leftarrow result + (ln(2) \times \left(E_i(0.85 \times \alpha \times n_1) - E_i(0.85 \times \alpha \times n_0)\right) \times (pr/(1-pr))^{0.85}$
$-\alpha \times (n_1 - n_0))/(\mu \times T) + ln((1-pr)/pr)$
else
$result \leftarrow result + \Big(ln(2) \times \Big(\left(E_1(0.85 \times \alpha \times n_b) - E_1(0.85 \times \alpha \times n_1)\right) \times ((1-pr)/pr)^{0.85}$
$+\left(E_i(0.85 \times \alpha \times n_b) - E_i(0.85 \times \alpha \times n_0)\right) \times (pr/(1-pr))^{0.85}\big) - \alpha \times (n_b - n_0)$
$+ln((1-pr)/pr) \times \left(ln(n_b) - ln(n_0)\right)\Big)/(\mu \times T)$
end if
return *result*

Listing 10. Algorithm to calculate the partial expected contribution of an observation

5.2.4 Example simulation

In the following, we show an example to illustrate the statistics we have described in the previous sections. The simulation has been executed with a Java implementation of the algorithms that we have presented. It covers a period of 7 days with estimates every 2 minutes, resulting in a total of 5040 estimates processed in ~1500ms on an average computer.

Following the hospital robot example, the degree of engagement was determined by observing how patients behave with the robot, so it is a sign of acceptance that people call the robot by its name, or, conversely, it is negative that people refuse to interact with the robot when it approaches them. The details of these two observations can be seen in Table 16 according to the specification that was introduced in Section 4.1. In this context, we will consider a scenario in which, at first, people are reluctant to interact with the robot, but as time goes by, they become familiar with it.

Table 16. Observations considered in the simulation

(PATTERN) People call the robot by its name	(PERSISTENCE) Some hours	(REPETITION) 5
(PROPERTY) Engagement	(DIRECTION) Reinforce	(STRENGTH) Low
(PATTERN) People refuse to interact	(PERSISTENCE) Medium (7-day timeframe)	(REPETITION) 5
(PROPERTY) Engagement	(DIRECTION) Undermine	(STRENGTH) Low

We have run the simulation with 200 random occurrences for each observations in Table 16 distributed over 7 days, in such a way that the observation "People refuse to interact" ("refused" for short) is more frequent at the beginning and decreases gradually during the simulation, while the observation "People call the robot by its name" ("named" for short) becomes more frequent as the simulation progresses. Figure 23 shows 3 days of the simulation.

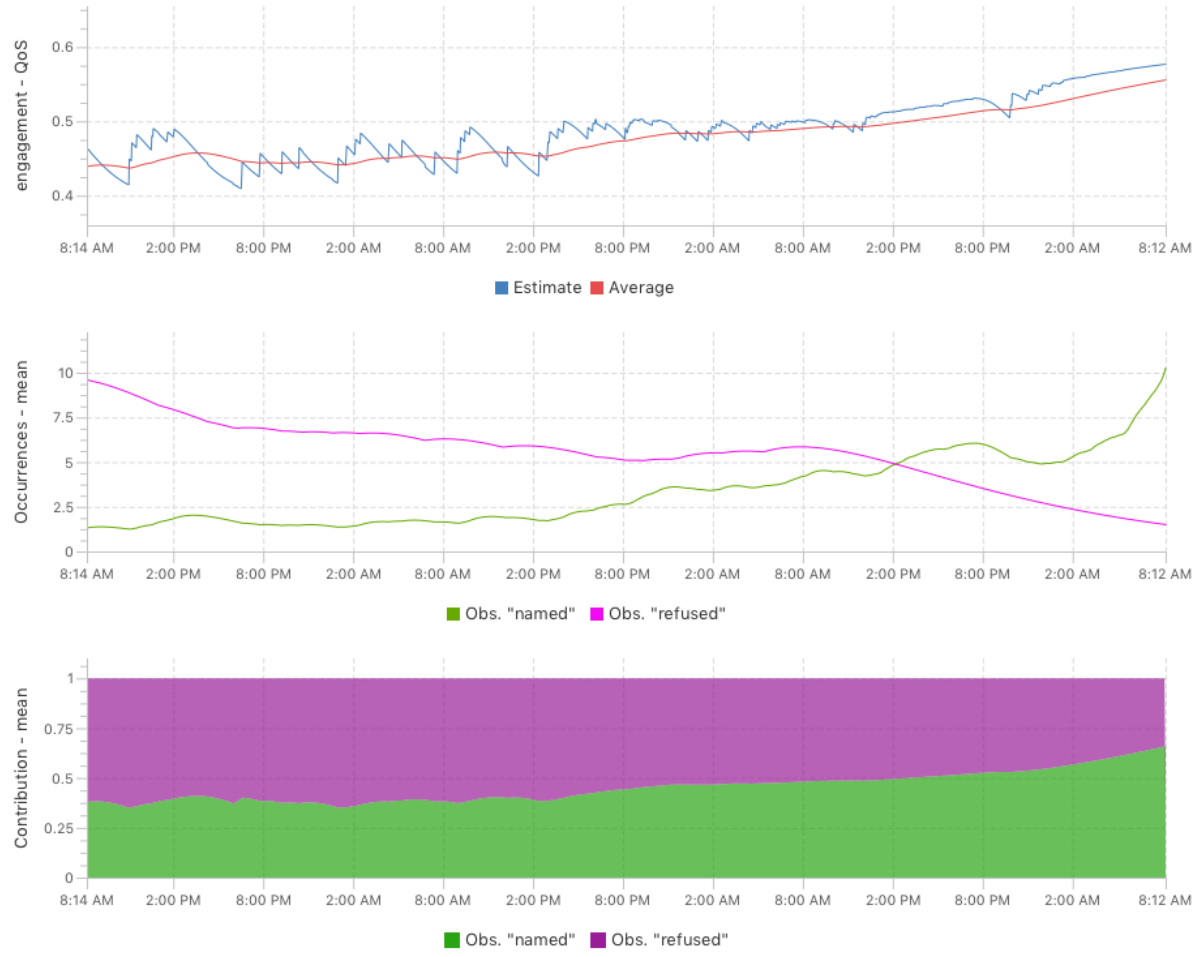


Figure 23. Simulation results. (Top) QoS estimates for engagement and its time-exponential moving average. (Middle) Temporal mean of the average number of active occurrences for each observation. (Bottom) Relative contribution of the observations to the QoS estimates

The upper graph presents the instantaneous and the average value of the degree of engagement that the robot has shown during the simulation. As expected, the performance improved over time as the observation “named” becomes more frequent and “refused” less. This trend stands out more clearly in the time-exponential moving average as it smooths out the fluctuations. Note that moving averages in the simulation have been set up with 6-hour time windows. Regarding the graph in the middle, it shows the temporal mean of the average number of active occurrences for each observation, which quantifies the average evidence that supports (or not) the idea that the robot works optimally in terms of engagement. Finally, the lower graph corresponds to the relative contribution of the observations to the QoS estimates. According to this graph, at the beginning, the QoS estimates have the following support: the observation “refused” would account for 62% of the evidence, while “named” would account for 38%. However, the contribution of the latter grows during the simulation and ends up being 66% of the evidence.

5.3 Tuning QoS metrics

Chapter 4 presented the bases to create a modelling language, from which to automatically generate a probabilistic network to estimate non-functional properties. Through this language, domain experts can easily express high-level specifications without having to deal with the complexities of the underlying probabilistic model. However, the trade-off is the consequent loss of flexibility, since users must stick to what is provided. For instance, according to Section 4.1, domain experts can choose from 5 levels (from very low to very high) to define the strength of the evidence produced by an observation. Anything else would require domain experts to have the ability to adjust the probabilistic model, which is out of the question in most cases.

In this section, we will introduce an alternative approach where probabilistic networks give way to neural networks. The weights of the neural network will be initialized by the parameters of the probabilistic network (as a way of transferring the knowledge), so in the end both approaches should produce the same estimations. Although they may be equivalent, neural networks have the advantage of backpropagation [53], which is a widely-used training

algorithm that allows adjusting the weights through examples. This could provide domain experts with a simple method to fine-tune QoS estimates, where they would only need to prepare a limited number of examples showing how some input observations should produce certain output estimates, based on experience, analysis, testing, whatever they think is best.

5.3.1 Proposed neural network architecture

Figure 24 shows the proposed neural network architecture that results from the probabilistic model presented in Section 3.4. More specifically, this architecture arises from the translation as a computation graph of the formulas (5.13), (5.16) and (5.17). Also, see that it is a feedforward network since the information moves in one direction from the inputs to the outputs. There are no cycles or loops in the network.

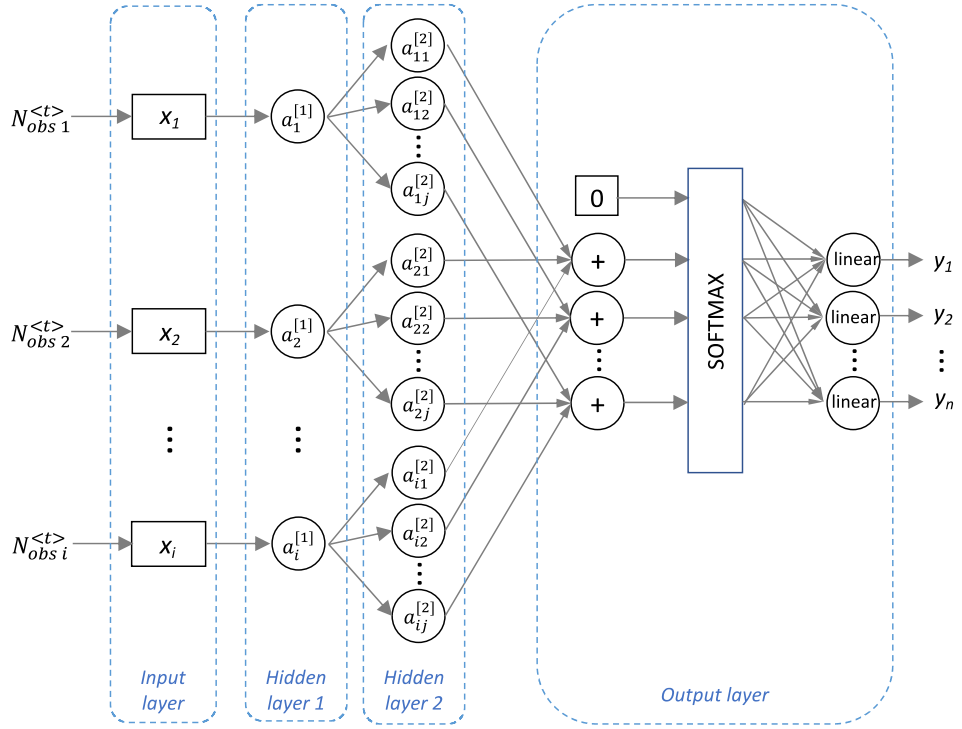


Figure 24. Proposed neural network architecture

The neural network consists of the following layers.

- **Input layer:** It considers the average number of active occurrences for each observation (denoted by $N_{obs\,i}^{<t>}$), so the input will be a vector of real numbers greater than or equal

to 0 with size equal to the number of observations. The point of considering $N_{obs\,i}^{<t>}$ instead of the raw occurrence of the observations allows us to simplify the network at the cost of losing the ability to adjust the persistence of the observations (which is embedded in the calculation of $N_{obs\,i}^{<t>}$).

- **Hidden layer 1:** This layer comprises a number of units equal to the number of observations, where a unit is defined by Equation (5.43), being w_i and b_i trainable parameters. Since this expression corresponds to $f(z)$ in (5.16) with $k = 0$, the unit is initialized according to (5.44). Inverting the expressions, we can retrieve the updated probabilities after a training through (5.45). Note that these probabilities can be useful when performing analyses or considering other applications. Finally, this layer seems to be involved in how the influence of an observation varies along $N_{obs\,i}^{<t>}$ (what we defined as repetition of the observations in Chapter 4).

$$a_i^{[1]} = sigmoid(w_i x_i + b_i) \tag{5.43}$$

$$w_i = \alpha_{obs\,i}\,; \quad b_i = -\ln\left(\frac{P(\neg CTX_i \mid PROPS_0)}{P(CTX_i \mid PROPS_0)}\right) \tag{5.44}$$

$$P(CTX_i \mid PROPS_0) = \frac{1}{1+e^{-b_i}}; \quad P(\neg CTX_i \mid PROPS_0) = 1 - P(CTX_i \mid PROPS_0) \tag{5.45}$$

- **Hidden layer 2:** This layer consists of a set of activations $a_{ij}^{[2]}$ where the first index identifies an observation and the second a state combination of properties, i.e., a value of $props \in \{PROPS_0, PROPS_1, \ldots, PROPS_{2^M-1}\}$, being M the number of properties (see Section 3.4.4). Equation (5.46) shows the definition of a unit in this layer, being w_{ij} and v_{ij} trainable parameters. Note that it uses logarithm as the activation function. Although this is not a common feature, there have been similar proposals in the past [54]. In addition, we use exponentials with the weights, which is convenient to make them be real numbers, initialized as shown in (5.47). Otherwise, the training of w_{ij} and v_{ij} would have to be limited to positive numbers, as these weights represent a ratio of probabilities according to (5.16). Finally, we can retrieve the updated probabilities after a training through (5.48).

$$a_{ij}^{[2]} = \ln\left(e^{w_{ij}}\, a_i^{[1]} + e^{v_{ij}}\left(1 - a_i^{[1]}\right)\right) \tag{5.46}$$

$$w_{ij} = ln\left(\frac{P(CTX_i \mid PROPS_j)}{P(CTX_i \mid PROPS_0)}\right); \quad v_{ij} = ln\left(\frac{P(\neg CTX_i \mid PROPS_j)}{P(\neg CTX_i \mid PROPS_0)}\right) \tag{5.47}$$

$$P(CTX_i \mid PROPS_j) = e^{w_{ij}}\, P(CTX_i \mid PROPS_0); \quad P(\neg CTX_i \mid PROPS_j) = e^{v_{ij}}\, P(\neg CTX_i \mid PROPS_0) \tag{5.48}$$

Considering the initialization of the hidden layer 1 and 2, we can observe that they rely on some common probabilities, so there is a dependency between the parameters of these layers. Since the training process would normally assume independent parameters, we may not be able to extract a consistent set of probabilities from the trained weights. That is, the resulting weights might produce invalid probability values or the relationship $P(\neg CTX_i \mid PROPS_j) = 1 - P(CTX_i \mid PROPS_j)$ (for $j = 0,1,\ldots$) might not hold. The dependency between layers seems necessary to make $a_{ij}^{[2]}$ equal to 0 when $N_{obs\,i}^{<t>}$ is 0. In other words, a unit in layer 2 establishes the influence of an observation i in relation to $PROPS_0$ and $PROPS_j$, so when there is no observation ($N_{obs\,i}^{<t>} = 0$), the influence should be 0 ($a_{ij}^{[2]} = 0$). Following this principle, we have added a regularization term in the loss function to favour weight values that keep probabilities consistent. This regularization term is defined in (5.49) as the L1 norm of the activations of the second layer when the input of the neural network is 0. The importance of the regularization term in the loss function is controlled by the hyper-parameter λ.

$$\lambda \sum_{\substack{i,j \\ j = 1,\ldots,2^M - 1 \quad \forall i \mid x_i = 0}} \left| a_{ij}^{[2]} \right| \tag{5.49}$$

- **Output layer:** This layer is responsible for transforming the activations of the previous layer into QoS estimates. The layer applies three consecutive operations that do not have trainable parameters. First, the layer adds the contribution of each observation in relation to $PROPS_0$ and $PROPS_j$, which corresponds to the term $\sum_i \ln\,(LR_{obs\,i}^{<l,k>})$ in (5.13) for $l = j$ and $k = 0$. Equation (5.50) defines this first step. Second, the layer performs a softmax [53], which ends in a vector of probabilities $P(PROPS_k \mid N_{obs\,1}^{<t>}, \ldots, N_{obs\,N}^{<t>})$ for $k = 0, \ldots, 2^M - 1$, being M the number of properties. These probabilities correspond to the term $\left(1 + \sum_{l \neq k} e^{\sum_i \ln\,(LR_{obs\,i}^{<l,k>})}\right)^{-1}$ in (5.13). Equation (5.51) defines this second step.

$$z_j = \begin{cases} 0 & j = 0 \\ \sum_{\forall i} a_{ij}^{[2]} & j = 1, \dots, 2^M - 1 \end{cases} \tag{5.50}$$

$$\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{j=1}^{2^M-1} e^{z_j}}, \qquad j = 0, 1, \dots, 2^M - 1 \tag{5.51}$$

Finally, the layer applies a linear function to obtain a QoS estimate for each property. It multiplies the vector resulting from the softmax function by a $2^M \times M$ matrix with the binary representation of the numbers from 0 to $2^M - 1$. An example for two properties is shown in (5.52), where y_1 and y_2 are the resulting QoS estimates associated with property 1 and 2, respectively.

$$(y_2 \quad y_1) = (\sigma_0 \quad \sigma_1 \quad \sigma_2 \quad \sigma_3)\begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} = (\sigma_2 + \sigma_3 \quad \sigma_1 + \sigma_3) \tag{5.52}$$

5.3.2 Practical cases

In the following, we are going to introduce some examples to illustrate the use of the proposed neural network. All the cases below are based on the hospital robot example presented in Section 3.3 and have been implemented in Python with TensorFlow [55].

5.3.2.1 Adjusting the strength of an observation

The observation "People interact through the robot's touch screen" ("screen interaction" for short) is considered a good sign in terms of user engagement. The details of this observation can be seen in Table 17 according to the specification that was introduced in Section 4.1.

Table 17. Observation "screen interaction"

(PATTERN) A screen event is received	(PERSISTENCE) Short	(REPETITION) None
(PROPERTY) Engagement	(DIRECTION) Reinforce	(STRENGTH) High

Suppose that after evaluating the probabilistic model generated from the specification, we decide to adjust the strength of this observation. We want the QoS estimate to reach 0.9 instead of 0.78 when the observation occurs. For this, the training data set that we will need to prepare

will have only two samples: (1) an input-output pair $(x^{(1)}, y^{(1)})$ to indicate the starting point when there are no occurrences and (2) an input-output pair $(x^{(2)}, y^{(2)})$ to indicate the desired endpoint when there is at least one occurrence of the observation. See in (5.53) the data set used in this example. Note that $x^{(n)}$ is a column vector containing the value of $N_{obs\,i}^{<t>}$ for each observation, where the last element of the vector corresponds to the observation "screen interaction". On the other hand, $y^{(n)}$ is also a column vector with the QoS estimates for each property, where the first element is the estimate for "safety" and the last element the one for "engagement".

$$X = [x^{(1)}, x^{(2)}] = \left[\begin{pmatrix}0\\ \vdots \\ 0\end{pmatrix}, \begin{pmatrix}0\\ \vdots \\ 1\end{pmatrix}\right] \quad Y = [y^{(1)}, y^{(2)}] = \left[\begin{pmatrix}0.5\\ 0.5\end{pmatrix}, \begin{pmatrix}0.5\\ 0.9\end{pmatrix}\right] \tag{5.53}$$

The training process was configured with Adam [56] as the optimization algorithm, a learning rate of 0.001 and the mean squared error (MSE) as the loss function with $\lambda = 0.001$ for the regularization term. As for the testing, since the adjustment we want to perform is equivalent to changing the strength from "high" to "very high", we have generated a data set of 100,000 random samples using the corresponding probabilistic model. It is worth noting that the adjustments are not required to match the terms of a high-level specification (e.g., from "high" to "very high"), but domain experts can propose any input-output sample based on experience, tests, etc. The sole purpose of mapping adjustments to changes in the high-level specification is to facilitate the validation of the example.

We ran the training for 2,000 epochs, ending with a mean absolute percentage error of 0.15%, while the testing shows an error of 1%. Figure 25 illustrates the effect of the observation "screen interaction" on the QoS estimate before and after the training. We can see how the training was able to change this effect from 0.78 to 0.9079. Regarding the consistency of the underlying probabilities, after obtaining them with the expressions in (5.45) and (5.48), we can calculate $P(CTX_i \mid PROPS_j) + P(\neg CTX_i \mid PROPS_j)$. The result of this sum should always be 1 for any j, thus we can measure consistency as the deviation from this value through the mean absolute error, so that the smaller the error, the greater the consistency. The example shows a mean error of $1.27 \cdot 10^{-4}$ with a maximum of $3.87 \cdot 10^{-4}$.

Although all these results are not bad, we have the option to improve them by segmenting the neural network into two smaller networks, one for each property. This is possible due to the

absence of common observations between the properties. With the segmented network, the error on the test data set decreases from 1% to 0.39% with a consistency error of $1.94 \cdot 10^{-5}$ (maximum of $8.46 \cdot 10^{-5}$).

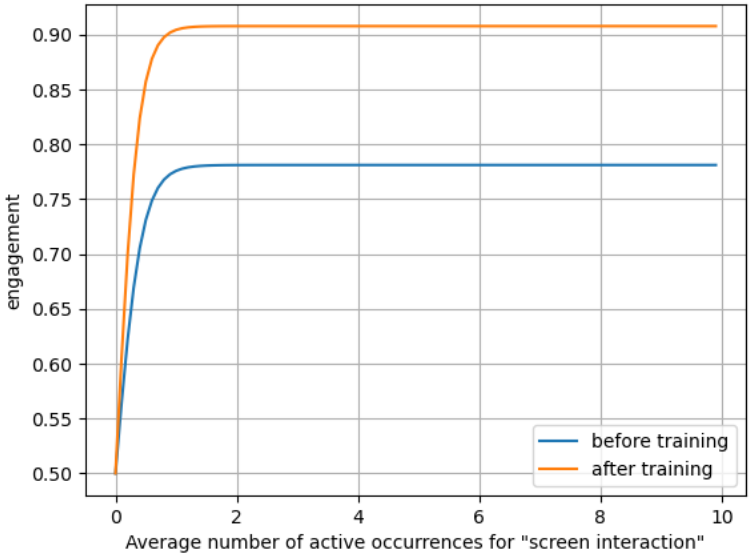


Figure 25. QoS estimate before and after training

5.3.2.2 Adjusting the strength and repetition of an observation

Now, let us adjust the strength and the repetition of the observation "screen interaction" shown in Table 17. As in the previous example, we want to increase the QoS estimate up to 0.9, but this time gradually. That is, the observation will not be consolidated until it is repeated at least 5 times. The training data set will have only three samples: (1) a sample $\left(x^{(1)}, y^{(1)}\right)$ to indicate the starting point when there are no occurrences, (2) a sample $\left(x^{(2)}, y^{(2)}\right)$ to consider the midpoint and (3) another sample $\left(x^{(3)}, y^{(3)}\right)$ to indicate the desired endpoint when $N_{obs\,i}^{<t>} = 6$. Note that when an observation is repeated 5 times, it means 6 occurrences: the first occurrence plus 5 repetitions. See in (5.54) the data set used in this example, where $x^{(n)}$ and $y^{(n)}$ are column vectors and follow the same notation that we defined in the previous section.

$$X = \left[x^{(1)}, x^{(2)}, x^{(3)}\right] = \left[\begin{pmatrix}0\\ \vdots \\ 0\end{pmatrix}, \begin{pmatrix}0\\ \vdots \\ 3\end{pmatrix}, \begin{pmatrix}0\\ \vdots \\ 6\end{pmatrix}\right]; \quad Y = \left[y^{(1)}, y^{(2)}, y^{(3)}\right] = \left[\begin{pmatrix}0.5\\ 0.5\end{pmatrix}, \begin{pmatrix}0.5\\ 0.83\end{pmatrix}, \begin{pmatrix}0.5\\ 0.9\end{pmatrix}\right] \tag{5.54}$$

For the training process, we used the Adam optimization algorithm with a learning rate of 0.01 and the MSE as the loss function with $\lambda = 0.001$ for the regularization term. Since the adjustment we want to perform is equivalent to changing in Table 17 the strength from "high" to "very high" and the repetition from 0 to 5, we have generated a test data set with 100,000 random samples using the corresponding probabilistic model. We ran the training for 2,000 epochs, ending with a mean absolute percentage error of 0.08%, while the testing shows an error of 1.27%. See in Figure 26 the effect of the observation before and after the training. Regarding the consistency of the probabilities, the mean absolute error is $1.02 \cdot 10^{-3}$ with a maximum error of $4.61 \cdot 10^{-3}$. Finally, considering a segmented neural network focused on "engagement", we can decrease the error on the test data set to 0.11% with a consistency error of $3.07 \cdot 10^{-4}$ (maximum of $9.32 \cdot 10^{-4}$).

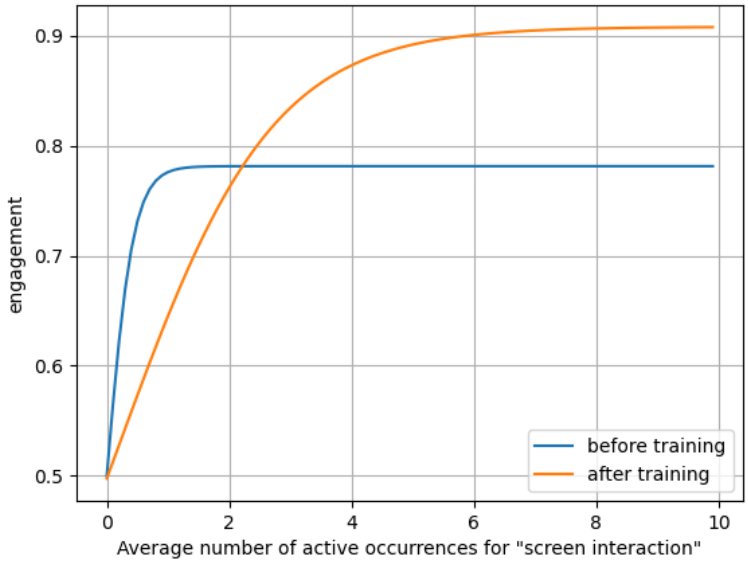


Figure 26. QoS estimate before and after training

5.3.2.3 Adjusting several observations

In this example we are going to consider the observation "People ask questions" ("question" for short) together with the observation "screen interaction". The details of this new observation can be seen in Table 18 according to the specification that was introduced in Section 4.1.

Table 18. Observation "question"

(PATTERN) A question is recognized	(PERSISTENCE) Short	(REPETITION) None
(PROPERTY) Engagement	(DIRECTION) Reinforce	(STRENGTH) High

In addition to the adjustment applied in the previous example, we will decrease the strength of "question" from 0.78 to 0.68. Moreover, this observation will need to be repeated at least once to reach its maximum influence. The training data set will have five samples: (1) a sample $\left(x^{(1)}, y^{(1)}\right)$ to indicate the starting point when there are no occurrences, (2) two samples, $\left(x^{(2)}, y^{(2)}\right)$ and $\left(x^{(3)}, y^{(3)}\right)$, to adjust the midpoint and endpoint of the observation "screen interaction" and (3) two samples, $\left(x^{(4)}, y^{(4)}\right)$ and $\left(x^{(5)}, y^{(5)}\right)$, to indicate the midpoint and endpoint of the observation "question". See in (5.55) the training data set used in this example. The vector $x^{(n)}$ contains the value of $N_{obs\,i}^{<t>}$ for each observation, where the first component of the vector corresponds to "question" and the last one to "screen interaction".

$$X = \left[x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}\right] = \left[\begin{pmatrix}0\\ \vdots\\ 0\end{pmatrix}, \begin{pmatrix}0\\ \vdots\\ 3\end{pmatrix}, \begin{pmatrix}0\\ \vdots\\ 6\end{pmatrix}, \begin{pmatrix}1\\ 0\\ \vdots\\ 0\end{pmatrix}, \begin{pmatrix}2\\ 0\\ \vdots\\ 0\end{pmatrix}\right]$$
$$Y = \left[y^{(1)}, y^{(2)}, y^{(3)}, y^{(4)}, y^{(5)}\right] = \left[\begin{pmatrix}0.5\\ 0.5\end{pmatrix}, \begin{pmatrix}0.5\\ 0.83\end{pmatrix}, \begin{pmatrix}0.5\\ 0.9\end{pmatrix}, \begin{pmatrix}0.5\\ 0.65\end{pmatrix}, \begin{pmatrix}0.5\\ 0.68\end{pmatrix}\right] \tag{5.55}$$

The training process was executed with the Adam optimization algorithm considering a learning rate of 0.01 and the MSE as the loss function with $\lambda = 0.001$ for the regularization term. The adjustment we want to perform is equivalent to (1) changing the strength of "question" from "high" to "medium" and its repetition from 0 to 1, and (2) changing the strength of "screen interaction" from "high" to "very high" and its repetition from 0 to 5. Considering these changes in the probabilistic model, we have generated a test data set with 100,000 random samples. The training was carried out for 2,000 epochs and ends with a mean absolute percentage error of 0.15%, showing an error of 1.52% on the test data. As for the consistency of the probabilities, the mean absolute error is $1.02 \cdot 10^{-3}$ with a maximum error of $4.61 \cdot 10^{-3}$. As we have already seen in the previous examples, we can improve these results with a segmented neural network, in that case, the error on the test data set decreases to 0.11% with a consistency error of $7.36 \cdot 10^{-4}$ (maximum of $3.58 \cdot 10^{-3}$). Figure 27 shows the effect of the observation "question" before and after the training.

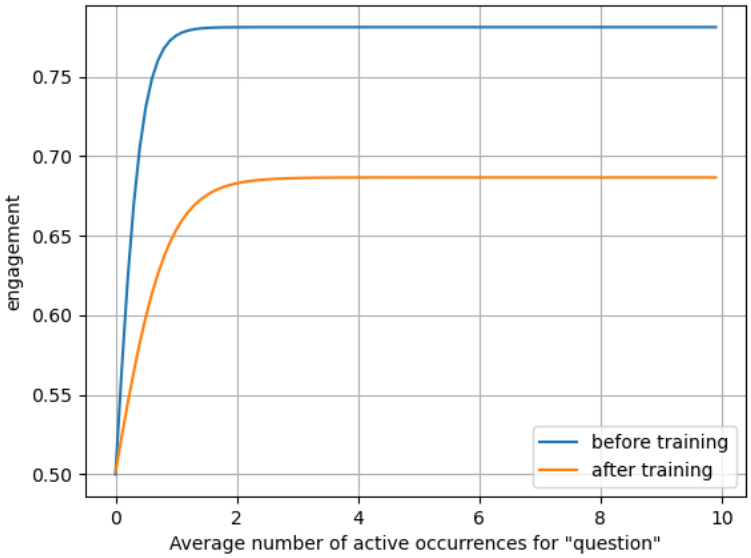


Figure 27. QoS estimate before and after training

Until now, we have considered a single training set in the examples. Next, we will try a different approach, in which we will perform several consecutive training sessions, each one focused on the adjustment of an observation. In this sense, we can see in (5.56) the division of the training data into two sets.

$$X_{screen} = [x^{(1)}, x^{(2)}, x^{(3)}] = \left[\begin{pmatrix}0\\\vdots\\0\end{pmatrix}, \begin{pmatrix}0\\\vdots\\3\end{pmatrix}, \begin{pmatrix}0\\\vdots\\6\end{pmatrix}\right]; \quad Y_{screen} = [y^{(1)}, y^{(2)}, y^{(3)}] = \left[\begin{pmatrix}0.5\\0.5\end{pmatrix}, \begin{pmatrix}0.5\\0.83\end{pmatrix}, \begin{pmatrix}0.5\\0.9\end{pmatrix}\right]$$
$$X_{question} = [x^{(1)}, x^{(4)}, x^{(5)}] = \left[\begin{pmatrix}0\\\vdots\\0\end{pmatrix}, \begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}, \begin{pmatrix}2\\0\\\vdots\\0\end{pmatrix}\right]; \quad Y_{question} = [y^{(1)}, y^{(4)}, y^{(5)}] = \left[\begin{pmatrix}0.5\\0.5\end{pmatrix}, \begin{pmatrix}0.5\\0.65\end{pmatrix}, \begin{pmatrix}0.5\\0.68\end{pmatrix}\right] \tag{5.56}$$

We have repeated the training process with the same configuration for the optimizer, but arranged in two consecutive steps: 2,000 epochs for the data set (X_{screen}, Y_{screen}) and 2,000 epochs for $(X_{question}, Y_{question})$. Considering the complete neural network (without segmentation), the error on the test data decreases to 1.04% with a consistency error of $9.83 \cdot 10^{-4}$ (maximum of $4.45 \cdot 10^{-3}$). The training of one observation appears to be independent of all other observations, at least to some extent, which provides users with a lot of flexibility to tune the network. We will see more details about this approach in Section 5.3.3.3.

5.3.3 Comparison with a regular feedforward neural network

This section presents a comparison between the proposed architecture and a conventional feedforward neural network, which will allow us to point out the benefits of the proposal.

5.3.3.1 Model complexity

Looking at the probabilistic model introduced in Section 3.4 as a non-linear function that maps observations (in terms of $N_{obs\,i}^{<t>}$) to QoS estimates, the complexity shows up in the variability of the outputs. Put simply, if the entire range of inputs were mapped to just two output values, it would be easy to remember, and the complexity would be low. On the other hand, the complexity would be high, if the outputs change so much that it is difficult to establish a pattern. This idea of complexity is related to the Shannon entropy [57], so we will consider the entropy of the QoS estimates as a measure of the complexity of the model. The higher the entropy, the higher the complexity.

Basically, the complexity of the probabilistic model increases as the corresponding high-level specification (1) has more observations, (2) which are shared among more properties, and (3) whose "repetition" are set to higher values. Based on these three points, we have created several specifications of different complexity, from which we generate data by sampling the probabilistic model with random inputs. Once we have data, it is possible to train a conventional neural network to approximate the QoS estimates. Table 19 shows the results, which establish a relationship between the complexity of the specification and the size of the neural network and the training data. The columns in the table are the followings:

- Entropy: It is calculated based on the histogram of the QoS estimates resulting from the random sampling of the probabilistic model. Equation (5.57) shows the definition where $p_i(Y)$ is the probability mass associated with the i-th bin.

$$\mathrm{H(Y)} = -\sum_{\forall i} p_i(Y)\ \ln\big(p_i(Y)\big) \tag{5.57}$$

Table 19. QoS specifications with a regular neural network

Entropy	QoS specification	Neural network	Trainable parameters	Training data set	MAPE
0.068	1 property + 1 observation (repetition = 0)	Network with a single sigmoid unit	2 (4)	100 samples	0.18%
0.89	1 property + 10 observations (repetition = 0)	Network with a single sigmoid unit	20 (40)	1,000 samples	1.2%
3.64	1 property + 2 observations (repetition ~ $\mathcal{N}(\mu = 50, \sigma^2 = 1)$)	Fully connected network with 2 layers: 10 ReLU units + 1 sigmoid unit	60 (8)	5,000 samples	1%
4.96	2 properties + 2 shared observations (repetition ~ $\mathcal{N}(\mu = 50, \sigma^2 = 1)$)	Fully connected network with 2 layers: 10 ReLU units + 2 sigmoid units	80 (16)	5,000 samples	0.51%
7.06	2 properties + 10 shared observations (repetition ~ $\mathcal{N}(\mu = 50, \sigma^2 = 25)$)	Fully connected network with 2 layers: 30 ReLU units + 2 sigmoid units	720 (80)	10,000 samples	1%

- QoS specification: Description of the high-level specification from which the probabilistic model is automatically generated.
- Neural network: Architecture used to approximate the QoS estimates. The network architecture has been selected empirically based on experiments.
- Trainable parameters: It shows the number of trainable parameters of the neural network compared to what would be needed with the architecture we proposed in Figure 24 (indicated in parentheses).
- Training data set: Size of the data set that was used to train the neural network.
- MAPE: Mean absolute percentage error of the results produced by the neural network with respect to those of the probabilistic model. The test was performed with a data set of 100,000 samples.

Although the complexity of the QoS specification could be increased much more than what is shown in the table, research states that a feedforward neural network with a single hidden layer is sufficient to represent any function [58]. In this sense, a fully connected network with two layers should have no problem approximating any specification if the hidden layer and the training data are large enough. However, it does not seem feasible for domain experts to create a data set of thousands of samples from scratch. In contrast to this traditional data-centric approach, our proposal promotes the initialization of the neural network based on the

probabilities derived from the QoS specification, so that knowledge is transferred more efficiently from the experts to the network.

In the table, decisions about the size of the neural network and the training data were made empirically on the basis of trial and error. Note that to keep the error small they must grow as the complexity of the specification increases. On the contrary, in our proposal, the neural network architecture naturally grows in complexity as new inputs are added. Moreover, this architecture is not arbitrary, but it is based on the probabilistic model we have developed throughout this Thesis.

As for the number of trainable parameters, they grow linearly with the number of observations. In a fully connected neural network, they increase twice the number of hidden units for each new observation, while, in our proposal, the number of parameters increases 2^{M+1} per observation, being *M* the number of properties. In general, the number of connections will tend to be higher in a fully connected network, which means more parameters and a longer training. However, a larger number of parameters can extend the relationships that the network can learn. For example, we have considered independent observations in our model, but a fully connected neural network can learn that the effect of two observations is different depending on their joint values. In this sense, the proposed architecture is more rigid in that it has been built based on some assumptions that limit it.

5.3.3.2 Fitting QoS estimates with small data sets

As we saw in the examples, the proposed neural network allows users to adjust the QoS estimates using very few samples. In this section, we try the same with a regular neural network. In particular, we have repeated the example in Section 5.3.2.2, in which we adjusted the strength and repetition of the observation “screen interaction”. For that, we have considered a fully connected network with two layers: a hidden layer with 20 ReLU units and an output layer with 2 sigmoid units.

The first step was to train the neural network to approximate the initial QoS specification. For that, we have used the optimization algorithm "Follow the Regularized Leader" (FTRL) [59] with a learning rate of 0.01 and the MSE as the loss function. The training data set consisted

of 10,000 samples randomly generated with the probabilistic model that is derived from the QoS specification. We ran the training for 20,000 epochs, ending with a mean absolute percentage error of 0.047%, tested on a data set of 100,000 samples.

Once we had the neural network ready to produce QoS estimates, we proceeded with the adjustment of the observation "screen interaction" using the training data in (5.54), which corresponds to changing the observation strength from "high" to "very high" and its repetition from 0 to 5. The training was run for 8,000 epochs using FTRL, a learning rate of 0.01 and the MSE loss function. Finally, the testing shows an error of 16% over a data set of 100,000 samples. We have also tried to extend the training data up to 100 samples, increasing the error to 18.34%.

In the proposed architecture, the data from each observation flows separately until everything is combined in the output layer. It seems that this scheme favours the adjustment of an observation almost without affecting the rest. On the contrary, in a fully connected neural network, everything is interrelated, which means that a complete training set (with thousands of samples) is required to adjust the observation in question and ensure that the rest behaves as it should.

5.3.3.3 Breaking down the training process into several independent sessions

As already mentioned before, the estimation of QoS can be seen as a function $Y = f(X)$, where the vector X contains the observations (in terms of $N_{obs\,i}^{<t>}$) and the vector Y the QoS estimates corresponding to each non-functional property. If the function $f(X)$ could be decomposed into several independent functions defined on mutually exclusive subsets of inputs, so that $f(X) = g(f_1(X_1), f_2(X_2) \ldots)$, being g a known aggregation function, then $f(X)$ could be approximated by training each function f_i separately.

The proposed neural network architecture is suitable for this type of training decomposition since the input data is developed independently until it is aggregated in the output layer. This is the same reason that allows the neural network to be trained with a few samples (see the previous section). To illustrate this feature, let us consider the hospital robot example, in which instead of initializing the parameters of the neural network using the probabilities derived from

the QoS specification, we are going to train the proposed architecture with data generated from the probabilistic model. However, unlike the traditional approach, we will perform 11 consecutive training sessions on different data sets, each one focused on an observation. For instance, the training data set linked to the i-th observation would follow the configuration shown in (5.58). The column vectors in X_i have 0s in all the components, except the one associated with the i-th observation, which varies from 0 to 9.99. As for Y_i, it shows the QoS estimates resulting from evaluating the probabilistic model with X_i. Each data set will have 100 samples.

$$X_i = \left[x_i^{(1)}, x_i^{(2)}, x_i^{(3)}, \dots, x_i^{(100)}\right] = \left[\begin{pmatrix}0\\ \vdots\\ 0\\ 0\\ \vdots\end{pmatrix}, \begin{pmatrix}0\\ \vdots\\ 0.1\\ 0\\ \vdots\end{pmatrix}, \begin{pmatrix}0\\ \vdots\\ 0.2\\ 0\\ \vdots\end{pmatrix}, \dots, \begin{pmatrix}0\\ \vdots\\ 9.99\\ 0\\ \vdots\end{pmatrix}\right] \tag{5.58}$$

$$Y_i = \left[y^{(1)}, y^{(2)}, y^{(3)}, \dots, y^{(100)}\right] = \left[\begin{pmatrix}0.5\\ 0.5\end{pmatrix}, \begin{pmatrix}0.5\\ 0.63\end{pmatrix}, \begin{pmatrix}0.5\\ 0.68\end{pmatrix}, \dots, \begin{pmatrix}0.5\\ 0.9\end{pmatrix}\right]$$

We have used the Adam optimization algorithm considering a learning rate of 0.001, the MSE as the loss function and $\lambda = 0.001$ for the regularization term. We ran each training session for 3,000 epochs, and after that, the neural network was tested with a set of 100,000 samples, showing a mean absolute percentage error of 1.76%.

The same was repeated with a fully connected network with two layers: a hidden layer with 20 ReLU units and an output layer with 2 sigmoid units. The training was set with FTRL as the optimization algorithm, a learning rate of 0.01 and MSE as the loss function. We ran each training session for 5,000 epochs, ending with a mean absolute percentage error of 31.11%, tested on a data set of 100,000 samples. As expected, the architecture of a fully connected neural network is not appropriate for this type of training since the observations are not treated in isolation but rather the opposite, all the inputs are combined in each unit of the hidden layer.

Finally, the ability to break down the training process gives users a lot of flexibility in how they can tune the neural network. Thus, they can do it in several steps, focusing only on a limited set of observations and using few samples.

5.4 Summary

In this chapter, we have presented the main aspects to support the execution of QoS metrics, including the algorithms for calculating QoS estimates, the application of statistics, and how to tune QoS metrics. Next chapter will address the creation of a Domain Specific Modelling Language (DSML) to specify QoS metrics. This modelling language will be based on the concepts discussed so far and will provide users with a tool to write and run high-level descriptions of QoS metrics.

Chapter 6 Modelling language for QoS metrics

In this chapter we propose a Domain Specific Modelling Language (DSML) to specify QoS metrics on non-functional properties. This DSML builds on the concepts discussed in the previous chapters and provides users with a modelling tool to write and run high-level descriptions for QoS estimation. Basically, we take the following approach. Users can model context variables and, from them, observations of relevant context patterns. At runtime, these observations can provide evidence that reinforce (or undermine) the belief that the system is optimal (or not) in terms of one or more non-functional properties. Finally, QoS metrics quantify this belief by expressing the estimated degree of fulfilment of each property as a number.

6.1 Supporting the role of QoS Engineers

Our proposal aims to help QoS Engineers with tasks such as (1) the evaluation of non-functional requirements, e.g., to check statements like "the robot should perform, at least, *well* with respect to *safety*"; (2) benchmarking, e.g., to test the impact of different robot realizations on *user engagement*; and (3) self-adaptation, e.g., the robot could select, at runtime, its navigation strategy based on *power consumption*. The success of all these tasks relies primarily on the ability to estimate the performance of the system in terms of non-functional properties, such as the previously mentioned safety, user engagement and power consumption. In this sense, the proposed DSML provides QoS Engineers with an easy-to-use modelling tool and the necessary runtime artifacts to deal with the estimation of non-functional properties. More specifically, our proposal provides the following benefits.

- QoS Engineers can use the DSML for any application domain. The concepts it includes (such as context variable, non-functional property, or observation) are domain independent.
- QoS Engineers can write domain-specific definitions in terms of reusable models, which can be imported later when modelling a particular application. In this sense, the DSML promotes the reuse of models by including in the language the ability to parameterize and import models.
- QoS Engineers can easily assimilate the language thanks to its simple textual syntax. In this regard, the DSML limits the use of numerical representations, promoting a more natural form of expression for users, where qualitative descriptions predominate over quantitative ones. For example, the strength of an observation could be defined as "high" and its persistence as "long".
- QoS Engineers can use the language to model complex context patterns. Although this is not the focus of this Thesis, the DSML includes primitives to process time series data. Thus, users could define that an observation occurs when a context variable follows a particular sequence of updates.
- QoS Engineers can specify causal relationships between context and non-functional properties in terms of observations. As we saw in Chapter 4, a probabilistic network will be transparently derived from these definitions, so the DSML presents a high level of abstraction that hides all the complexities of this probabilistic network.
- QoS Engineers can automatically generate runtime artifacts from their models. These artifacts can be integrated into their systems to allow the monitoring of input contexts and produce QoS estimates for each non-functional property. In addition, the artifacts will also provide engineers with the statistics developed in Section 5.2. The way in which all this information is used is entirely up to the QoS Engineers.

6.2 Regarding the DSML specification

Over the last years, the *Eclipse Modeling Tools* (EMT) [60] package has become the de facto standard for the development of modelling languages, graphical and textual model editors and

model transformations. Among the tools included in EMT, we have used the *Eclipse Modelling Framework* (EMF) (version 4.25.0) [61] to define the abstract syntax of the language in terms of metamodels. Moreover, the concrete textual syntax was created using the *Xtext* framework (version 2.29.0) [62], from which we generated a complete infrastructure including a parser, a linker, and a textual model editor for Eclipse with syntax colouring, auto-completion, and on-the-fly syntactic model validation, among other features.

The DSML specification is organized into three packages:

- The *Datatypes* package, which provides users with the basics of the modelling language including data types, typed variables and values.
- The *Expressions* package, which allows the definition of logical, arithmetical and pattern expressions.
- The *Kernel* package, which specifies the main modelling concepts, such as, context variables, properties, and observations.

The following sections will address the specification of these packages.

6.3 The Abstract syntax of the language

This section presents the specification of the abstract syntax of the proposed language in terms of metamodels. A metamodel is a model that describes the concepts of a language and the relationships between them. Basically, the procedure that we have followed to create the metamodels in Eclipse has been the following: (1) we have expressed the metamodels graphically with the Ecore editor included in EMF, and (2) we have generated the Java implementation of the metamodels. Note that this implementation will allow the modelling editor to internally represent the models that are created with the language.

6.3.1 The Datatypes metamodel

Figure 28 shows the Datatypes metamodel. This metamodel contains the definition of the basic elements of the DSML, such as the root element and the concepts related to data types.

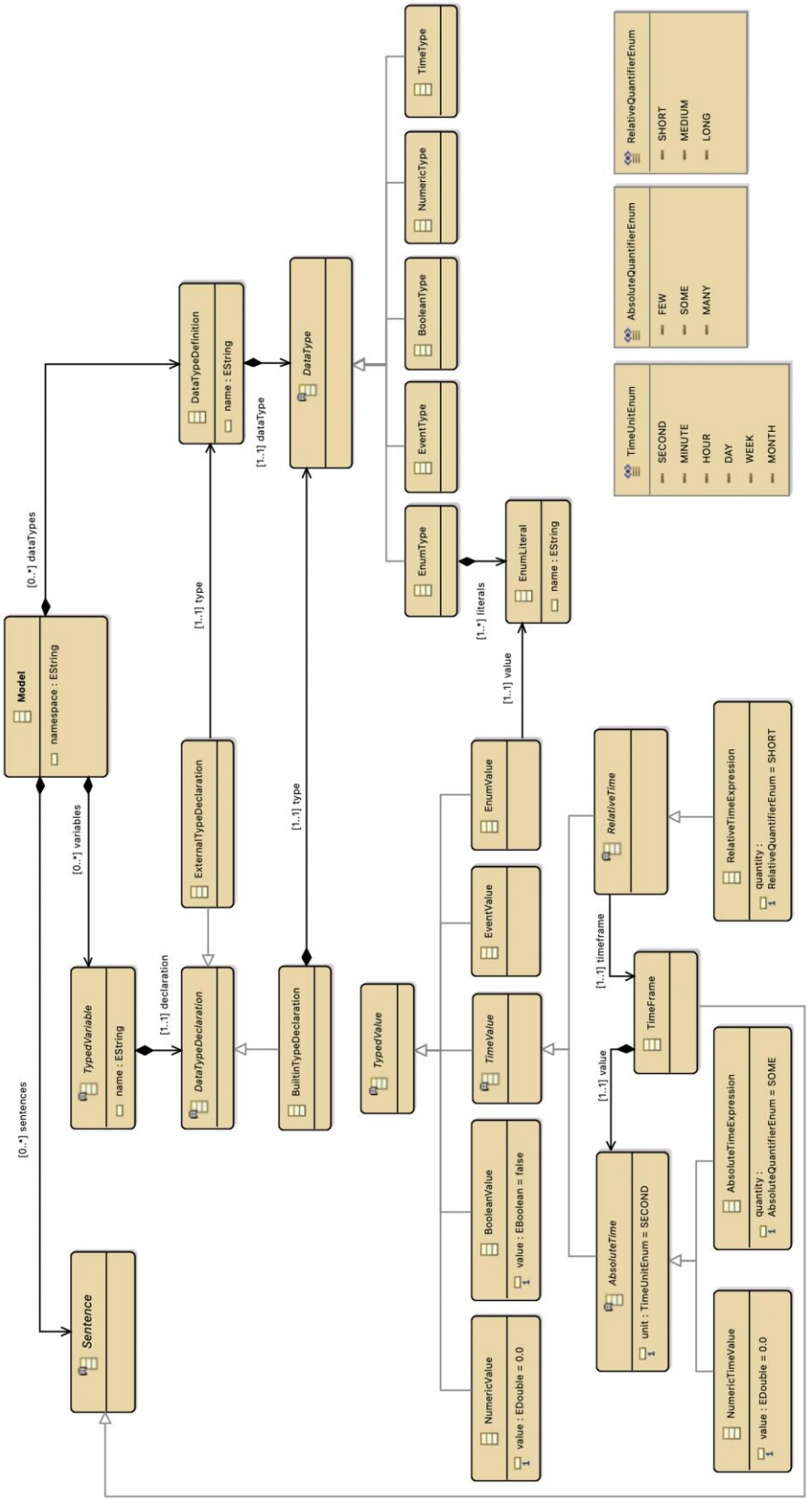


Figure 28. Datatypes metamodel

Next, we describe the elements included in the metamodel.

- *Model*: root class of a model, whose main functionality is to contain the elements belonging to the same specification. In this regard, a model can only contain definitions of data types, variables, and sentences. Also, note that users can define a namespace, which is used to form unique identifiers and avoid name conflicts, for example, when importing a model.
- *Sentence*: abstract class to represent sentences.
- *DataTypeDefinition*: represents a user definition of a data type that is intended to be reused in several variable declarations.
- *DataType*: abstract representation of a data type.
- *BooleanType*: Boolean data type.
- *Numeric Type*: data type for real numbers.
- *TimeType*: data type for time values, not only represented numerically but also with linguistic expressions (as we saw in section 4.1).
- *EventType*: data type for variables that send or receive events.
- *EnumType*: represents an enumerated data type consisting of a list of literals.
- *EnumLiteral*: denoted a named value in an enumeration.
- *TypedVariable*: abstracts a variable belonging to a data type.
- *DataTypeDeclaration*: generic class that represents the data type declaration of a variable.
- *BuiltinTypeDeclaration*: used when the declaration of a variable contains the definition of the data type.
- *ExternalTypeDeclaration*: used when the declaration of a variable references a data type created by the user.
- *TypedValue*: abstracts a value belonging to a data type.
- *BooleanValue*: represents a Boolean value.
- *NumericValue*: represents a real number.
- *EventValue*: denotes the occurrence of an event.
- *EnumValue*: references a literal from an enumeration.
- *TimeValue*: abstracts the notion of time value.

- *AbsoluteTime*: class to abstract the concept of absolute time value. It introduces the unit of measurement as an attribute.
- *NumericTimeValue*: allows the use of numeric values, such as “13 seconds”.
- *AbsoluteTimeExpression*, enables linguistic time expressions, such as “few seconds”.
- *TimeFrame*, sentence to represent a reference period defined in terms of an absolute time value.
- *RelativeTime*, abstract class that helps us express relative time values taking a *TimeFrame* instance as a reference.
- *RelativeTimeExpression*, it enables relative linguistic time expressions, such as “short time”.
- *TimeUnitEnum*: units of time.
- *AbsoluteQuantifierEnum*: enumeration to quantify time in *AbsoluteTimeExpression*.
- *RelativeQuantifierEnum*: enumeration to quantify time in *RelativeTimeExpression.*

6.3.2 The Expressions metamodel

The Expressions metamodel contains the concepts needed to form mathematical expressions, such as arithmetic, relational, or logical operators. In addition, it also includes special operators to define patterns in time series data, which are useful for creating context observations. Figure 29 shows the proposed metamodel, which has the following elements:

- *Expression*: represents a mathematical expression that can be defined with pattern, arithmetic, relational and logical operators.
- *Term*: abstract class for representing the building blocks of an expression. A term can identify an operator, a constant, or a variable.
- *ConstTerm*: term for including constant values in expressions.
- *VarTerm*: term for including variable references in expressions.

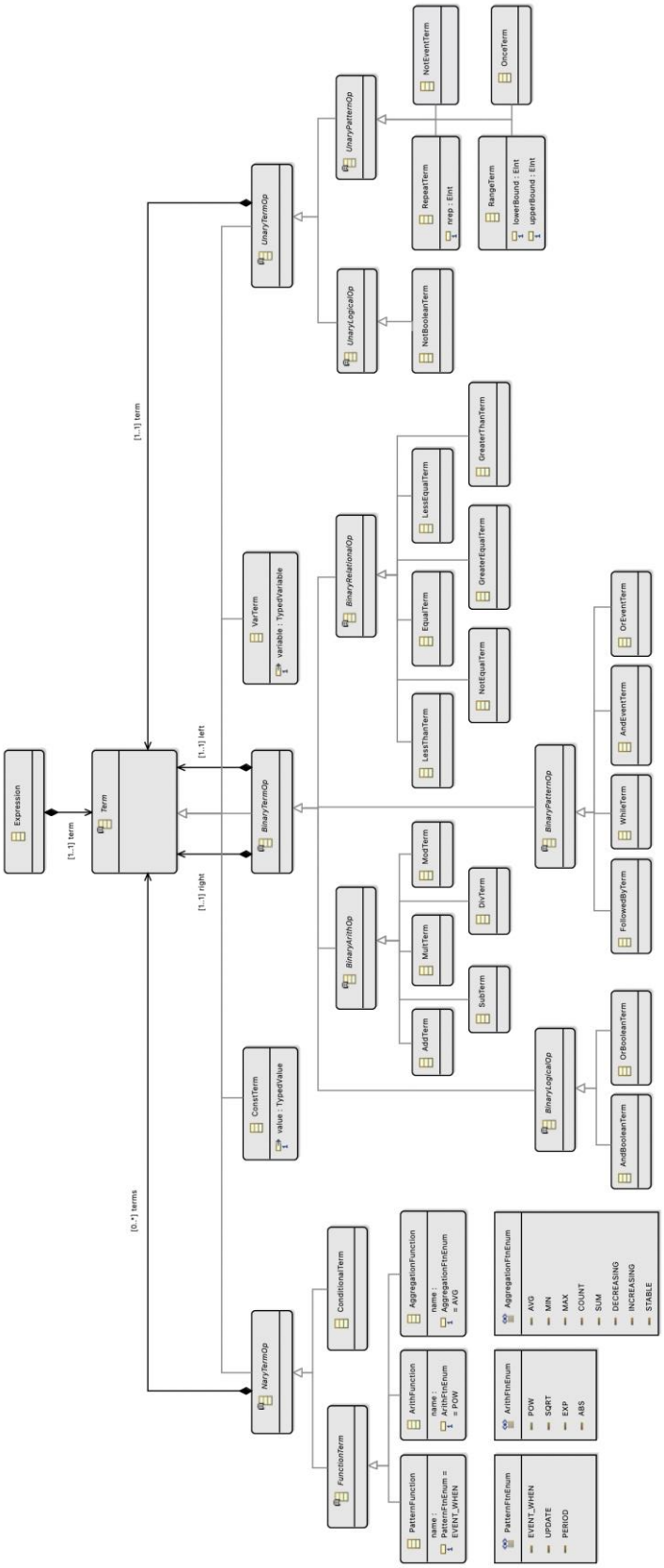


Figure 29. Expressions metamodel

- *UnaryTermOp*: represents unary operators (i.e., those that contain a single term).
- *UnaryLogicalOp*: parent class for unary logical operators.
- *NotBooleanTerm*: logical *NOT* operator.
- *UnaryPatternOp*: parent class for unary pattern operators.
- *OnceTerm*: satisfied the first time a pattern is detected.
- *NotEventTerm*: checks for the absence of a pattern.
- *RepeatTerm*: operator that defines how many times a pattern must occur.
- *RangeTerm*: operator that specifies the minimum and maximum number of times a pattern must occur.
- *BinaryTermOp*: represents binary operators (i.e., those that contain two terms).
- *BinaryLogicalOp*: parent class for binary logical operators.
- *AndBooleanTerm*: logical *AND* operator.
- *OrBooleanTerm*: logical *OR* operator.
- *BinaryArithOp*: parent class for binary arithmetical operators.
- *AddTerm*: addition operator.
- *SubTerm*: subtraction operator.
- *MultTerm*: multiplication operator.
- *DivTerm*: division operator.
- *ModTerm*: remainder operator.
- *BinaryRelationalOp*: abstracts binary relational operators.
- *LessThanTerm*: “less than” operator.
- *LessEqualTerm*: “less than or equal to” operator.
- *EqualTerm*: “equal to” operator.
- *GreaterEqualTerm*: “greater than or equal to” operator.
- *GreaterTerm*: “greater than” operator.
- *BinaryPatternOp*: abstracts binary pattern operations.
- *FollowedByTerm*: determines that the occurrence of a pattern must be followed by another occurrence.
- *WhileTerm*: checks for a pattern while the condition (of another pattern) is evaluated to true.

- *AndEventTerm*: checks for the simultaneous occurrence of two patterns.
- *OrEventTerm*: checks for the occurrence of at least one of two patterns.
- *NaryTermOp*: abstract class for representing a *n*-ary operator (i.e., those that contain any number of terms).
- *ConditionalTerm*: ternary operator. The first term is a comparison argument, the second is the result if the comparison is true, and the third is the result if it is false.
- *FunctionTerm*: abstract class for representing a function.
- *PatternFunction*: represents a pattern function.
- *PatternFtnEnum*: enumeration of *n*-ary pattern functions: *EVENT_WHEN* (produces an event when the specified Boolean condition changes to true), *UPDATE* (event when the argument receives an update), and *PERIOD* (returns the period of time since the last update in the past).
- *ArithFunction*: represents an arithmetical function.
- *ArithFtnEnum*: enumeration of *n*-ary arithmetical functions: *POW* (power), *EXP* (exponential), *SQRT* (square root), and *ABS* (absolute value).
- *AggregationFunction*: represents an aggregation function. An aggregation function takes a set of values and groups them to form a single value.
- *AggregationFtnEnum*: enumeration of *n*-ary aggregation functions: *AVG* (average), *MIN* (minimum), *MAX* (maximum), *COUNT* (number of values), *SUM* (summation), *DECREASING* (returns true when the trend of the values is descending), *INCREASING* (true when the trend of the values is ascending), and *STABLE* (true when the values are stable).

6.3.3 The Kernel metamodel

The Kernel metamodel contains the main modelling concepts of the language. Figure 30 shows the metamodel.

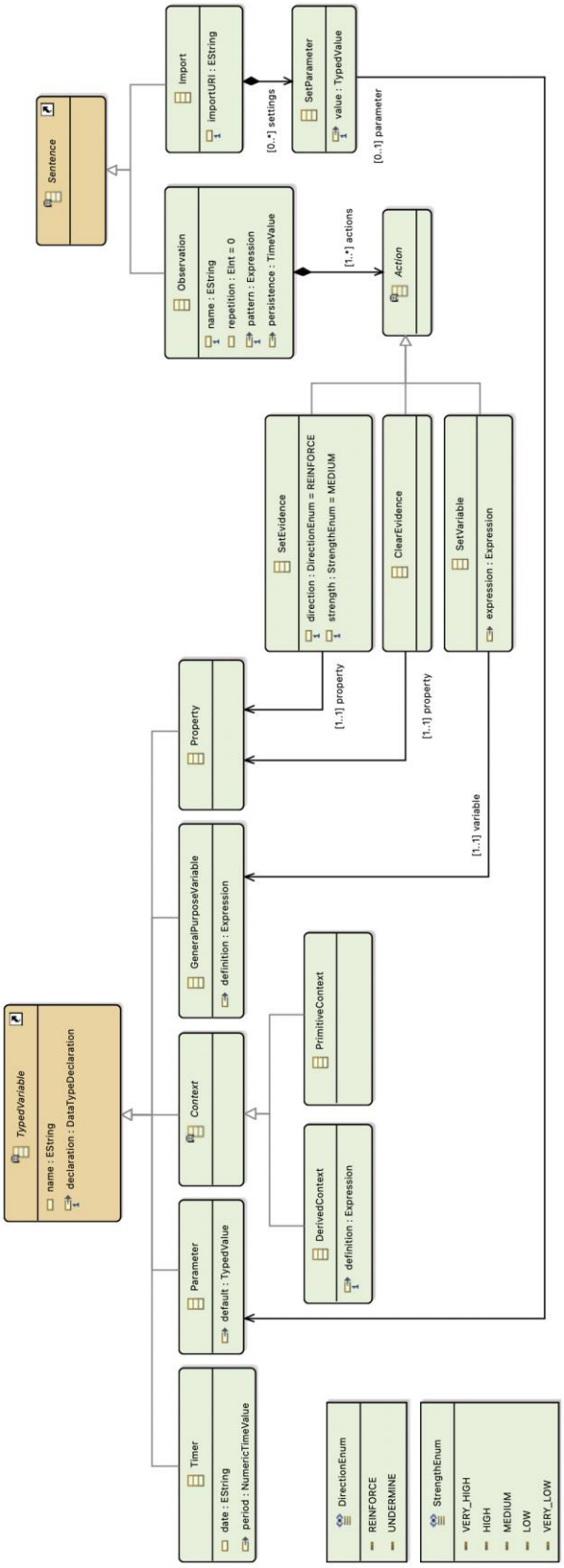


Figure 30. Kernel metamodel

- *Property*: defines a non-functional property. A property has been modelled as a *TypedVariable* since it represents the corresponding QoS metric from which we can read the current QoS estimate. Thus, properties must always be declared as *NumericType*.
- *GeneralPurposeVariable*: general-purpose variable for auxiliary operations, creation of state variables, among other uses.
- *Parameter*: special type of variable to define configuration parameters. They are created with default values that can only be changed when importing the model into another (see *SetParameter*), which allows users to adjust the model when it is reused.
- *Timer*: special type of variable that triggers an event after a specified date or period. The data type of this variable must be *EventType*.
- *Context*: abstract class for representing a context variable.
- *PrimitiveContext*: variables that take context data monitored by the system (e.g., from sensors or other components).
- *DerivedContext*: context variable that can be expressed in terms of one or more *PrimitiveContext* (and any other type of variable, such as *GeneralPurposeVariable* or *Timer*).
- *Import*: allows importing other models to reuse data types, variables, or observations.
- *SetParameter*: allows users to adjust the value of a parameter defined in an importing model.
- *Observation*: named sentence that identifies a context condition as a *pattern* expressed in terms of one or more context variables. The detection of this pattern can trigger *actions*. Moreover, users can optionally configure the following attributes to adjust how the occurrences of an observation behave over time: (1) *persistence*, it specifies the maximum period in which the effect of an occurrence remains with some influence until it disappears; and (2) *repetition*, which indicates how many times an observation must be repeated to have full effect.
- *Action*, abstract class to represent an action that takes place when an observation occurs.
- *SetEvidence*: provides evidence that the system behaviour is optimal (or not) in terms of a non-functional property (linked by *property*). The effect of this evidence is

established according to the attributes: (1) *direction*, which indicates the orientation of the evidence; and (2) *strength*, which qualitatively determines the intensity of the evidence.

- *DirectionEnum*: enumeration to establish the orientation of the evidence, *REINFORCE* (to increase) or *UNDERMINE* (to reduce) the belief that the system performs optimally.
- *StrengthEnum*: enumeration to describe the intensity of the evidence considering five different levels: from *VERY_LOW* to *VERY_HIGH*.
- *ClearEvidence*: removes all previous evidence (which was set with *SetEvidence*).
- *SetVariable*: allows setting a general-purpose variable when an observation occurs.

6.4 The textual concrete syntax of the language

This section addresses the textual concrete syntax of the proposed language. As we have already mentioned, the language was developed using the Xtext framework, which allows us to generate a complete model editor for Eclipse. Basically, the procedure that we have followed to create the editor has been the following: (1) we have created a new Xtext project using the metamodels from the previous section, (2) we have written the grammar for each of the concepts included in the metamodels, and (3) we have generated the language artifacts.

6.4.1 A language walkthrough

Next, we are going to introduce how the main elements of the language are expressed, which will be presented in a simple way using the hospital robot example from Section 3.3.

6.4.1.1 Context variables

We are going to model, in a file called *contexts.qosme*, some of the context variables we saw in Table 1. Note that the extension *qosme* identifies the models created with the DSML.

At the beginning of the model, we have the option to define the namespace. In that case, when importing the model, we would need to prepend this namespace to reference any of the imported variables, e.g. `contexts.batteryLevel`. Regarding the definition of primitive

context variables, lines 10-13 of Listing 11 show 4 variables, declared with the keyword `context` followed by the variable name and the data type. These variables use built-in data types except for `state`, whose data type is defined separately (see line 7). As we will see in Section 6.6, each primitive context variable will be linked to a context monitor in the system, which will be responsible for providing updates at runtime. As for the definition of derived context variables, line 16 shows `attentionLoss`, which becomes `true` when `eyeContact` changes more than `lossAttThreshold` times in less than 1 minute, indicating that the user is not paying attention to the robot. The `lossAttThreshold` parameter is a variable whose default value can only be changed at the time of importing the model.

```
1   namespace contexts
2
3   // Parameter
4   param lossAttThreshold : number default 8
5
6   // Data type definition
7   type StateEnum : enum {IDLE, GREETING, SERVING, PARTING, DOCKED}
8
9   // Primitive context variables
10  context state : StateEnum
11  context collision : event
12  context batteryLevel : number
13  context eyeContact : boolean
14
15  // Derived context variable
16  context attentionLoss : boolean
17     = count(eyeContact, 1min) > lossAttThreshold
```

Listing 11. Content of contexts.qosme

6.4.1.2 Properties

Let us create a new model called *example.qosme*. Listing 12 shows its content. The first thing in the model is to import *contexts.qosme* so that we can use the context variables defined in it. Besides, we take the opportunity to adjust the parameter `lossAttThreshold` with a new value (see line 2).

Lines 7-8 shows the declaration of two properties with the keyword `property` followed by a name. We do not have to define the data type as it is fixed to `NumericType` in the grammar. Remember that a property stores the current value of the corresponding QoS metric.

6.4.1.3 Observations

Lines 11-21 in Listing 12 show the definition of three observations. They are expressed with the keyword `observation` followed by (1) a name, (2) a context pattern, (3) optionally, the configuration of the persistence and the repetition, and finally (4) one or more actions. These observations correspond to the "People lose interest", "Bump into someone" and "Stranded without battery" observations in Table 4, respectively. Note that some of the observations use relative time expressions to set the persistence (see lines 16 and 20), which requires the definition of the timeframe (line 4).

```
1  import "contexts.qosme"
2     with lossAttThreshold = 10
3
4  timeframe 7 days
5
6  // Properties
7  property userEngagement
8  property safety
9
10 // Observations
11 observation obs1 : contexts.attentionLoss & contexts.state == SERVING
12    persistence few hours
13    undermines userEngagement
14
15 observation obs2 : contexts.collision
16    persistence long
17    undermines safety very high
18
19 observation obs3 : contexts.batteryLevel<1 & contexts.state != DOCKED
20    persistence medium repetition 5
21    undermines safety high
```

Listing 12. Content of example.qosme

6.4.1.4 Other resources: general-purpose variables and timers

Consider that the robot is put into maintenance every day from 7 a.m. to 8 a.m. When this happens, we want to (1) disable some observations to avoid accumulating false evidence and (2) reset the properties to start the day with clean statistics. Listing 13 presents a new model that illustrates how to express general-purpose variables and timers using this example. First, line 8 shows the definition of a general-purpose variable to indicate if the robot is under maintenance. This is declared using the keyword `var` followed by a name, a data type, and an

expression to be evaluated. Moreover, lines 11-12 shows the definition of two timers. A timer is declared using the keyword `timer` followed by a name and a value. The value can be a period expressed numerically (e.g., “3 seconds”, which means that the timer will go off every 3s) or a string to express a date or time (as in the example, “7:00h”, which means that the timer will be activated every day at 7:00 a.m.). Note that we do not have to indicate the data type of a timer as it is fixed to `EventType` in the grammar. Finally, observations `ob1` and `ob2` control the value of `underMaintenance`, and reset the properties when the maintenance is finished. The last observation will be disabled whenever the robot is under maintenance.

```
1   import “contexts.qosme”
2
3   // Properties
4   property userEngagement
5   property safety
6
7   // General-purpose variable
8   var underMaintenance : boolean = false
9
10  // Timers
11  timer startMaintenance = “7:00h”
12  timer endMaintenance = “8:00h”
13
14  // Observations
15  observation obs1 : startMaintenance
16     sets underMaintenance = true
17
18  observation obs2 : endMaintenance {
19     sets underMaintenance = false
20     clears userEngagement
21     clears safety
22  }
23  observation obs3 : !underMaintenance & contexts.collision
24     undermines safety very high
```

Listing 13. Content of example2.qosme

6.4.2 The grammar specification

Listing 14 shows the EBNF specification of the language grammar. Although the textual concrete syntax was defined using Xtext, EBNF provides a more compact representation of the grammar. The complete specification developed with Xtext can be found in A.1.

Model	::=	[**'namespace'** ID]
		(**'import'** STRING ('with' ID '=' Value)*)*
		[**'timeframe'** (NumericTimeValue | AbsoluteTimeExpression)]
		(DataTypeDefinition | TypedVariable | Observation) +
DataTypeDefinition	::=	**'type'** ID **':'** DataType
DataType	::=	EnumType | NumericType | TimeType | BooleanType | EventType
EnumType	::=	**'enum'** **'{'** ID (**','** ID)* **'}'**
NumericType	::=	**'number'**
TimeType	::=	**'time'**
BooleanType	::=	**'boolean'**
EventType	::=	**'event'**
TypedValue	::=	EnumValue | NumericValue | BooleanValue | EventValue | TimeValue
EnumValue	::=	ID **'::'** ID
EventValue	::=	**'trigger'**
NumericValue	::=	INT | DOUBLE
BooleanValue	::=	**'true'** | **'false'**
TimeValue	::=	NumericTimeValue | AbsoluteTimeExpression | RelativeTimeExpression
NumericTimeValue	::=	NumericValue TimeUnitEnum
AbsoluteTimeExpression	::=	(**'few'** | **'some'** | **'many'**) TimeUnitEnum
RelativeTimeExpression	::=	**'short'** | **'medium'** | **'long'**
TimeUnitEnum	::=	**'seconds'** | **'minutes'** | **'hours'** | **'days'** | **'weeks'** | **'months'** |
		'second' | **'minute'** | **'hour'** | **'day'** | **'week'** | **'month'**
TypedVariable	::=	GeneralPurposeVariable | Context | Property | Parameter | Timer
GeneralPurposeVariable	::=	**'var'** ID **':'** (ID| DataType) '=' Expr
Context	::=	PrimitiveContext | DerivedContext
PrimitiveContext	::=	**'context'** ID **':'** (ID | DataType)
DerivedContext	::=	**'context'** ID **':'** (ID | DataType) '=' Expr
Property	::=	**'property'** ID
Parameter	::=	**'param'** ID **':'** (ID | DataType) ['default' TypedValue]
Timer	::=	**'timer'** ID '=' (NumericTimeValue | STRING)
Observation	::=	**'observation'** ID **':'** Expr
		['repetition' INT] ['persistence' TimeValue]
		('{' Action+ '}') | Action
Action	::=	SetEvidence | ClearEvidence | SetVariable
SetEvidence	::=	(**'reinforces'** | **'undermines'**) ID
		(**'very low'** | **'low'** | **'medium'** | **'high'** | **'very high'**)
ClearEvidence	::=	**'clears'** ID
SetVariable	::=	**'sets'** ID '=' NumericTimeValue
Expr	::=	'once' Expr | Expr 'repeat' **'('** INT **')'**
	|	Expr 'range' **'('** INT **','** INT **')'** | Expr 'while' **'('** Expr **')'**
	|	Expr **'->'** Expr | Expr ('or' | 'and') Expr

```
|       'not' Expr                          | Expr '?' Expr ':' Expr
|       Expr ('|' | '$') Expr               | '!' Expr
|       Expr ('<' | '>' | '<=' | '>=' | '==') Expr
|       Expr ('+' | '-' | '*' | '/') Expr   | '(' Expr ')'
|       [ID ' ' '] ID '(' Expr ( ',' Expr )* ')'
|       ID                                  | TypedValue
```

Listing 14. EBNF specification

6.5 Validation of the models

The model editor, developed with Xtext, provides three different types of validations based on the grammar specification we have created.

6.5.1 Syntactical correctness

The parser validates any textual input, including the format of attribute values. For example, in the Kernel metamodel (see Figure 30), the *Observation* metaclass defines *repetition* as an integer. We could have used OCL [63] to restrict this value to positive numbers, but we did it more conveniently through the grammar. Listing 15 shows an excerpt from the grammar in which the INT rule does not allow us to type a negative sign. If we do, the editor will generate an error message.

```
Observation returns kernel::Observation:
  'observation' name=ID ':' pattern=Expression
    (('repetition' repetition=INT)?
    ...
terminal INT returns ecore::EInt:
  ('0'..'9')+;
```

Listing 15. Snippet of the grammar to show how the attribute "repetition" does not allow negative values

6.5.2 Cross-reference validation

The editor incorporates a linker to handle cross-references, for example, to allow the use of a variable already defined. The default mechanism is not appropriate for some of the language elements. For instance, *SetParameter*, in the Kernel metamodel (see Figure 30), allows users

to configure the value of the parameters defined in the model that is imported. However, the linker does not check the origin of the references, so the parameters might not come from the imported model. To fix it, we had to extend the Java implementation of the editor.

6.5.3 Concrete syntax validation

The serializer validates all constraints that are implied by the grammar. Thus, by tuning the grammar, it is possible to introduce many of the constraints that need to be placed on the metamodel. For example, the Kernel metamodel (see Figure 30), extends *TypedVariable* to define *Property* and *Timer* as two special variables. We could have added in the abstract syntax a constraint to ensure that a property is always declared as *NumericType* and a timer as *EventType*. However, we did it easily through the grammar. Listing 16 and Listing 17 shows an excerpt from the grammar that addresses this issue for properties and timers, respectively. Basically, the solution is to assign the type directly when creating a property or a timer.

```
Property returns kernel::Property:
  'property' name=ID declaration=ImplicitNumericTypeDeclaration;

ImplicitNumericTypeDeclaration returns datatypes::BuiltinTypeDeclaration:
  type=ImplicitNumericType;

ImplicitNumericType returns datatypes::NumericType:
  {datatypes::NumericType};
```

Listing 16. Snippet of the grammar to show how "declaration" is implicitly set in properties

```
Timer returns kernel::Timer:
        'timer' name=ID declaration = ImplicitEventTypeDeclaration
        '=' (period=NumericTimeValue | date=EString);

ImplicitEventTypeDeclaration returns datatypes::BuiltinTypeDeclaration:
        type=ImplicitEventType;

ImplicitEventType returns datatypes::EventType:
        {datatypes::EventType};
```

Listing 17. Snippet of the grammar to show how "declaration" is implicitly set in timers

6.6 Runtime support

This section presents the artifacts that allow the estimation of QoS metrics, necessary to run the models created according to the specification covered in the previous sections.

6.6.1 Overall process

Figure 31 shows the overall process for enabling the estimation of QoS metrics. The process is as follows:

1. At design-time, users create valid models using the textual editor developed with the Xtext framework.
2. From these models, users can generate two components: the *Event Processor* and the *QoS Metrics Estimator*.
3. At runtime, *context monitors* collect contextual data (from the system, its environment, other systems, etc.) and share it as context events in the *Publish/Subscribe Data Space*. Context monitors are not automatically generated as they will usually depend on the system architecture. Note that the proposed modelling language is platform-independent, so models does not contain information about the architecture. However, Section 6.6.2 will provide the message format specification so that developers can easily integrate their monitors.
4. The *Event Processor* receives context events and searches for the patterns specified in the model. When found, produces an observation event. It is worth noting that context events are updates of primitive context variables.
5. The *QoS Metrics Estimator* receives the observations and produces QoS estimates, which quantify the degree of fulfilment of the non-functional properties.
6. *QoS consumers* subscribe to QoS estimates to use them in, for example, a visualization tool that generates graphs to help engineers analyse the performance of the system.
7. Like context monitors, Section 6.6.2 will provide the message format specification so that developers can integrate their QoS consumers.

The following sections describe in more detail the Publish/Subscribe Data Space, the Event Processor, and the QoS Metrics Estimator.

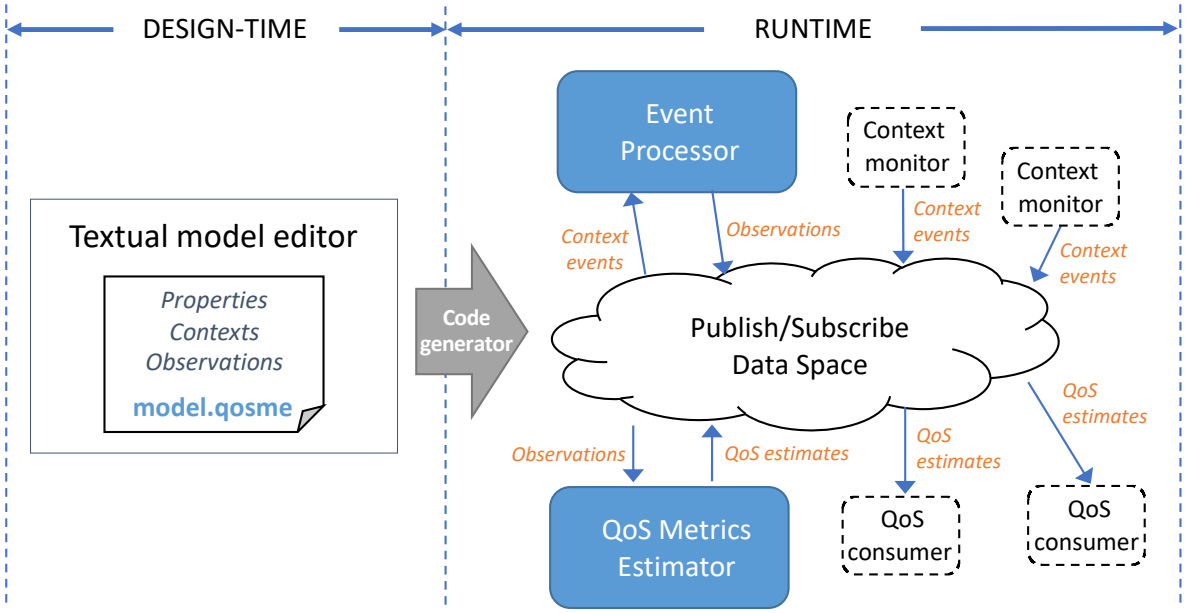


Figure 31. Overall process for enabling the estimation of QoS metrics

6.6.2 Publish/Subscribe Data Space

The process introduced in the previous section involves several components sharing information across the Publish/Subscribe Data Space. This data space is achieved through a publish/subscribe middleware, such as the Data Distribution Service (DDS) [64] or the Message Queuing Telemetry Transport (MQTT) [65].

In these platforms, the information is organized in a hierarchy of topics, which identifies data of a particular type. When a publisher has a new item of data on a topic, it is sent to the middleware, which distributes the information to any client subscribed to the topic. The publisher does not need to know about the number or location of the subscribers, and the subscribers, in turn, do not have to be configured with any data about the publisher. Moreover, this flexibility allows system builders to seamlessly deploy the components on different scenarios. For instance, in the hospital robot example, the Event Processor and the QoS Metrics Estimator could be executed in a different computer to save computational resources in the robot, and without further configuration. This is also the case for context monitors and QoS consumers, they can run outside the robot.

We have defined 5 topics for the Publish/Subscribe Data Space: (1) `qosmetrics/context` for context data; (2) `qosmetrics/observation` for observations; (3) `qosmetrics/estimate` for QoS estimates; (4) `qosmetrics/estimateStats` for statistics related to QoS estimates; and (5) `qosmetrics/observationStats` for statistics related to observations. Listing 18 shows the message format associated with each topic. This definition has been made with Protocol buffers [66], which is a language-neutral, platform-neutral, and extensible mechanism for serializing structured data.

```
message Context {
   string name = 1;
   oneof value {
      int32 intValue = 2;
      uint32 uintValue = 3;
      double doubleValue = 4;
      bool boolValue = 5;
      string stringValue = 6;
   }
}

message Observation {
   string name = 1;
}

message Estimate {
   string property = 1;
   double value = 2;
}

message EstimateStats {
   string property = 1;
   double max = 2;
   double min = 3;
   double mean = 4;
   map<string,double> evidenceSupport = 5;
}

message ObservationStats {
   string observation = 1;
   double n = 2;
   double mean = 3;
}
```

Listing 18. Message format specified in Protocol buffers

The `Context` message contains two fields: the name of the corresponding context variable and value, which is selected according to its type. The `Observation` message only needs the name of the observation and `Estimate` the name and the estimate value of the property. Note that all the names must match the ones defined in the model. As for the statistics, the QoS Metrics Estimator will publish 2 types of messages: `EstimateStats` and `ObservationStats.` The former contains the name of the property, the central tendency and variability of the estimates

(in terms of `mean`, `max` and `min`, calculated according to Section 5.2.1), and the percentage to which each observation supports the current estimate (`evidenceSupport`, see Section 5.2.3). On the other hand, `ObservationStats` contains the name of the observation, the average number of active occurrences (*n* in Listing 18), and its temporal mean (computed according to Section 5.2.2).

6.6.3 Event Processor

The Event Processor is aimed at detecting the context patterns specified in the model and, when found, producing the corresponding observations. This is accomplished through the Publish/Subscribe Data Space, as this component subscribes to context updates and publishes observations.

6.6.3.1 Semantics of the pattern expressions

Before addressing the implementation of the Event Processor, we need to establish the semantics of the elements used to create pattern expressions, which mainly involves aggregate functions and pattern operators. Table 20 summarizes the aggregate functions available in the language. As for the pattern operators, Table 21 shows their specification.

Table 20. Aggregate functions

Aggregate functions	Description
mean, sum, min, max **observation** o1 : variable'mean(5)>80 ...	Applied to numerical variables, they return a number after calculating the mean, sum, minimum, or maximum, respectively. The window size is an argument. E.g., *o1* triggers if the mean of *variable* over its last 5 updates exceeds 80.
stable, increasing, decreasing **observation** o1 : variable'increasing(1 minute) ...	Applied to numerical variables, they return a Boolean. They are *true* when all the values remain constant, increase, or decrease, resp. The window size and the tolerance are arguments. The latter indicates the minimum variation to consider an increase or decrease. E.g., o*1* triggers if *variable* constantly increases for 1 minute.
count **observation** o1 : variable'count(1 minute)>6 ...	Applied to any variable. *Count* gets the number of updates of a variable. E.g., *o1* triggers if *variable* get more than 6 updates in 1 minute.

Notes: Aggregate functions are computed on a set of past values using a sliding window, which can be specified by (1) period or (2) number of samples. Highlight that the notation `arg1'function(arg2,...)` is equivalent to `function(arg1, arg2,...)`. It is used to emphasize the target variable of the aggregate function.

Table 21. Pattern operators

Operator	Description
update(*expr*) **observation** o1 : **update**(anyVariable) ...	Satisfied when at least a variable in *expr* is updated. E.g., observation o*1* triggers whenever *anyVariable* changes. Formally, $\langle w,i\rangle \vDash update\big(expr(V')\big) \Leftrightarrow \exists v_j \in V' \mid w \vDash v_j \ (V' \subseteq V)$
eventWhen(*expr*) **observation** o1 : **eventWhen**(boolVariable) ...	Satisfied when a Boolean expression changes to true. E.g., observation o*1* triggers whenever the state of *boolVariable* changes to true. Formally, $\langle w,i\rangle \vDash eventWhen(expr) \Leftrightarrow$ $\langle w,i\rangle \vDash update(expr) \wedge expr = true$
expr **observation** o1 : boolVariable ...	It means *eventWhen(expr)* if *expr* is Boolean, *update(expr)* otherwise. E.g., observation *o1* is equivalent to the example seen above. $\langle w,i\rangle \vDash expr \Leftrightarrow \begin{cases} \langle w,i\rangle \vDash eventWhen(expr) & Boolean\ expr. \\ \langle w,i\rangle \vDash update(expr) & otherwise \end{cases}$
expr1 **or** *expr2* **observation** o1 : intVariable < 50 **or** boolVariable ... **observation** o2 : intVariable < 50 | boolVariable ...	Satisfied if *expr1* or *expr2* is met. E.g., o*1* is executed if *intVariable* is updated with a value lower than 50 or if *boolVariable* is updated to true. Note that this is an event-typed operator, not a Boolean one (denoted by '|' instead). The operands must be events, so *o1* implicitly is: `eventWhen(intVariable<50) or eventWhen(boolVariable)`, while *o2* is: `eventWhen(intVariable<50 | boolVariable)` $\langle w,i\rangle \vDash expr1\ or\ expr2 \Leftrightarrow \langle w,i\rangle \vDash expr1 \vee \langle w,i\rangle \vDash expr2$
expr1 **and** *expr2* **observation** o1 : intVariable > 50 **and** eventVariable ... **observation** o2 : intVariable > 50 **&** intVariable < 80 ...	Satisfied when *expr1* and *expr2* are met (not necessarily at the same time). E.g., observation o*1* triggers when *intVariable* is greater than 50 and, at some point in the past, *eventVariable* occurred (or vice versa). Note that it is an event-typed operator and not a Boolean one (denoted by '&'). E.g., *o2* uses & to express that *intVariable* is updated between two values. $\langle w,i\rangle \vDash expr1\ and\ expr2 \Leftrightarrow \langle w,i\rangle \vDash expr1$ $\wedge\ \exists j < i \mid \langle w,j\rangle \vDash expr2$ (commutative)
not *expr1* **observation** o1 : **not** eventVariable ...	Satisfied if *expr1* is not met. E.g., observation o*1* is executed when *eventVariable* is not updated at the time of checking the observation. $\langle w,i\rangle \vDash not\ expr \Leftrightarrow \langle w,i\rangle \nvDash expr$
expr1 **->** *expr2* **(Followed-by)** **observation** o1 : intVariable>50 **->** intVariable>80 ...	Satisfied when *expr2* is met just after *expr1*. E.g., observation o*1* triggers if *intVariable* exceeds 80 after exceeding 50. $\langle w,i\rangle \vDash expr1 \rightarrow expr2 \Leftrightarrow \langle w,i\rangle \vDash expr2$ $\wedge \exists j < i \mid \langle w,j\rangle \vDash expr1$
expr1 **while**(*expr2*) **observation** o1: intVariable>50 **while** (boolVariable) ... **observation** o2: intVariable >50 **&** boolVariable ... **observation** o3: intVariable >50 **and** boolVariable ...	Satisfied when *expr1* is met while the Boolean *expr2* is true. E.g., observation o*1* is executed when *intVariable* exceeds 50 while boolVariable is true. Note that o*1*, o*2* and o*3* are not the same. The first is only check when *intVariable* is updated, o*2* when *intVariable* or *boolVariable* are updated, and o*3* requires both variables to be updated. $\langle w,i\rangle \vDash expr1\ while(expr2) \Leftrightarrow \langle w,i\rangle \vDash expr1 \wedge$ $expr2 = true$
expr **repeat**(*n*) **observation** o1 : eventVariable **repeat**(3) ...	Satisfied when *expr* is met *n* times. E.g., observation *o1* is executed when *eventVariable* occurs three times. $\langle w,i\rangle \vDash expr\ repeat(n) \Leftrightarrow \langle w,i\rangle \vDash expr \wedge$ $\exists k_1, k_2 \ldots k_{n-1} < i \mid \langle w,k_j\rangle \vDash expr \ (n \in \mathbb{N})$
expr **range**(*a*,*b*) **observation** o1: eventVariable **range**(3,5) …	Satisfied when *expr* is met between *a* and *b* times. E.g., observation *o1* is executed when *eventVariable* occurs between 3 and 5 times (including 3 and 5). $\langle w,i\rangle \vDash expr\ range(a,b) \Leftrightarrow \langle w,i\rangle \vDash expr \wedge$ $\exists\, n, a \le n \le b \mid K_n < i \wedge \langle w,K_j\rangle \vDash expr$ $(a < b\ being\ n,a,b \in \mathbb{N})$
once *expr* **observation** o1: **once** intVaraible >10 …	Satisfied only the first time *expr* is met. E.g., observation o*1* is triggered the first time *intVariable* exceeds 10, after that, it will not be triggered again. $\langle w,i\rangle \vDash once\ expr \Leftrightarrow \langle w,i\rangle \vDash expr$ $\wedge \nexists j < i \mid \langle w,j\rangle \vDash expr$

The semantics of the pattern operators have been described as an extension of the *Linear Temporal Logic* [67]. Let $V = \{v_1, \dots, v_n\}$ be the set of variables defined in a model and $\Sigma = \{\alpha_1, \dots, \alpha_n\}$ the set of atomic propositions, where α_j denotes that the variable v_j was updated. We consider a discrete and linear model of time represented by the structure $\langle w, i \rangle$, where *w* is a temporal sequence of updates and *i* is a natural number that indicates the current time point. E.g., w formed by $w(0) = \alpha_4, w(1) = \alpha_2, w(2) = \alpha_7, w(3) = \alpha_1, \dots$ presents a sequence of updates, each one performed in a different moment in time, such that, the variable v_4 is updated before v_2, this one before v_7, etc. If the current time were $i = 2$, v_7 would take a new value at that moment, the variables v_4 and v_2 would already be updated, and v_1 would be updated in the future. Formally, we say that, at the current time *i*, a variable $v_j \in V$ is updated if it satisfies the following relation $\langle w, i \rangle \vDash v_j \Leftrightarrow \alpha_j \in w(i)$. In Table 21, the terms *expr(V')*, *expr*, *expr1* or *expr2* denote mathematical expressions involving a set of variables *V'* (defined in *V*). Finally, $w[k, j]$ represents a subsequence of *w*, i.e., $w[k, j] = w(k), w(k+1), \dots, w(j-1), w(j)$ where k and *j* are natural numbers being $k \leq j$.

6.6.3.2 Code generation

The methods and techniques necessary to implement the Event Processor are available in the field of Complex Event Processing (CEP) [68], whose goal is to extract information from event streams to identify significant patterns and respond to them as quickly as possible. Most CEP platforms provide an Event Processing Language (EPL) to allow users to describe event pattern expressions. Although there are similarities, our proposal offers a different way of expressing context patterns regarding EPLs. First, contrary to most approaches, pattern expressions in our language are implicitly recursive, that is, they remain active after being satisfied and can be activated multiple times. And second, our proposal is more oriented to variables than to events. While CEP sees event occurrences, our proposal sees variable updates.

Despite the differences, the idea is to rely on CEP for the implementation of the Event Processor. To do this, during the code generation phase, part of the model will be translated into a EPL program. This program will support (1) the evaluation of variables, including derived context, general-purpose variables, and timers. For example, if a user defines a derived

context, the Event Processor comes into play to calculate its value each time the primitive contexts on which it depends are updated; (2) observations as event-typed expressions, which produces an event whenever the pattern is satisfied; and (3) the computation of functions, including aggregate functions, such as, sum, average, maximum and minimum. Figure 32 shows the Kernel metamodel we presented in Section 6.3.3, in which we have identified in red the elements that would be involved in the generation of the Event Processor. In what has to do with the implementation of this generation, our contribution has focused on the language specification. Therefore, the generation of EPL is outside the scope of this Thesis. It was addressed by the RoQME project [69], where Esper [70] was selected as the runtime platform for the Event Processor.

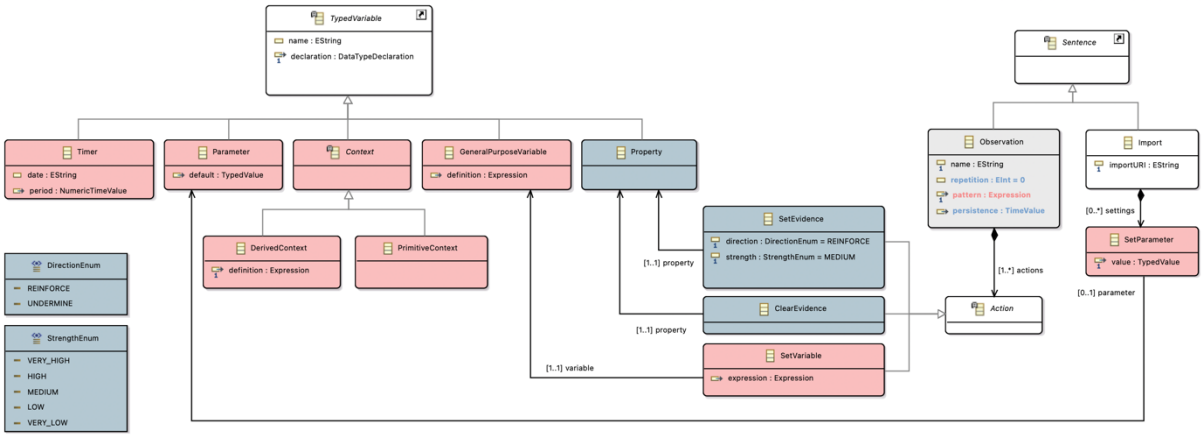


Figure 32. Elements of the Kernel metamodel used by each generator. The elements in red are involved in the generation of the Event Processor and the elements in blue in the generation of the QoS Metrics Estimator

6.6.4 QoS Metrics Estimator

The QoS Metrics Estimator receives notifications when observations are detected and, as a result, publishes estimates of the specified QoS metrics. The implementation of this component involves two elements: (1) a model-to-text transformation that produces a simplified specification with the information only concerning the QoS estimation, and (2) a component that translates this especification into a probabilistic network and runs the QoS metrics.

6.6.4.1 The model-to-text transformation

The goal of this transformation is to remove irrelevant information for the QoS Metrics Estimator. Figure 32 shows the Kernel metamodel presented in Section 6.3.3, in which we can see in blue the elements that matter. That is, everything related to context pattern detection will be filtered out, since the QoS Metrics Estimator only needs to know two things: (1) when an observation occurs and (2) how it impacts on the properties. The transformation is straightforward, it was implemented with Xtend [71] and, as a result, produces a JSON file (see an example in Listing 19).

```
{
        "timeframe" : {"value" : 7, "unit" : "day"},
        "properties" : [ {"name" : "safety"}, {"name" : "userEngagement"}],
        "observations" : [{
                "name" : "Bump into someone",
                "persistence" : {"value" : "long"},
                "evidence" : [{
                        "direction" : "undermines",
                        "strength"  : "very_high",
                        "property"  : "safety"
                }]},...
        ]
}
```

Listing 19. Extract of the JSON file generated from an example model

6.6.4.2 The QoS estimator component

This component is responsible for (1) deriving the parameters and topology of the probabilistic network resulting from the specification in the JSON file, and (2) running the algorithms to obtain QoS estimates and statistics. Note that the first point has been detailed in Chapter 4, while the second in Chapter 5. The component was implemented in Java. However, we also developed an alternative solution based on the neural network model, we saw in Section 5.3, with TensorFlow [55] (but it could be done in any other deep learning framework). This alternative provides great flexibility, since TensorFlow is available in several programming languages and allows the deployment of the model on the web, on mobile phones or on embedded devices.

6.7 Summary

This chapter introduced a Domain Specific Modelling Language (DSML) for specifying QoS metrics on non-functional properties. The DSML is built upon the concepts discussed earlier and gives users a tool for creating and executing high-level specifications of QoS metrics. The following chapter covers the evaluation of this approach with two real scenarios and the characterization of the QoS metrics through simulations.

Chapter 7 Evaluation

In this chapter, we address the evaluation of the proposed approach. First, we will show the application of QoS metrics in two real scenarios. After that, Section 7.2 tests the characteristics of the QoS metrics through simulations.

7.1 Experiments in real-world scenarios

This section shows the application of the proposal in two real robotics scenarios. One related to the movement of goods in an intralogistics environment and the other one using a clinic assistant robot.

7.1.1 Intralogistics Industry 4.0 Robot Fleet Pilot

7.1.1.1 Scenario description

This Thesis has been carried out in the framework of the RoQME Integrated Technical Project [69], funded by the EU H2020 RobMoSys Project [72]. The Ulm University of Applied Sciences (HSU) was the technical lead partner of RobMoSys. HSU made the Intralogistics Industry 4.0 Robot Fleet Pilot [73] available to all the Integrated Technical Projects funded by RobMoSys as a test-bed for experimentation and showcasing. The scenario counts on a fleet of robots which can collect, deliver, and transport objects from/to different workstations, conveyor belts, etc. Robots can interact with each other and with human operators. Figure 33 illustrates some of the capabilities of these robots.

Figure 33. Intralogistics Industry 4.0 Robot Fleet Pilot. (Left) Picking items. (Middle) Transport and autonomous delivery of boxes with goods. (Right) Order picking into boxes. (Source: https://robmosys.eu)

7.1.1.2 Modelling QoS metrics

Listing 20 shows the model that was developed for the scenario. This model specifies QoS metrics on two non-functional properties: `Performance` and `Safety`. It also defines contexts identified as relevant, and observations that indicate which context patterns reinforce (or undermine) each property and to what extent. The `O1` observation strongly undermines `Safety` when a collision with the robot is detected (`Bump` event); `O2` undermines `Safety` if the robot drives faster than allowed when there is a person in the robot working area; `O3` reinforces `Performance` every time the robot successfully completes a task within the required time; and `O4` and `O5` undermine `Performance` if the robot enters into the error state or if its current task is aborted, respectively. Note that the persistence and repetition of the observations were left by default.

```
param MAX_VELOCITY : number
param MAX_JOB_DONE : number

property Performance
property Safety

context Bump        : event
context Velocity    : number
context PersonState : boolean
context JobState    : enum { NOT_STARTED, STARTED, COMPLETED, ABORDED }
context RobotState  : enum { IDLE_NOT_CHARGING, IDLE_CHARGING,
    BUSY_DRIVING_WITH_LOAD, BUSY_DRIVING_EMPTY, ERROR
}
context TimeJobDone : time
    = interval(JobState::STARTED -> JobState::COMPLETED)
```

```
observation O1 : Bump
    undermines Safety very high

observation O2 : Velocity > MAX_VELOCITY and PersonState
    undermines Safety very high

observation O3 : JobState::COMPLETED while (TimeJobDone < MAX_JOB_DONE)
    reinforces Performance high

observation O4 : RobotState::ERROR
    undermines Performance

observation O5 : RobotState::ABORTED
    undermines Performance
```

Listing 20. Model for the intralogistics robotic application

7.1.1.3 Implementation

The component architecture of the robot was modelled using the SmartMDSD Tool-Chain [74]. This is an Eclipse-based IDE that provides support to the RobMoSys Ecosystem, covering the full cycle of the robotics software development. The RoQME project delivered a number of Eclipse plug-ins ready to be integrated into SmartMDSD. With them, we were able to specify how the contexts modelled in Listing 20 could be obtained from the services provided by the robot architecture (i.e., through a RoQME-to-RobMoSys mapping model). Based on the models, RoQME produced a series of transformations to generate the RobMoSys component in charge of estimating the QoS metrics. The resulting component was ready to be integrated into the robot architecture and included (1) a DDS-based Publish/Subscribe Data Space [64], (2) an Event Processor using Esper [70], (3) the QoS Metrics Estimator, (4) the context monitors, and (5) a QoS consumer for recording and displaying all the data resulting from the execution of the component: contexts, observations and QoS metrics.

7.1.1.4 Results

The experiment took place on November 8, 2018, at the Service Robotics Research Center of the Ulm University of Applied Sciences (Germany). The experiment started with a robot in an `IDLE` state and placed in its initial position. There was an operator ready to handover boxes to the robot.

1. The robot was ordered a new task (`JobState::STARTED`). It moved to the handover position (`RobotState::BUSY_DRIVING_EMPTY`). The operator put a box on top of it and the robot moved to the delivery position (`RobotState::BUSY_DRIVING_WITH_LOAD`). The robot delivered the box successfully (`JobState::FINISHED`) and within the required timeslot. Observation `O3` was fired, and `Performance` improved.
2. A visitor entered the room and met the operator (`PersonState::IN`). Simultaneously, the robot was ordered a new task (`JobState::STARTED`) and moved to the handover position (`RobotState::BUSY_DRIVING_EMPTY`). On its way, the robot moved faster than allowed when a person is in the robot working area. Observation `O2` was fired, and `Safety` worsened.
3. The visitor bumped into the robot (`Bump`) when turning around for leaving the room. Observation `O1` was fired, and `Safety` worsened.
4. The bumping set the robot in the `ERROR` state (`RobotState::ERROR`). Observation `O4` was fired, and `Performance` worsened.
5. The error caused the task to be aborted (`JobState::ABORTED`). Observation `O5` was fired, and `Performance` worsened.
6. The robot was ordered a new task (`JobState::STARTED`), which was successfully completed (`JobState::FINISHED`). Observation `O3` was fired, and `Performance` improved.

The evolution of the QoS metrics defined on `Safety` and `Performance` during the experiment is shown in Figure 34 and Figure 35. Further details about the experiment can be found in [75].

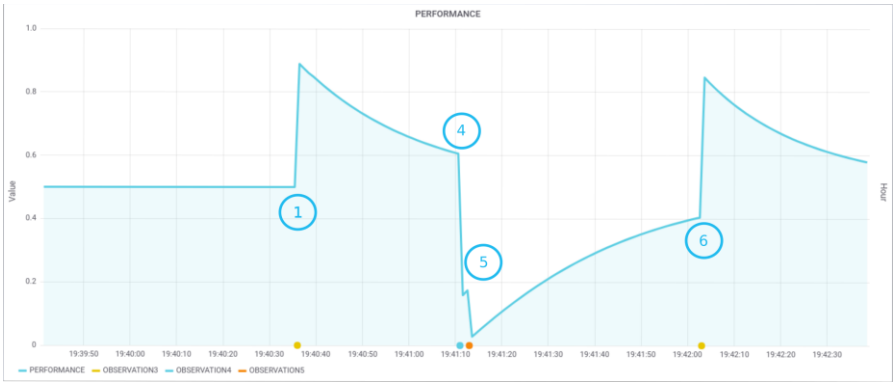


Figure 34. Evolution of the QoS Metrics defined on performance. The numbers in the graph refer to the steps described in the section. (Source: [75])

Figure 35. Evolution of the QoS Metrics defined on safety. The numbers in the graph refer to the steps described in the section. (Source: [75])

7.1.2 Geriatric assessment

7.1.2.1 Scenario description

This scenario was carried out in the context of the ECHORD++ CLARC Project [76]. This project developed a social robot aimed at helping clinicians perform Comprehensive Geriatric Assessment (CGA) procedures. The CGA is a multidimensional clinical procedure that includes questionaries, cognitive exercises, and motion analysis exercises. The CLARC robot autonomously drives some of these tests (interacting with the patients), saving time for the clinicians to perform more added-value activities. See the CLARC robot in Figure 36. In this scenario we only considered the Barthel test [77] and the "Get Up & Go" test [78].

The Barthel test is composed of 10 questions about daily life activities. It usually lasts about 10–15 min. CLARC can ask questions using natural interaction (via voice or on-screen text). For each question, the patient can choose two, three, or four possible answers by touching the option on the screen, by using a specially designed remote control (see image on the right in Figure 36), or by answering verbally. Patients have two chances to answer, if they do not, the robot continues with the next question.

In the "Get Up & Go" test, the patient is asked to get up from a chair, walk a short distance (about three meters) in a straight line, return to the chair, and sit down. The objective is to measure the balance, detecting deviations from a normal and confident performance. The test

has two main parts. In the first, the patient sits next to CLARC to receive the test instructions. In the second, the patient is asked to go to the position where the test will be performed, and the robot moves to a suitable location to observe the complete movement. The robot emits a signal to start the test and measures the patient's gait to analyse balance and timing.

Figure 36. The CLARC robot. (Left) CLARC performing a Barthel test on a patient. (Right) Remote control that patients can optionally use to interact with the robot. (Source: [79])

7.1.2.2 Modelling QoS metrics

Listing 21 shows the model that was developed for the scenario. It is worth noting that the observations were defined with the help of a physiotherapist and an expert in usability/accessibility of human-computer interfaces. Both were involved in the CLARC project from the beginning.

The model specifies QoS metrics on three non-functional properties: `Performance`, `Interaction` and `VoiceInteractionQuality`. We consider `Performance` as the degree to which the robot succeeds in completing a test. For example, the robot will show a good performance in the Barthel test if it gets the patient to answer all the questions on the first try. Conversely, a poor performance will be obtained if the robot is unable to end the session. Regarding the `Interaction` property, it expresses whether the patient is actively interacting with the robot. Note that this property can score high regardless of whether the robot succeeds in getting the patient to complete the test (which is captured by `Performance`). Finally, `VoiceInteractionQuality` evaluates the patient's preference for verbal interaction against

non-verbal (touch screen and remote control). This property was included to analyse the preferences of the patients for possible future improvements of the robot. It only concerns the Barthel test, since the "Get Up & Go" test does not have verbal interaction.

The context variables involved in the Barthel and "Get Up & Go" test are `TestState`, `PersonDetected` and `CallingDoctor`. First, `PersonDetected` indicates if robot has detected the patient, `TestState` shows the current state of the test, and `CallingDoctor` is triggered when the patient presses the calling doctor button. Regarding the contexts only related to the Barthel test, the system provides information on whether a patient answers a question on the first attempt (`FirstAttemptAnsewerd`), on the second attempt (`SecondAttemptAnswered`) or if the patient has not answered (`FirstAttemptAnswered` and `SecondAttemptAnswered` would be false). Finally, `UserInteraction` indicates the interaction mechanism that has been used to answer a question.

`Scored`, `PersonIsNextToChair`, `PersonIsSeated` and `PersonGetsUp` are contexts used in the "Get Up & Go" test. `Scored` states whether the evaluation of the last task returns a score, whereas `PersonGetsUp`, `PersonIsNextToChair` and `PersonIsSeated` are used to see if the patient follows the robot's instructions.

The model defines 14 observations of which the first 12 impact on `Performance` or `Interaction` in any of the tests, while the last two only affect `VoiceInteractionQuality` in the Barthel test. Note that the persistence and repetition of the observations were left by default.

```
property Performance
property Interaction
property VoiceInteractionQuality

context JobState : enum { RUNNING, PAUSED, RESTARTED, STOPPED, REPEATING, FINISHED }
context FirstAttemptAnswered : boolean
context SecondAttemptAnswered : boolean
context PersonDetected : boolean
context CallingDoctor : event
context UserInteraction : enum { TOUCH_SCREEN, VOICE_RECOGNITION, REMOTE_CONTROL }
context Scored : boolean
context PersonGetsUp : boolean
context PersonIsNextToChair : boolean
context PersonIsSeated : boolean

observation O1 : TestState::RUNNING or Scored or PersonIsNextToChair or
    PersonIsSeated or PersonGetsUp
    reinforces Performance high
```

```
observation O2 : FirstAttemptAnswered or TestState::FINISHED or PersonDetected
    reinforces Performance low

observation O3 : SecondAttemptAnswered or TestState::RESTARTED
    reinforces Performance very low

observation O4 : TestState::STOPPED or TestState::PAUSED
    undermines Performance very high

observation O5 : !Scored or !SecondAttemptAnswered or TestState::REPEATING or
    !PersonIsNextToChair or!PersonIsSeated or!PersonGetsUp or CallingDoctor
    undermines Performance high

observation O6 : !FirstAttemptAnswered or !PersonDetected
    undermines Performance low

observation O7 : PersonIsSeated or PersonIsNextToChair or FirstAttemptAnswered
    reinforces Interaction high

observation O8 : PersonDetected or PersonGetsUp
    reinforces Interaction low

observation O9 : CallingDoctor
    undermines Interaction very high

observation O10 : !PersonIsSeated or !SecondAttemptAnswered or !PersonIsNextToChair
    undermines Interaction high

observation O11 : !PersonGetsUp or !FirstAttemptAnswered or TestState::PAUSED or
    TestState::REPEATING or !PersonDetected
    undermines Interaction low

observation O12 : SecondAttemptAnswered or TestState::RESTARTED
    reinforces Interaction very low

observation O13 : UserInteraction::VOICE_RECOGNITION
    reinforces VoiceInteractionQuality high

observation O14 : UserInteraction::REMOTE_CONTROL or UserInteraction::TOUCH_SCREEN
    undermines VoiceInteractionQuality high
```

Listing 21. Model for the geriatric assessment scenario

7.1.2.3 Implementation

The software architecture of the robot is based on a collection of agents that interact and cooperate to achieve a global goal using a data structure called Deep State Representation (DSR) [80]. We were able to use the RoQME toolchain with the model in Listing 21 to generate part of the code for a new agent in this architecture aimed at estimating QoS metrics. It was part because CLARC does not use the RobMoSys component architecture, so it was necessary to implement some of the code manually, such as the context monitors or the glue code to

integrate the RoQME component into CLARC. In the end, the new agent included (1) a DDS-based Publish/Subscribe Data Space, (2) an Event Processor using Esper, (3) the QoS Metrics Estimator, (4) the context monitors, and (5) a QoS consumer for recording the resulting metrics during the execution of the scenario.

7.1.2.4 Results

The experiment was divided into two parts, a first one based on simulations and a second with real patients. In the first part of the experiment, a domain expert was in charge of designing the simulations with the aim of covering the most common cases. These simulations served to verify that the resulting metrics were sufficiently representative and, if necessary, to be able to adjust the model before tackling the second part.

Preliminary tests. Table 22 presents the cases simulated for the Barthel test and the resulting average of each QoS metric. Regarding `Performance`, as expected, the best outcome is achieved when the patient answers all the questions on the first attempt. `Performance` decreases as the patient begins to answer on the second attempt or does not answer some of the questions. In the worst case, the patient pauses the session several times, calls the doctor, and occasionally the robot is unable to detect him/her (see Test #6). In terms of `interaction`, the patient interacts more efficiently with the robot when it responds on the first try. On the other hand, as the robot has to repeat more questions, the `interaction` gets worse (e.g., see Test #3, where all the questions were repeated). The worst interaction appears when we simulate the typical actions of a patient who feels uncomfortable with the robot (see Test #6). Finally, `VoiceInteractionQuality` seems to reflect well the degree to which the patient interacts verbally with the robot.

Table 23 shows the simulations for the "Get Up & Go" test and the results. Note that, unlike the Barthel test, we recorded the minimum and the last value of each QoS metric. This is so because the duration of the "Get Up & Go" tests is variable and depends on the patients, which significantly alter the averages. On the other hand, it seems that the minimum value reflects well any problems during the test (regardless of its success), while the last value captures how the test ends.

Table 22. Preliminary Barthel tests. The results correspond to the average value of the QoS metrics

Test	Description	Performance	Interaction	VoiceIQ
#1	The patient answers all questions verbally on the first attempt	0.8188	0.9084	0.8870
#2	The patient answers all questions on the first attempt using the touch screen or the remote control	0.8187	0.9076	0.1139
#3	The patient answers all questions on the second attempt using the touch screen	0.3351	0.3349	0.1055
#4	The patient sometimes answers the questions on the first attempt, sometimes on the second attempt, and sometimes does not answer. The patient does it verbally or using the touch screen alike	0.3248	0.6107	0.6870
#5	Using the remote control or the touch screen, the patient sometimes answers the questions on the first try, on the second try, or does not answer. The patient calls the doctor and pauses the session twice	0.1671	0.4026	0.1454
#6	Same as the previous case but occasionally the robot loses the patient (patients who do not feel comfortable tend to move around)	0.1357	0.1893	0.2098

Table 23. Preliminary "Get Up & Go" tests. The results correspond to [min, last] of the QoS metrics

Test	Description	Performance	Interaction
#1	The patient performs the test successfully	[0.5000, 0.9987]	[0.5000, 0.9857]
#2	The robot loses the person for a few seconds when starting the test. After that, the test is performed successfully	[0.4982, 0.9987]	[0.4982, 0.9857]
#3	The robot loses the person for a few seconds when starting the test. On the first try, the person does not stay next to the chair. On the second, he does, and the test is successful	[0.2978, 0.9808]	[0.2978, 0.9567]
#4	The robot loses the person for a few seconds when starting the test. The person does not stay next to the chair. The test is stopped	[0.1550, 0.1555]	[0.0824, 0.0828]
#5	The patient stays next to the chair, but he sits on the chair at the second attempt (the robot must repeat the instructions). After that, the test is performed successfully	[0.5000, 0.9956]	[0.4879, 0.9361]
#6	The patient stays next to the chair, sits on the chair but he does not get up. The robot does not give him a score	[0.5000, 0.8310]	[0.5000, 0.7067]
#7	The patient stays next to the chair, sits on chair, gets up but does not sit on the chair again. The robot does not give him a score	[0.5000, 0.9343]	[0.5000, 0.8270]
#8	During the test introduction the patient presses the restart button. The test is performed successfully	[0.1242, 0.9904]	[0.4981, 0.9818]
#9	During the test introduction the patient presses the calling doctor button. The doctor presses the restart button. The test is performed successfully	[0.2984, 0.9963]	[0.1242, 0.9058]

Regarding `performance`, the cases in which the test ends successfully always obtain a high last value (greater than 0.98). If the patient makes progress but performs poorly, the robot may not be able to score the task, in which case, the last QoS value is lower (see Test #3 and #7). The worst situation appears when the test has to be stopped before concluding (Test #4). Note that a minimum value lower than 0.5 (the initial value of the metrics) indicates that the test execution was not perfect (e.g., the robot lost the patient for a moment, or the test was restarted).

In terms of `interaction`, according to the results, the patients interact more efficiently with the robot when they follow the instructions on the first try (see Test #1). The metric gets worse as the number of times the robot has to repeat the instructions increases.

Real tests. From January 2019 to March 2019, one CLARC robot was put into operation at the Reims Hospital (France) and two at the Hospital Virgen del Rocío in Seville (Spain). The experimental evaluation was based on data collected during this period from 22 patients for the Barthel test, and 13 patients for the "Get Up & Go" test.

Table 24 and Table 25 present the results for the Barthel and the "Get Up & Go" tests, respectively. We can see that some cases are similar to the ones that were designed for the preliminary tests. Moreover, the results are also in line with these preliminary tests. In general, the metrics seem to capture well the degree of performance and interaction that is expected based on the case description. For example, as in the simulations, the best performance for the Barthel test is obtained when the patients answer all the questions on the first attempt (see #1 in Table 24). Further details about the experiment can be found in [79]

Table 24. Real Barthel tests. The results correspond to the average value of the QoS metrics

Group	Description	Performance	Interaction	VoiceIQ
#1	5 patients answer all questions on the first try using the touch screen	0.8204	0.9094	0.1117
#2	1 patient answer all questions verbally on the first try	0.8182	0.9078	0.8862
#3	7 patients answer all questions on the first try using remote control	0.8191	0.9081	0.1133
#4	1 patient answers 9 questions verbally on the first try. 1 unanswered question	0.6800	0.8946	0.8803
#5	2 patients answer 9 questions on the first try using touch screen. 1 unanswered question	0.6428	0.8688	0.1252
#6	Using the touch screen, 2 patients answer 9 questions on the first try and 1 question on the second try	0.7433	0.8399	0.1386
#7	Using the remote control, 2 patients answer 9 questions on the first try and 1 question on the second try	0.7482	0.8514	0.1316
#8	Using the remote control, 1 patient answers 9 questions on the first try and 1 question on the second try. The patient paused the test and then resumed it.	0.5586	0.8879	0.1284
#9	1 patient answers 3 questions verbally on the first try. 7 unanswered questions	0.1602	0.2423	0.8126
#10	Using the remote control, 1 patient answers 5 questions on the first try and another 5 on the second. The patient paused the test and then resumed it. Occasionally the robot loses the patient	0.2647	0.7596	0.1278

Table 25. Real "Get Up & Go" tests. The results correspond to [min, last] of the QoS metrics

Group	Description	Performance	Interaction
#1	12 patients perform the test successfully	[0.5000, 0.9987]	[0.5000, 0.9858]
#2	1 patient stays next to the chair, sits on the chair but he does not get up. The robot does not give him a score	[0.5000, 0.8310]	[0.5000, 0.7067]

7.1.3 Discussion

The scenarios were developed in collaboration with researchers from the RoQME and the CLARC projects. See the list of authors in [79], [81], [82]. The main contributions of this Thesis to the scenarios were (1) the formal framework for the estimation of non-functional properties, (2) the design and implementation of the modelling language to express QoS metrics, and (3) the QoS Metric Estimator. The results shown in Section 7.1.1 and Section 7.1.2 were obtained when the experiments were carried out. Since then, and thanks to the experience gained in these experiments, the proposal has received minor changes and improvements that do not invalidate the results. Next, we will discuss some aspects of our proposal based on this experience.

7.1.3.1 On the effectiveness to estimate non-functional properties

In view of the results in both scenarios, QoS metrics seem to be able to express, at least to some extent, the fulfilment that a system shows with respect to one or more non-functional properties. For instance, in the intralogistics scenario, when the visitor bumped into the robot, the safety metric dropped drastically. Or, in the Barthel test, whenever a patient answers all the questions on the first try, the average of the performance metric shows a high score. Regardless of whether the visitor was reckless, or if answering all the questions was thanks to the good work of the patient (and not of the robot), these circumstances modify the perception of how the system is working, and QoS metrics provide a mechanism to quantify it.

In our proposal, QoS metrics not only react to context patterns, but can also exhibit certain dynamics. For example, the incident with the visitor will be forgotten over time, so the safety metric gradually recovers its initial value as time passes (see Figure 35).

According to our approach, to be effective in estimating non-functional properties, it is necessary (but not sufficient) that: (1) properties produce observable effects in the context, which (2) should be adequately expressed with the modelling language and (3) detected by the system. Thus, among the factors that could limit the estimation of properties we find that: (1) an observation could not only occur due to a property but there could be other hidden causes; (2) the modelling language might not be expressive enough to define certain observations; and (3) the system may not be able to monitor certain contextual information. However, while there may be limitations affecting the accuracy of the estimates, in the end, what matters is getting results that are good enough for our purposes. In this regard, we will discuss the applicability of the proposal in Section 7.1.3.3.

7.1.3.2 On the benefits of raising the level of abstraction

Figure 37 shows the probabilistic network that is derived from the model in Listing 20. This illustrates how the modelling language manages to abstract the details that require specific knowledge and present the information in a qualitative way with a simple vocabulary. Note that the observations of the geriatric assessment scenario were defined with the help of a physiotherapist and an expert in usability/accessibility of human-computer interfaces. This collaboration would have been almost impossible if they had worked directly in terms of probabilities. In addition, thanks to handling fewer details, it is reasonable to think that the communication became more efficient, and the task could be completed in less time.

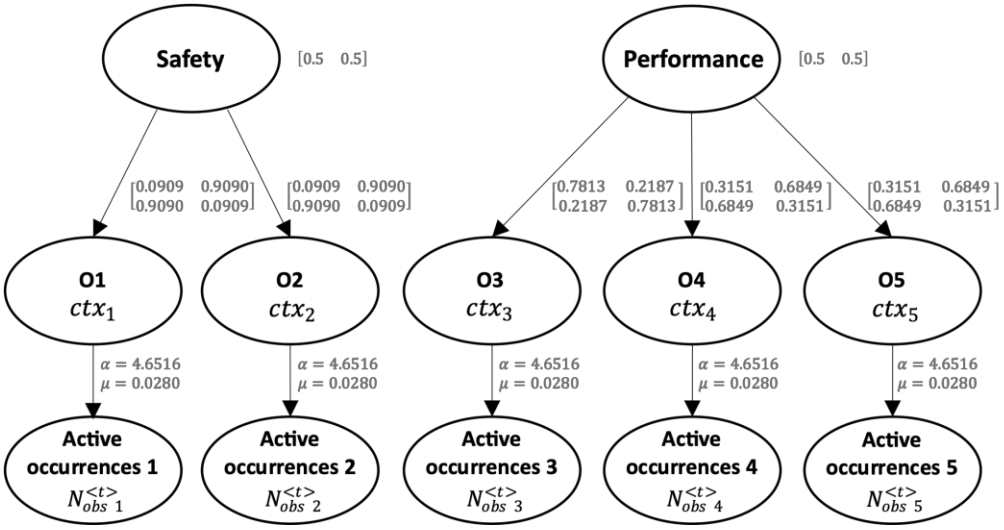


Figure 37. Probabilistic network for the intralogistics scenario

The trade-off of abstraction is the consequent loss of flexibility, since users must stick to what is provided in the modelling language. This is related to the limitation that we discussed in the previous section, in which we imagined that there could be observations unable to be expressed with the modelling language. Fortunately, neural networks can mitigate this limitation since they offer us the possibility of tuning a model based on examples, as we saw in Section 5.3.

7.1.3.3 On the applicability of the proposal

The application of the proposal in two real environments has allowed us to demonstrate its feasibility. In this sense, the proposal did not require the incorporation of new hardware to the robots, and we did not observe any deterioration of the system performance due to the processing of QoS metrics.

The proposal was put into practice in two different application domains: intralogistics and geriatric assessment. The modelling language was suitable for both, which confirms that it is independent of the application domain. Moreover, the proposal was deployed on two different robotic platforms with little effort. In the case of the intralogistics application, it was possible to generate the complete code of a component for enabling QoS estimation, ready to be added to the software architecture. While in the case of the clinic assistant robot, it was only possible to generate part of the code and the manual implementation of glue code and context monitors was required. In this vein, the distributed approach of the proposal, thanks to the publish/subscribe data space, was useful.

Regarding the possible applications, QoS metrics could be used to compare different realizations of a system in terms of non-functional properties, such as safety or interaction. For instance, considering the clinic assistant robot, we could try two different ways to engage patients in the tests, and QoS metrics would give us the means to assess which is best. In this case, since we would be comparing two alternatives in the same conditions, it may not be so important to have totally accurate metrics. For example, if there is a bias, it would be in both alternatives and would become irrelevant when comparing. As we mentioned in Section 7.1.3.1, QoS metrics only need to be good enough for our purposes.

QoS metrics could also be used to check non-functional requirements at runtime. For example, let us imagine that an intralogistics robot, by design, must always be safe. The robot would monitor the corresponding QoS metric and, if the estimate fell below a threshold, the robot would execute a contingency plan: send an alert to the maintenance team and return to its docking station.

Ultimately, what is sought is QoS-aware systems capable of adapting their behaviours when necessary. For instance, a clinic assistant robot that selects the way to engage a patient in the tests based on previous experience, the person's mood, preferences, etc. in order to improve QoS metrics, such as performance or interaction.

7.2 Characterization of QoS metrics

This section presents some simulations to examine the characteristics that QoS metrics can exhibit under certain conditions.

7.2.1 The persistence of the observations

Observations gradually lose their significance as time passes. As soon as an observation occurred its influence is maximum, but as time passes it weakens until it eventually disappears. Our proposal allows us to set a persistence time to each observation. The following experiments aim to explore the behaviour of QoS metrics under different persistence configurations.

7.2.1.1 Experiment 1

We have generated a sequence of 10 occurrences equidistant in time. These occurrences correspond to observations that reinforces a property with random strength. All the observations have a persistence of 20 minutes. After running several simulations of the same sequence varying the time between occurrences, Figure 38 shows the average of the resulting QoS estimates.

Since observations weaken as time passes, the evidence accumulated over a period will be potentially more intense the more concentrated the occurrences are in time. Let us use an example to show that this effect not only makes sense but is also practical. Suppose that we have an intralogistics robot like the one in Section 7.1.1. Every time the robot completes a job, an observation is triggered to reinforce performance. With that, if the robot had completed the jobs, it does in a day, in half the time, the QoS metric would have been higher. Thus, Figure 38 tells us that if the sequence of 10 occurrences were jobs done, completing them in 50s intervals would result in an average estimate of 0.89, while if we took 365s to complete each job the estimate would drop to 0.67.

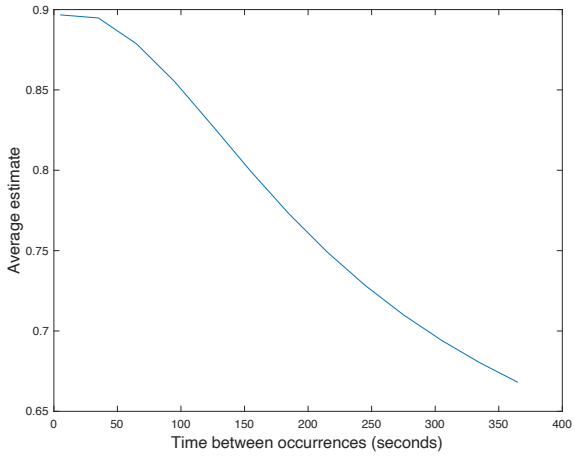


Figure 38. Average value of the metric for different times between occurrences

7.2.1.2 Experiment 2

The time between occurrences and the persistence of an observation are often related. In the sense that rare events usually have a long persistence. For example, a visitor bumping into the robot in a warehouse is something that will remain in everyone's memory for a long time. In this experiment we will go against logic, and we will simulate frequent observations with long persistence.

We have simulated the effect of 400 random occurrences evenly distributed over 6 hours. The occurrences were produced from 4 random observations, being the average time between occurrences of 3min 36s. In addition, all the observations were set with a persistence of 6 hours.

Figure 39 shows the first 40 minutes of two simulations. At first, we can see that the QoS metric reacts to the occurrences (marked with dotted lines). However, shortly after starting, the metric reaches a value that remains constant regardless of the occurrences. This effect appears when there are enough occurrences to support the maximum level of evidence that the observations can provide. The situation is maintained because the evidence does not have time to dissipate before the arrival of a new occurrence.

Regarding the value at which the metric remains constant, it will depend on the balance of the observations, whether the mass of evidence that reinforces is greater, less or equal to that which undermines. For example, the upper graph resulted from equally strong reinforcing and undermining observations (so that the metric tends to 0.5), while in the lower graph the observations were more reinforcing (so that the metric tends to 0.63). In conclusion, a long persistence in frequent observations eliminates the dynamic behaviour of the metrics. The QoS estimation becomes a static process where what matters is the weight of the observations.

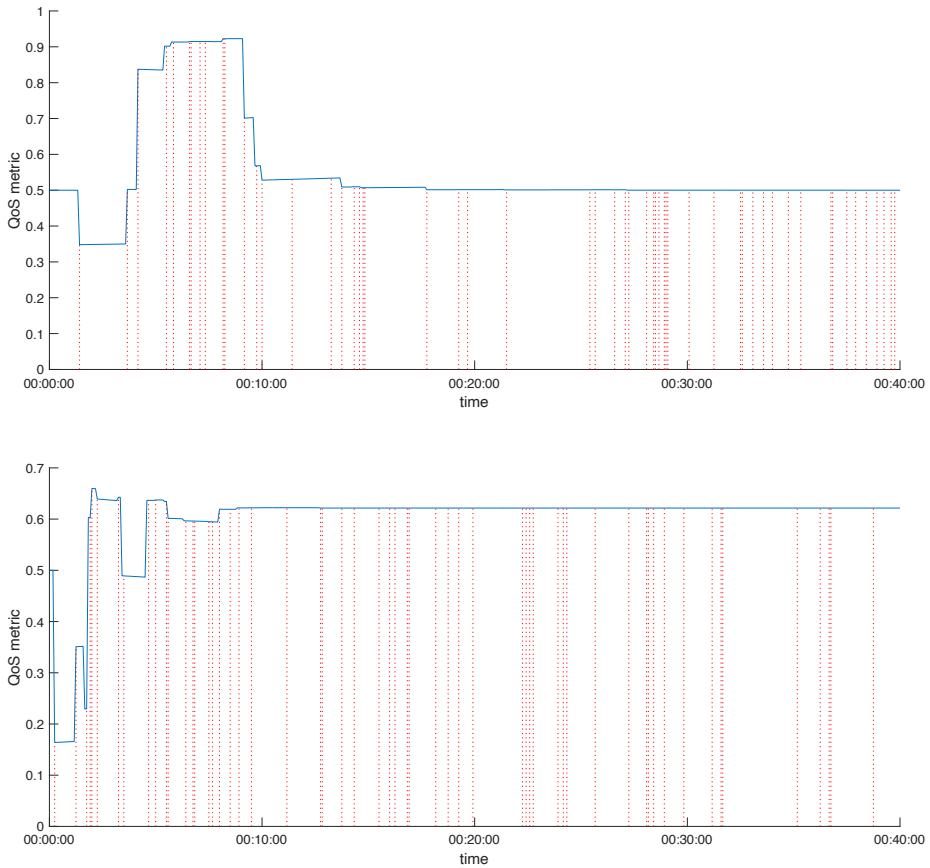


Figure 39. Simulations of a QoS metric from frequent occurrences with long persistence

7.2.1.3 Experiment 3

In this experiment we are going to do the opposite of what we did in the previous experiment. We will simulate sparse observations with short persistence. For that, we have generated 20 random occurrences evenly distributed over 6 hours. The occurrences were produced from 4 random observations. The average time between occurrences is 72 minutes while their persistence was set to 10 seconds. Figure 40 shows the resulting QoS metric. Although we can see sporadic spikes every time there is an occurrence, the impact on the metric is minimal. Proof of this is the temporal mean of the metric, which only varies between 0.4869 and 0.5163 (using a 2-minute window). The observations are so ephemeral that they cannot build solid evidence.

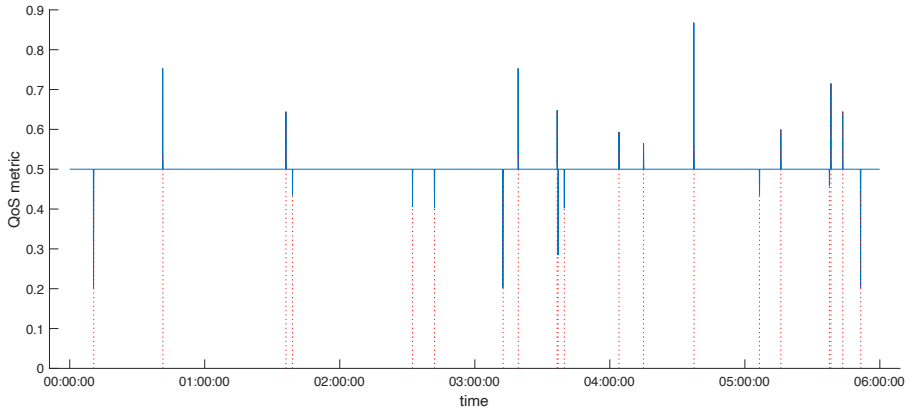


Figure 40. QoS metric from sparse occurrences with short persistence

7.2.2 The effect of repeating observations

Apart from the persistence, *repetition* is another attribute to establish the dynamic behaviour of the observations. This attribute indicates how many times an observation must be repeated to have full effect. Thus, the first time an observation is detected it may not have a great impact (it could be an error), but as the observation is repeated it would gain in reliability as well as in intensity. In the following experiments, we will explore this feature.

7.2.2.1 Experiment 4

In this experiment we are going to visualize the effect of the repetition attribute on a QoS metric. For that, we have executed 5 simulations with different repetition values, from 0 to 60. We have considered 100 occurrences from an observation that reinforces very highly with long persistence (10 hours). These occurrences were regularly distributed every 20 seconds. In total, the duration of the simulation was 33min 20s. As expected, we can see in Figure 41 how as repetition is greater, the observation takes longer to reach its maximum effect.

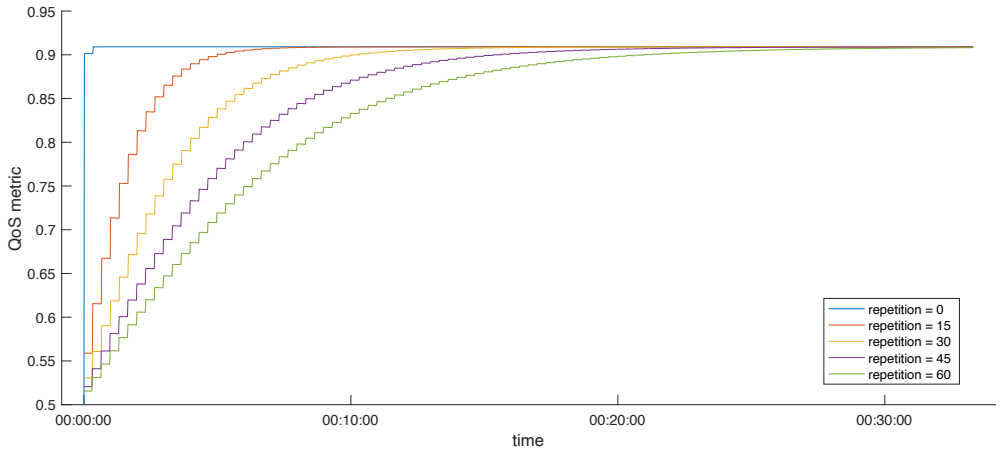


Figure 41. QoS metric with different repetitions

7.2.2.2 Experiment 5

In the experiment in Section 7.2.1.2, we saw that the metric remains constant when the observations get enough occurrences to support the maximum level of evidence. One way to delay this effect would be to increase the number of occurrences that the observation must reach to have full effect. That is, increasing the value of repetition.

We have generated 400 random occurrences evenly distributed over 6 hours. The occurrences were produced from 4 random observations, with persistence set to 6 hours. We have run two simulations with different repetition values: 12 and 40. Figure 42 shows the results. In the upper graph (with repetition equal to 12), we can see a similar tendency to the one we showed in Experiment 2. The QoS metric now takes almost 2 hours to reach a stable state, much longer compared to the 8-9 minutes it took in Experiment 2, where repetition was set to 1.

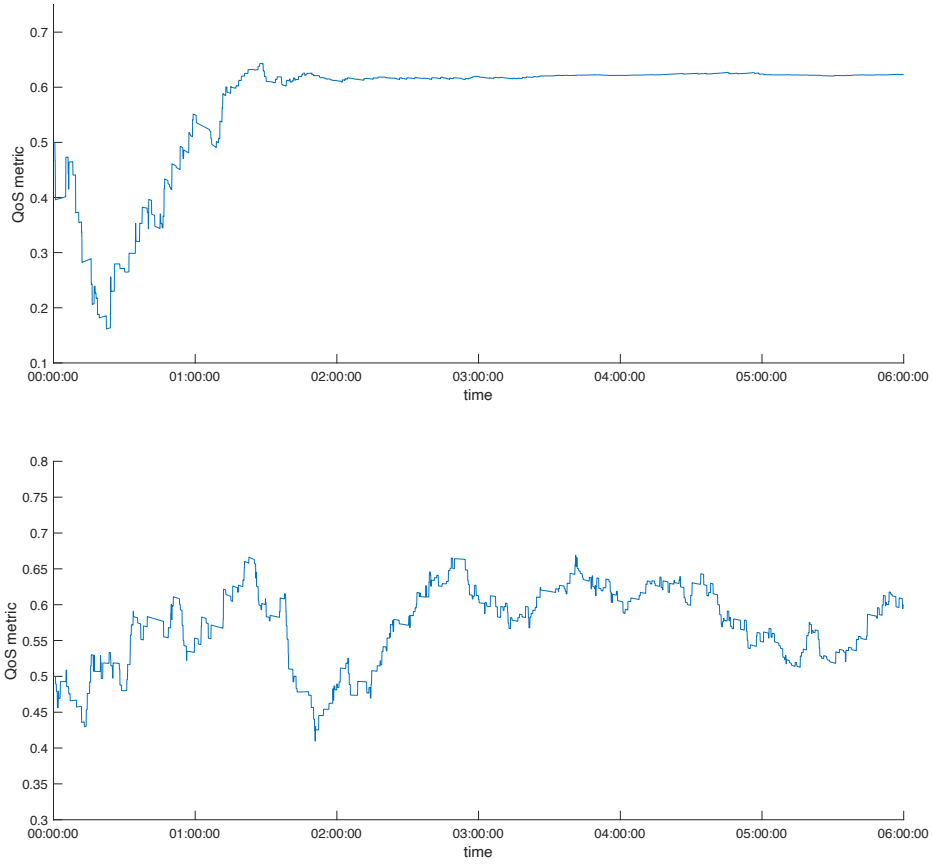


Figure 42. QoS metric with repetition set to 12 (upper) and 40 (lower)

As for the lower graph in Figure 42 (with repetition equal to 40), it does not tend to a constant value. To help us understand this result, Figure 43 shows the average number of active occurrences for each observation (defined as $N_{obs\,i}^{<t>}$ in Section 3.4). The graph expresses the number of past occurrences that persist, which is like the mass of evidence supporting an observation. At the beginning of the simulation, this evidence grows until it stabilizes around 23 approximately. If repetition is much lower than this value, the observation will reach and maintain its maximum effect over time, resulting in a constant estimate for the QoS metric (the case when repetition is 12). On the other hand, if we choose a value for repetition greater than the highest value shown in the graph, the observations will never achieve their maximum effect and the estimate will continue to vary in relation to the available evidence (the case when repetition is 40).

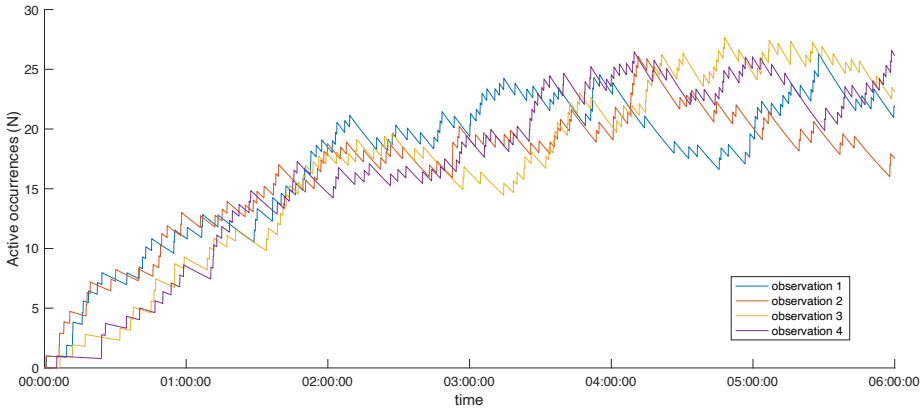


Figure 43. The average number of active occurrences

At the beginning of Section 5.2.2, we discussed some aspects of $N_{obs\ i}^{<t>}$ that could be of interest when specifying QoS metrics. Among them, we commented that those observations whose $N_{obs\ i}^{<t>}$ never reaches the number of repetitions + 1 could be underestimated. To investigate the effects of selecting oversized values for repetition, we have performed several simulations considering 1600 random occurrences evenly distributed over 24 hours. As in the previous simulations, the occurrences were produced from 4 random observations, with persistence set to 6 hours. Note that the time between occurrences is also maintained. Table 26 shows that as repetition increases, the values of the metric narrow and the mean tends to 0.5. That is, the observations lose influence and become more and more underestimated.

Table 26. Statistics for different values of repetition

Repetition	Mean	[min, max]	Range
40	0.5947	[0.4690, 0.7544]	0.2854
60	0.5730	[0.4511, 0.7279]	0.2768
80	0.5580	[0.4534, 0.6926]	0.2348
100	0.5477	[0.4585, 0.6633]	0.2048
150	0.5327	[0.4694, 0.6156]	0.1462
200	0.5248	[0.4762, 0.5886]	0.1124

7.2.3 The impact of the observations

In the previous experiments, we analysed how observations behave over time by setting the persistence and the number of repetitions. Now, we are going to study the impact that an observation has just when it occurs.

7.2.3.1 Experiment 6

Figure 44 shows the value of a QoS metric when an observation configured with a certain strength and direction has just occurred. In each simulation, we considered a single isolated occurrence and 0.5 as the metric value before the occurrence. Regarding the results, in Section 4.4.2, we assigned a uniform likelihood ratio for each strength configuration an observation might have. This decision seems to be reflected in the almost linear relationship shown in the figure.

The impact of an observation not only depends on its strength and direction, but repetition also plays a role in this regard, as we saw in Experiment 4. Figure 45 shows the influence that repetition has on the impact of an observation. Unsurprisingly, the higher the value for repetition, the more the impact is reduced. Finally, although repetition should only take non-negative integers, the formulation we developed in Section 4.4.4.2 allows real numbers equal or greater than 0.

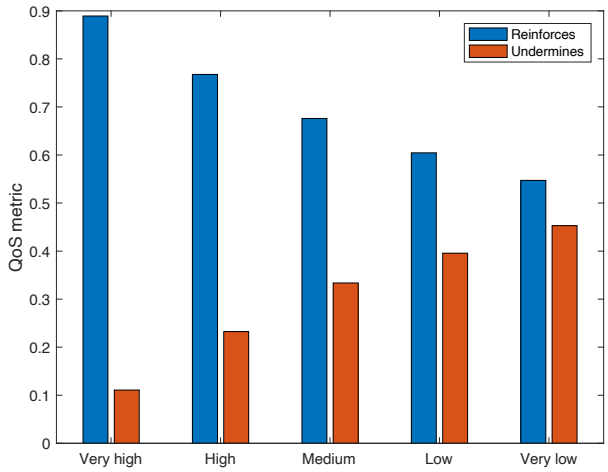


Figure 44. QoS metric when an observation has just occurred

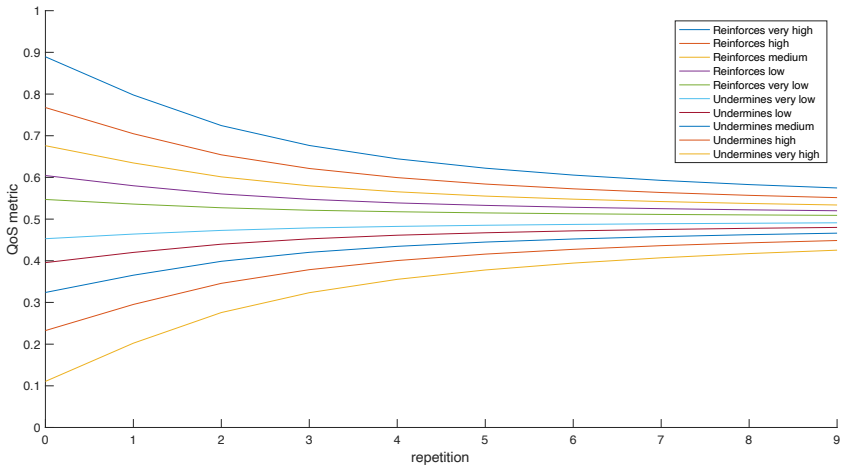


Figure 45. Impact of an observation for different repetitions

7.2.3.2 Experiment 7

In this experiment, we explore the effect that sharing evidence among several properties has on an observation. Figure 46 shows the impact of an observation when it provides evidence for up to 7 properties. Note that the number of properties has been represented as a continuous value for simplicity (although it is a positive integer). In addition, to carry out each simulation, we considered a single isolated occurrence and 0.5 as the metric value before the occurrence. Regarding the results, when sharing evidence among several properties, the uncertainty increases as the number of possibilities also increases, i.e., the occurrence may be the effect of one of the properties, of two, of a combination of them. It is not known. Therefore, due to this greater uncertainty, the evidence loses strength, which means observations with less impact.

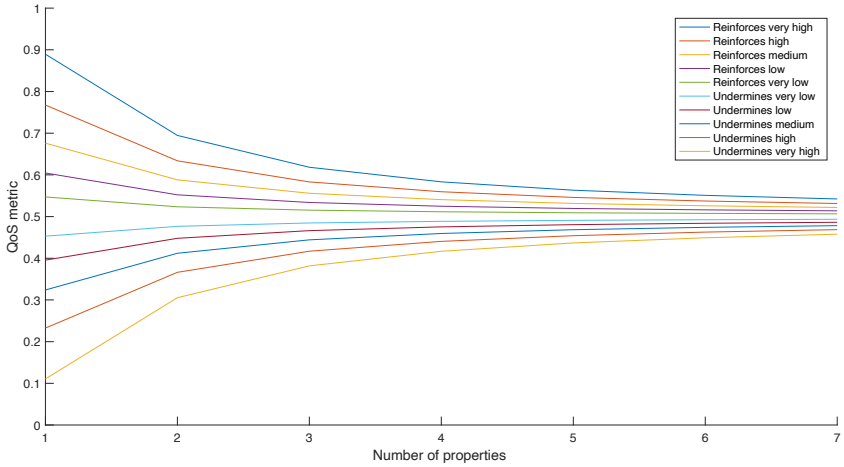


Figure 46. Impact of a shared observation for up to 7 properties

7.2.4 Discussion

In our proposal, the behaviour of an observation can be defined through 4 basic attributes: persistence, repetition, strength, and direction. While the first two establish the dynamic response of the observation, the last two focus on the impact that the evidence has on the properties. In the previous sections, we have examined, with 7 experiments, the effects of different attribute configurations. Below, we highlight the following conclusions:

- Since observations weaken as time passes, the evidence accumulated over a period will be potentially more intense the more concentrated the occurrences are in time. Therefore, in general terms, a system that completes the same jobs faster than another will obtain higher estimates for performance, or two incidents in a matter of days will be perceived worse than if months had passed between them. This kind of implication is intrinsic in the behaviour of QoS metrics and can be adjusted thanks to the persistence attribute.
- Rare events usually have long persistence. However, to study the effects that we could obtained with different configurations of persistence, we tested the opposite: frequent observations with long persistence. The result was that the metric tends to a constant value after accumulating so many occurrences that saturate the observations. The value at which the metric remains constant will depend on the balance of the observations.
- Continuing with the study of the effects that we could obtained with different configurations of persistence. We have simulated sparse observations with short persistence. The resulting observations had little impact on the mean of the QoS estimates. These were so ephemeral that they could not build solid evidence.
- The attribute repetition indicates how many times an observation must be repeated to have full effect. Consequently, as the repetition value is higher, the observation takes longer to reach its maximum effect.
- We have analysed the role of the attribute repetition in the case of frequent observations with long persistence. We studied the repetition value based on the average number of active occurrences ($N_{obs\,i}^{<t>}$). If repetition is much lower than this values that $N_{obs\,i}^{<t>}$ shows, the observation will reach and maintain its maximum effect over time, resulting in a constant estimate for the QoS metric. Observations become saturated. On the other

hand, if repetition is higher than any $N_{obs\,i}^{<t>}$, the observations may not achieve their maximum effect and the estimate will move in relation to the available evidence. However, as repetition increases further, the values of the metric narrow and the mean tends to 0.5. That is, the observations lose influence and become increasingly underestimated.

- We have studied the impact that an observation has just when it occurs. The impact of an observation not only depends on its strength and direction, but repetition also plays a role in this regard. In this sense, the higher the repetition value, the more the impact is reduced.
- When an observation provides evidence among several properties, the uncertainty increases and, consequently, the evidence loses strength, which means observations with less impact.

Chapter 8 Conclusions and future work

This chapter draws the conclusions of the Thesis and outlines some potential research directions to extend it in the future. Research papers published in different journals and conferences in relation to this Thesis are also listed, as well as some previous publications in which the foundations of this work were laid.

8.1 Conclusions

The research presented in this Thesis addresses the dynamic estimation of non-functional properties using Quality of Service (QoS) metrics. In this work, QoS metrics are defined as subjective and coarse-grained scores, indicating how well the system performs in terms of non-functional properties, such as safety, resource consumption or user satisfaction, among others.

A formal approach for deriving QoS estimates from contextual observations using probabilistic networks has been presented. These networks are created by making explicit the causal relationship between QoS perception and context. Unfortunately, the use of this method requires domain experts to have the ability to make the probabilistic model fit each application, which is challenging and requires specialized knowledge. To mitigate this problem, a Domain-Specific Modelling Language (DSML) has been defined, which hides the complexity of dealing with probabilistic networks. This DSML allows users to model QoS metrics in simple terms, and then derive the underlying probabilistic network transparently.

The classical formulation of the proposed probabilistic networks is not practical from a computer-based implementation perspective. It makes extensive use of multiplications and, when operands are small numbers (probabilities), numerical stability and precision can become

an issue. Thus, an alternative formulation of the proposed probabilistic networks to perform exact inference in a more robust way has been presented, as another of the main contributions of this Thesis. The implementation of this new formulation provides QoS Engineers with some statistics, which may help them improve their specifications.

Apart from introducing a novel formulation for probabilistic networks, an alternative approach, based on neural networks, is also proposed. These neural networks can be initialized from the parameters of the probabilistic networks, ensuring that they produce the same estimations. Although both methods are equivalent, neural networks have an important advantage thanks to the use of backpropagation, which provides domain experts with a straightforward method to refine QoS estimates, using a limited number of input-output examples.

The DSML developed to hide the complexity of dealing with probabilistic networks has been supported with a textual model editor, implemented using the Eclipse Modelling Framework and Xtext. It is worth noting that the QoS specifications created with the proposed DSML are software architecture- and domain application-independent. From the QoS specifications, developed with the textual model editor, domain experts can automatically generate the software components required to estimate QoS metrics at runtime. These components are loosely connected to each other through a publish/subscribe middleware, offering high flexibility and compatibility to be deployed on different platforms.

The proposal has been evaluated in two real-world scenarios with robots, one related to the transportation of goods in an intralogistics environment, and the other one using a clinic assistant robot for geriatric assessment. The abstraction provided by the proposed DSML, allowed a physiotherapist and an accessibility/usability expert to define the QoS specifications for the geriatric assessment scenario. Besides, the QoS metrics defined for both scenarios were able to correctly estimate how well both systems performed with respect to several non-functional properties. Finally, some additional experiments were carried out to observe the behaviour of the QoS metrics under some specific conditions. In all cases, the dynamic response of the observations and the influence of the evidence on the selected non-functional properties was as expected.

8.2 Future work

The findings of this Thesis open several lines for future research. Next, we outline some of the areas that can be further explored:

1. **Aggregation of QoS metrics**. Sometimes, it is desirable to combine QoS metrics into a single score to reflect the overall (aggregate) ability of a system to meet several quality requirements. The aggregation of QoS metrics may help reflect specific user specifications, including logic conditions, such as simultaneity and substitutability, or the relative importance of each non-functional property. Based on the work in Continuous Preference Logic (CPL) [83] and Generalized conjunction/disjunction (GDC) [84], further investigations could be conducted to build hierarchies of QoS metrics.
2. **Quality monitoring networks**. Service ecosystems are rapidly emerging to tame the complexity derived from the increasing number of entities providing and using services. In these ecosystems, the ability to derive global quality information seems an essential feature, and QoS metrics could be an important asset for monitoring, fault detection, or decision-making processes. Unfortunately, quality assessment becomes more and more challenging as ecosystems grow in complexity. The advancement of quality monitoring networks could facilitate the creation of more comprehensive quality metrics using heterogeneous and distributed contextual data.
3. **QoS metrics in reinforcement learning**. One of the fundamental problems of reinforcement learning is the so-called cursing of the objective specification [85]. Rewards are an essential part of any reinforcement learning problem, as they implicitly determine the desired behaviour. However, the specification of a good reward function can be highly complex. QoS metrics could help in the specification of these problems.

These and other related research lines show the great potential to continue exploring the development and application of QoS metrics on non-functional properties.

8.3 Publications

8.3.1 Publications related to the Thesis

The publications related to this Thesis are listed below in chronological order:

- A. Romero-Garcés, R. Salles De Freitas, R. Marfil, C. Vicente-Chicote, J. Martínez, J. F. Inglés-Romero, and A. Bandera, "QoS metrics-in-the-loop for endowing runtime self-adaptation to robotic software architectures," Multimed Tools Appl, vol. 81, no. 3, pp. 3603–3628, Jan. 2022, doi: 10.1007/s11042-021-11603-7.
- J. F. Ingles-Romero, R. Salles de Freitas, A. Romero-Garces, A. Bandera, J. Martinez, J.R. Lozano-Pinilla, D. Garcia-Pérez, and C. Vicente-Chicote, "The MIRoN Project — Endowing robots with context-awareness and self-adaptation capabilities," in JISBD2021, 2021. [Online]. Available: http://hdl.handle.net/11705/JISBD/2021/049
- R. Salles De Freitas, A. Romero-Garcés, R. Marfil, C. Vicente-Chicote, Jesús Martínez-Cruz, J. F. Inglés-Romero, and A. Bandera "QoS Metrics-in-the-Loop for Better Robot Navigation," in Advances in Physical Agents II, 2021, pp. 94–108.
- A. Romero-Garcés, J. Martínez-Cruz, J. F. Inglés-Romero, C. Vicente-Chicote, R. Marfil, and A. Bandera, "Measuring Quality of Service in a Robotized Comprehensive Geriatric Assessment Scenario," Applied Sciences, vol. 10, no. 18, 6618, 2020, doi: 10.3390/app10186618.
- C. Vicente-Chicote, D. García-Pérez, P. García-Ojeda, J. F. Inglés-Romero, A. Romero-Garcés, and J. Martínez, "Modeling and Estimation of Non-functional Properties: Leveraging the Power of QoS Metrics," in From Bioinspired Systems and Biomedical Applications to Machine Learning, 2019, pp. 380–388.
- J. F. Inglés-Romero, J. M. Espín, R. Jiménez, R. Font, and C. Vicente-Chicote, "Towards the use of quality of service metrics in reinforcement learning: A robotics example," CEUR Workshop Proc, vol. 2245, pp. 465–474, 2018.
- C. Vicente-Chicote, J.F. Inglés-Romero, J. Martínez, D. Stampfer, A. Lotz, M. Lutz, C. Schlegel. "A Component-Based and Model-Driven Approach to Deal with Non-Functional Properties through Global QoS Metrics," Proc. 5th International Workshop

on Interplay of Model-Driven and Component-Based Software Engineering (ModComp'18), in conjunction with MODELS 2018. Copenhagen (Denmark), 14-19 October, 2018.

- J. M. Espín, R. Font, J. F. Inglés-Romero, and C. Vicente-Chicote, "Towards the Application of Global Quality-of-Service Metrics in Biometric Systems," in IberSPEECH 2018, 2018, pp. 159–160.
- C. Vicente-Chicote, J. Berrocal, J. García-Alonso, J. Hernández, A. J. Bandera, J. Martínez, A. Romero-Garcés, R. Font, and J. F. Inglés-Romero, "RoQME: Dealing with non-functional properties through global robot QoS metrics," Actas de las 23rd Jornadas de Ingenieria del Software y Bases de Datos, JISBD 2018, 2018.

8.3.2 Previous publications on self-adaptive software

A selection of publications prior to the work of this Thesis are listed below in chronological order. Note that the ideas presented in this Thesis are largely influenced by prior research on adaptive software systems. The following publications serve as a representative sample of this work.

- M. Lutz, J. F. Inglés-Romero, D. Stampfer, A. Lotz, C. Vicente-Chicote, and C. Schlegel, "Managing Variability as a Means to Promote Composability: A Robotics Perspective," in New Perspectives on Information Systems Modeling and Design, A. M. Rosado da Cruz and M. E. Ferreira da Cruz, Eds. Hershey, PA, USA: IGI Global, 2019, pp. 274–295. doi: 10.4018/978-1-5225-7271-8.ch012.
- J. Ingles-Romero, A. Romero-Garces, C. Vicente-Chicote, and J. Martinez, "A Model-Driven Approach to Enable Adaptive QoS in DDS-Based Middleware," IEEE Trans Emerg Top Comput Intell, p. 1, 2017, doi: 10.1109/tetci.2017.2669187.
- R. Sanchez-Iborra, J. F. Ingles-Romero, G. Domenech-Asensi, J. L. Moreno-Cegarra, and M.-D. Cano, "Proactive Intelligent System for Optimizing Traffic Signaling," in 2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology

Congress(DASC/PiCom/DataCom/CyberSciTech), 2016, pp. 544–551. doi: 10.1109/DASC-PICom-DataCom-CyberSciTec.2016.104.

- C. Schlegel, A. Lotz, M. Lutz, D. Stampfer, J. F. Inglés-Romero, and C. Vicente-Chicote, "Model-driven software systems engineering in robotics: Covering the complete life-cycle of a robot," it - Information Technology, vol. 57, no. 2, Jan. 2015, doi: 10.1515/itit-2014-1069.
- A. Lotz, J. F. Inglés-Romero, D. Stampfer, M. Lutz, C. Vicente-Chicote, and C. Schlegel, "Towards a Stepwise Variability Management Process for Complex Systems," International Journal of Information System Modeling and Design, vol. 5, no. 3, pp. 55–74, 2014, doi: 10.4018/ijismd.2014070103.
- J. F. Inglés-Romero, R. Morales-Chaparro, C. Vicente-Chicote, and F. Sánchez-Figueroa, "A Model-Based Approach to Develop Self-Adaptive Data Visualizations," in Information System Development, Springer Nature, 2014, pp. 345–357. doi: 10.1007/978-3-319-07215-9_28.
- A. Romero-Garcés, J. F. Inglés-Romero, J. Martínez, and C. Vicente-Chicote, "Self-Adaptive Quality-of-Service in Distributed Middleware for Robotics," in 2nd International Workshop on Recognition and Action for Scene Understanding (REACTS 2013), 2013.
- A. Lotz, J. F. Inglés-Romero, C. Vicente-Chicote, and C. Schlegel, "Managing run-time variability in robotics software by modeling functional and non-functional behavior," Lecture Notes in Business Information Processing, vol. 147 LNBIP, pp. 441–455, 2013.
- J. F. Inglés-Romero and C. Vicente-Chicote, "Towards a formal approach for prototyping and verifying self-adaptive systems," Lecture Notes in Business Information Processing, vol. 148 LNBIP, pp. 432–446, 2013.
- J. F. Inglés-Romero, A. Lotz, C. V. Chicote, and C. Schlegel, "Dealing with Run-Time Variability in Service Robotics: Towards a DSL for Non-Functional Properties." arXiv, 2013. doi: 10.48550/ARXIV.1303.4296.
- J. F. Inglés-Romero, C. Vicente-Chicote, B. Morin, and O. Barais, "Towards the automatic generation of self-adaptive robotics software: An experience report," Proceedings of the 2011 20th IEEE International Workshops on Enabling

Technologies: Infrastructure for Collaborative Enterprises, WETICE 2011, pp. 79–86, 2011.

- J. F. Ingles-Romero, C. Vicente-Chicote, B. Morin, and O. Barais, “Using Models@Runtime for Designing Adaptive Robotics Software: an Experience Report,” 2010.

References

[1] R. R. Murphy and D. Schreckenghost, “Survey of metrics for human-robot interaction,” in *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2013, pp. 197–198. doi: 10.1109/HRI.2013.6483569.

[2] S. M. Anzalone, S. Boucenna, S. Ivaldi, and M. Chetouani, “Evaluating the Engagement with Social Robots,” *Int J Soc Robot*, vol. 7, no. 4, pp. 465–478, 2015, doi: 10.1007/s12369-015-0298-7.

[3] P. Damacharla, A. Y. Javaid, J. J. Gallimore, and V. K. Devabhaktuni, “Common Metrics to Benchmark Human-Machine Teams (HMT): A Review,” *IEEE Access*, vol. 6, pp. 38637–38655, 2018, doi: 10.1109/ACCESS.2018.2853560.

[4] M. Ma, J. A. Stankovic, and L. Feng, “Runtime Monitoring of Safety and Performance Requirements in Smart Cities,” in *Proceedings of the 1st ACM Workshop on the Internet of Safe Things*, 2017, pp. 44–50. doi: 10.1145/3137003.3137005.

[5] International Telecommunication Union (ITU), “ITU-T. Quality of telecommunication services: Concepts, models, objectives and dependability planning–Terms and definitions related to the quality of telecommunication services,” in *SERIES E: Overall Network Operation, Telephone Service, Service Operation and Human Factors*, 2008.

[6] S. Singh and I. Chana, “QoS-Aware Autonomic Resource Management in Cloud Computing,” *ACM Comput Surv*, vol. 48, no. 3, pp. 1–46, Feb. 2016, doi: 10.1145/2843889.

[7] L. Garcia, J. Lloret, C. Turro, and M. Taha, “QoE assesment of MPEG-DASH in polimedia e-learning system,” in *2016 International Conference on Advances in*

Computing, Communications and Informatics (ICACCI), Sep. 2016, pp. 1117–1123. doi: 10.1109/ICACCI.2016.7732194.

[8] Z. Xu, J. Lin, W. She, J. Xu, Z. Xiong, and H. Cai, "Neighbor Collaboration-Based Secure Federated QoS Prediction for Smart Home Services," 2022, pp. 71–85. doi: 10.1007/978-3-031-23515-3_6.

[9] Q. Dai, "A Survey of Quality of Experience," 2011, pp. 146–156. doi: 10.1007/978-3-642-23541-2_16.

[10] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach, Third Edition*, 3rd ed. USA: CRC Press, Inc., 2014.

[11] M. K. Daskalantonakis, "A practical view of software measurement and implementation experiences within Motorola," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 998–1010, 1992, doi: 10.1109/32.177369.

[12] R. B. Vaughn, R. Henning, and A. Siraj, "Information assurance measures and metrics - state of practice and proposed taxonomy," in *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, 2003, p. 10 pp. doi: 10.1109/HICSS.2003.1174904.

[13] ISO/IEC, "ISO/IEC 9126-1:2001 Software engineering - Product quality," 2001. https://www.iso.org/standard/22749.html (accessed Jan. 29, 2022).

[14] ISO/IEC, "ISO/IEC 25010:2011 - Systems and software Quality Requirements and Evaluation (SQuaRE)," 2011. https://www.iso.org/standard/35733.html (accessed Apr. 29, 2022).

[15] S. Robertson and J. Robertson, *Mastering the Requirements Process Getting Requirements Right*, 3rd ed. Addison-Wesley Professional, 2012.

[16] S. Adam, M. Larsen, K. Jensen, and U. P. Schultz, "Rule-based Dynamic Safety Monitoring for Mobile Robots," 2016.

[17] M. Ma, J. A. Stankovic, and L. Feng, “Runtime Monitoring of Safety and Performance Requirements in Smart Cities,” in *Proceedings of the 1st ACM Workshop on the Internet of Safe Things*, Nov. 2017, pp. 44–50. doi: 10.1145/3137003.3137005.

[18] P. Damacharla, A. Y. Javaid, J. J. Gallimore, and V. K. Devabhaktuni, “Common Metrics to Benchmark Human-Machine Teams (HMT): A Review,” *IEEE Access*, vol. 6, pp. 38637–38655, 2018, doi: 10.1109/ACCESS.2018.2853560.

[19] K. Bardsiri, “QoS Metrics for Cloud Computing Services Evaluation Amid,” 2016.

[20] F. Fleurey and A. Solberg, “A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems,” 2009, pp. 606–621. doi: 10.1007/978-3-642-04425-0_47.

[21] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, “Model evolution by run-time parameter adaptation,” in *2009 IEEE 31st International Conference on Software Engineering*, 2009, pp. 111–121. doi: 10.1109/ICSE.2009.5070513.

[22] L. H. G. Paucar and N. Bencomo, “RE-STORM: Mapping the Decision-Making Problem and Non-functional Requirements Trade-Off to Partially Observable Markov Decision Processes,” *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 19–25, 2018.

[23] J. Pearl, “The Do-Calculus Revisited,” *Uncertainty in Artificial Intelligence - Proceedings of the 28th Conference, UAI 2012*, pp. 4–11, Oct. 2012, doi: 10.48550/arxiv.1210.4852.

[24] J. Y. Halpern, “From Causal Models To Counterfactual Structures,” Jun. 2011.

[25] J. Sekhon, “The Neyman— Rubin Model of Causal Inference and Estimation Via Matching Methods,” in *The Oxford Handbook of Political Methodology*, Oxford University Press, 2009, pp. 271–299. doi: 10.1093/oxfordhb/9780199286546.003.0011.

[26] J. Pearl, *Causality*. Cambridge University Press, 2009. doi: 10.1017/CBO9780511803161.

[27] "pcalg: Methods for Graphical Models and Causal Inference version 2.7-8 from CRAN." https://rdrr.io/cran/pcalg/ (accessed Jan. 28, 2023).

[28] C. Aliferis, I. Tsamardinos, A. Statnikov, and L. Brown, *Causal Explorer: A Causal Probabilistic Network Learning Toolkit for Biomedical Discovery*. 2003.

[29] "TETRAD - Tools for causal inference and search (Carnegie Mellon University)." https://github.com/cmu-phil/tetrad (accessed Jan. 28, 2023).

[30] Y. Shimoni *et al.*, "An Evaluation Toolkit to Guide Model Selection and Cohort Definition in Causal Inference," Jun. 2019.

[31] A. Fahmi *et al.*, "Causal Bayesian Networks for Medical Diagnosis: A Case Study in Rheumatoid Arthritis," in *2020 IEEE International Conference on Healthcare Informatics (ICHI)*, 2020, pp. 1–7. doi: 10.1109/ICHI48887.2020.9374327.

[32] J. Yu, V. A. Smith, P. P. Wang, A. J. Hartemink, and E. D. Jarvis, "Advances to Bayesian network inference for generating causal networks from observational biological data," *Bioinformatics*, vol. 20, no. 18, pp. 3594–3603, Dec. 2004, doi: 10.1093/bioinformatics/bth448.

[33] M. Kliangkhlao, S. Limsiroratana, and B. Sahoh, "The Design and Development of a Causal Bayesian Networks Model for the Explanation of Agricultural Supply Chains," *IEEE Access*, vol. 10, pp. 86813–86823, 2022, doi: 10.1109/ACCESS.2022.3199353.

[34] "BayesiaLab." https://www.bayesia.com/articles/#!bayesialab-knowledge-hub/bayesialab-overview (accessed Jan. 28, 2023).

[35] "GeNIe Modeler – BayesFusion." https://www.bayesfusion.com/genie/ (accessed Jan. 28, 2023).

[36] J. Salvatier, T. v. Wiecki, and C. Fonnesbeck, "Probabilistic programming in Python using PyMC3," *PeerJ Comput Sci*, vol. 2, p. e55, Apr. 2016, doi: 10.7717/peerj-cs.55.

[37] M. Glinz, “Quality requirements and their role in successful products,” in *Proceedings - 15th IEEE International Requirements Engineering Conference, RE 2007*, 2007, pp. 21–26. doi: 10.1109/RE.2007.45.

[38] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, “Towards a better understanding of context and context-awareness,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1707, no. January, pp. 304–307, 1999, doi: 10.1007/3-540-48157-5_29.

[39] BBC News, “Pepper robot to work in Belgian hospitals,” 2016. https://www.bbc.com/news/technology-36528253

[40] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[41] R. G. Miller, *Survival Analysis*, 2nd Editio. John Wiley & Sons, 2011.

[42] J. C. Baird and E. Noma, *Fundamentals of scaling and psychophysics*. John Wiley & Sons, 1978.

[43] A. Jøsang, *Subjective logic : a formalism for reasoning under uncertainty*. 2016.

[44] C. Forbes, M. Evans, N. Hastings, and B. Peacock, *Statistical distributions*, 4th ed. Hoboken, N.J: Wiley, 2010.

[45] L. Devroye, *Non-Uniform Random Variate Generation*. Springer New York, 1986. doi: 10.1007/978-1-4613-8643-8.

[46] J. R. Norris, *Markov Chains*. Cambridge University Press, 1997. doi: 10.1017/CBO9780511810633.

[47] V. G. Kulkarni, *Modeling and analysis of stochastic systems: Third edition*. CRC Press, 2016. doi: 10.1201/9781315367910.

[48] M. G. (Maurice G. Kendall, A. Stuart, and J. K. Ord, “Kendall’s advanced theory of statistics,” 1987.

[49] A. Eckner, “Algorithms for Unevenly Spaced Time Series : Moving Averages and Other Rolling Operators,” 2015.

[50] W. Gautschi and W. F. Gahill, “Exponential Integral and Related Functions,” in *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables*, Dover Publications, Inc., 1974.

[51] D. A. Barry, J. Y. Parlange, and L. Li, “Approximation for the exponential integral (Theis well function),” *J Hydrol (Amst)*, vol. 227, no. 1–4, pp. 287–291, Jan. 2000, doi: 10.1016/S0022-1694(99)00184-5.

[52] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[53] J. W. Hines, “Logarithmic neural network architecture for unbounded non-linear function approximation,” *IEEE International Conference on Neural Networks - Conference Proceedings*, vol. 2, pp. 1245–1250, 1996, doi: 10.1109/ICNN.1996.549076.

[54] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” Mar. 2016, doi: 10.48550/arxiv.1603.04467.

[55] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[56] C. E. Shannon, “A Mathematical Theory of Communication,” *Bell System Technical Journal*, vol. 27, no. 4, pp. 623–656, 1948, doi: 10.1002/J.1538-7305.1948.TB00917.X.

[57] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, doi: https://doi.org/10.1016/0893-6080(89)90020-8.

[58] H. B. Mcmahan *et al.*, “Ad click prediction: A view from the trenches,” *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data*

Mining, vol. Part F128815, pp. 1222–1230, Aug. 2013, doi: 10.1145/2487575.2488200.

[59] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, 2nd edition. Morgan & Claypool, 2017.

[60] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd edition. Addison-Wesley Professional, 2008.

[61] "Xtext." www.eclipse.org/Xtext (accessed Jan. 01, 2023).

[62] "OMG: Object Constraint Language (OCL)." https://www.omg.org/spec/OCL/ (accessed Jan. 19, 2023).

[63] "OMG: Data-Distribution Service for Real-Time Systems (DDS)." https://www.omg.org/dds/ (accessed Jan. 19, 2023).

[64] "OASIS: Message Queuing Telemetry Transport (MQTT)." https://mqtt.org/ (accessed Jan. 19, 2023).

[65] Google, "Protocol Buffers," *https://developers.google.com/protocol-buffers*.

[66] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[67] O. Etzion and P. Niblett, *Event Processing in Action*, 1st edition. Manning Publications Co., 2010.

[68] "RoQME: Dealing with non-functional properties through global Robot Quality-of-Service Metrics." https://robmosys.eu/roqme/ (accessed Jan. 18, 2023).

[69] "EsperTech Inc.: Esper language, compiler and runtime for Complex Event Processing (CEP)." https://www.espertech.com/esper/ (accessed Jan. 18, 2023).

[70] "Xtend." https://www.eclipse.org/xtend/index.html (accessed Jan. 18, 2023).

[71] "RobMoSys: Composable Models and Software for Robotics." https://robmosys.eu/ (accessed Jan. 20, 2023).

[72] "RobMoSys: Intralogistics Industry 4.0 Robot Fleet Pilot."

[73] D. Stampfer, A. Lotz, M. Lutz, and C. Schlegel, "The SmartMDSD Toolchain: An Integrated MDSD Workflow and Integrated Development Environment (IDE) for Robotics Software," *Journal of Software Engineering for Robotics (JOSER)*, vol. 7, pp. 3–19, Aug. 2016.

[74] "RoQME Demonstration: Dealing with Metrics on Non-Functional Properties in RobMoSys," *https://robmosys.eu/wiki/community:roqme-intralog-scenario:start*.

[75] "CLARC: Smart Clinic Assistant Robot for CGA. The European Coordination Hub for Open Robotics Development (ECHORD++)." https://echord.eu/essential_grid/clark/ (accessed Jan. 22, 2023).

[76] F. I. Mahoney and D. W. Barthel, "Functional Evaluation: The Barthel Index," *Md State Med J*, vol. 14, pp. 61–65, 1965, [Online]. Available: http://europepmc.org/abstract/MED/14258950

[77] S. Mathias, U. S. Nayak, and B. Isaacs, "Balance in elderly patients: the 'get-up and go' test," *Arch Phys Med Rehabil*, vol. 67, no. 6, pp. 387–389, 1986, [Online]. Available: http://europepmc.org/abstract/MED/3487300

[78] A. Romero-Garcés, J. Martínez-Cruz, J. F. Inglés-Romero, C. Vicente-Chicote, R. Marfil, and A. Bandera, "Measuring Quality of Service in a Robotized Comprehensive Geriatric Assessment Scenario," *Applied Sciences*, vol. 10, no. 18, 2020, doi: 10.3390/app10186618.

[79] R. Marfil *et al.*, "Perceptions or Actions? Grounding How Agents Interact Within a Software Architecture for Cognitive Robotics," *Cognit Comput*, vol. 12, no. 2, pp. 479–497, 2020, doi: 10.1007/s12559-019-09685-5.

[80] C. Vicente-Chicote, D. García-Pérez, P. García-Ojeda, J. F. Inglés-Romero, A. Romero-Garcés, and J. Martínez, "Modeling and Estimation of Non-functional Properties: Leveraging the Power of QoS Metrics," in *From Bioinspired Systems and Biomedical Applications to Machine Learning*, 2019, pp. 380–388.

[81] C. Vicente-Chicote *et al.*, “A Component-Based and Model-Driven Approach to Deal with Non-Functional Properties through Global QoS Metrics,” in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2018.

[82] J. J. Dujmovic, “Continuous Preference Logic for System Evaluation,” *IEEE Transactions on Fuzzy Systems*, vol. 15, no. 6, pp. 1082–1099, 2007, doi: 10.1109/TFUZZ.2007.902041.

[83] J. J. Dujmović and H. L. Larsen, “Generalized conjunction/disjunction,” *International Journal of Approximate Reasoning*, vol. 46, no. 3, pp. 423–446, Dec. 2007, doi: 10.1016/J.IJAR.2006.12.011.

[84] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *Int J Rob Res*, vol. 32, no. 11, pp. 1238–1274, Sep. 2013, doi: 10.1177/0278364913495721.

Appendix A

A.1 Xtext Grammar Specification

```
grammar qosme.QoSMetrics with org.eclipse.xtext.common.Terminals

import "http://www.eclipse.org/emf/2002/Ecore" as ecore
import "platform:/resource/qosme.metamodel/metamodel/kernel/kernel.ecore" as kernel
import "platform:/resource/qosme.metamodel/metamodel/datatypes/datatypes.ecore" as datatypes
import "platform:/resource/qosme.metamodel/metamodel/expressions/expressions.ecore" as expressions


Model returns datatypes::Model:
        ('namespace' namespace=EString)?
        (sentences+=Import)*
        ( sentences += TimeFrame
        | dataTypes += DataTypeDefinition
        | variables += TypedVariable
        | sentences += Sentence)*;


/* ********************************
 * KERNEL EPACKAGE
 * ********************************/

/*
 * Variables
 */

TypedVariable returns datatypes::TypedVariable:
        Parameter | Timer | GeneralPurposeVariable | Context | Property;

Parameter returns kernel::Parameter:
        'param' name=ID ':' declaration = DataTypeDeclaration
        ('default' default=TypedValue)?;

Timer returns kernel::Timer:
        'timer' name=ID declaration = ImplicitTimeTypeDeclaration
        '=' period=NumericTimeValue;

GeneralPurposeVariable returns kernel::GeneralPurposeVariable:
        'var' name=ID ':' declaration = DataTypeDeclaration
        '=' definition=Expression;
```

```
Context returns kernel::Context:
        DerivedContext | PrimitiveContext;

PrimitiveContext returns kernel::PrimitiveContext:
        'context' name=ID ':' declaration = DataTypeDeclaration;

DerivedContext returns kernel::DerivedContext:
        'context' name=ID ':' declaration = DataTypeDeclaration
                '=' definition = Expression;

Property returns kernel::Property:
        'property' name=ID declaration=ImplicitNumericTypeDeclaration;


/*
 * Sentencies
 */

Import returns kernel::Import:
        'import' importURI=EString
        (settings += SetParameter)*;

SetParameter returns kernel::SetParameter:
        'with' parameter = [kernel::Parameter | QualifiedName]
        '=' value=TypedValue;

Sentence returns datatypes::Sentence:
        Observation;

Observation returns kernel::Observation:
        'observation' name=ID ':' pattern=Expression
        (('repetition' repetition=INT)?
                & ('persistence' persistence=TimeValue)?)
        (actions+=Action | ('{' (actions+=Action)+ '}'));

Action returns kernel::Action:
        SetVariable | ClearEvidence | SetEvidence;

SetVariable returns kernel::SetVariable:
        'sets' variable = [kernel::GeneralPurposeVariable | QualifiedName]
        '=' expression=Expression;

ClearEvidence returns kernel::ClearEvidence:
        'clears' property = [kernel::Property | QualifiedName];

SetEvidence returns kernel::SetEvidence:
        direction=DirectionEnum
        property = [kernel::Property | QualifiedName]
        (strength=StrengthEnum)?;

enum DirectionEnum returns kernel::DirectionEnum:
        REINFORCE = 'reinforces' | UNDERMINE = 'undermines';
```

```
enum StrengthEnum returns kernel::StrengthEnum:
        VERY_HIGH = 'veryhigh' | VERY_HIGH = 'very high' |
        HIGH = 'high' | MEDIUM | LOW = 'low' |
        VERY_LOW = 'verylow' | VERY_LOW = 'very low';




/* ********************************
 * DATATYPE EPACKAGE
 * ********************************/
/*
 * Data type definitions
 */

DataTypeDefinition returns datatypes::DataTypeDefinition:
        'type' name=EString ':' dataType=DataType;

DataType returns datatypes::DataType:
        BooleanType | EnumType | EventType | NumericType | TimeType;

BooleanType returns datatypes::BooleanType:
        {datatypes::BooleanType}
        'boolean';

EnumType returns datatypes::EnumType:
        'enum'
        '{'
                literals += EnumLiteral (',' literals += EnumLiteral )+
        '}';

EnumLiteral returns datatypes::EnumLiteral:
        {datatypes::EnumLiteral}
        name=ID;

EventType returns datatypes::EventType:
        {datatypes::EventType}
        'event';

NumericType returns datatypes::NumericType:
        {datatypes::NumericType}
        'number';

TimeType returns datatypes::TimeType:
        {datatypes::TimeType}
        'time';

/*
 * Typed Values
```

```
 */

TypedValue returns datatypes::TypedValue:
        BooleanValue | EnumValue | EventValue | NumericValue | TimeValue;

BooleanValue returns datatypes::BooleanValue:
        {datatypes::BooleanValue}
        value = EBoolean;

EnumValue returns datatypes::EnumValue:
        {datatypes::EnumValue}
        value=[datatypes::EnumLiteral | LiteralQualifiedName];

EventValue returns datatypes::EventValue:
        {datatypes::EventValue}
        'trigger';

NumericValue returns datatypes::NumericValue:
        {datatypes::NumericValue}
        value=EDouble;

TimeValue returns datatypes::TimeValue:
        AbsoluteTime | RelativeTime;

AbsoluteTime returns datatypes::AbsoluteTime:
        NumericTimeValue | AbsoluteTimeExpression;

NumericTimeValue returns datatypes::NumericTimeValue:
        {datatypes::NumericTimeValue}
        value=EDouble unit=TimeUnitEnum;

AbsoluteTimeExpression returns datatypes::AbsoluteTimeExpression:
        {datatypes::AbsoluteTimeExpression}
        quantity=AbsoluteQuantifierEnum unit=TimeUnitEnum;

enum TimeUnitEnum returns datatypes::TimeUnitEnum:
        SECOND = 'second' | SECOND = 'seconds' |
        MINUTE = 'minute' | MINUTE = 'minutes' |
        HOUR = 'hour' | HOUR = 'hours' |
        DAY = 'day' | DAY = 'days' |
        WEEK = 'week' | WEEK = 'weeks' |
        MONTH = 'month' | MONTH = 'months';

enum AbsoluteQuantifierEnum returns datatypes::AbsoluteQuantifierEnum:
        FEW = 'few' | SOME = 'some' | MANY = 'many';

RelativeTime returns datatypes::RelativeTime:
        RelativeTimeExpression;

RelativeTimeExpression returns datatypes::RelativeTimeExpression:
        {datatypes::RelativeTimeExpression}
        quantity=RelativeQuantifierEnum;
```

```
enum RelativeQuantifierEnum returns datatypes::RelativeQuantifierEnum:
        SHORT = 'short' | MEDIUM = 'medium' | LONG = 'long';

TimeFrame returns datatypes::TimeFrame:
        'timeframe' value=AbsoluteTime;

/*
 * Typed Variables
 */

DataTypeDeclaration returns datatypes::DataTypeDeclaration:
        BuiltinTypeDeclaration | ExternalTypeDeclaration;

BuiltinTypeDeclaration returns datatypes::BuiltinTypeDeclaration:
        type=DataType;

ExternalTypeDeclaration returns datatypes::ExternalTypeDeclaration:
        type=[datatypes::DataTypeDefinition];

ImplicitTimeTypeDeclaration returns datatypes::BuiltinTypeDeclaration:
        type=ImplicitTimeType;

ImplicitTimeType returns datatypes::TimeType:
        {datatypes::TimeType};

ImplicitNumericTypeDeclaration returns datatypes::BuiltinTypeDeclaration:
        type=ImplicitNumericType;

ImplicitNumericType returns datatypes::NumericType:
        {datatypes::NumericType};


/* ********************************
 * EXPRESSIONS EPACKAGE
 * ********************************/

Expression returns expressions::Expression:
        term = Term;

Term returns expressions::Term:
        (OnceTerm | WhileTerm);

/*
 * Event Pattern Operators
 */

OnceTerm returns expressions::OnceTerm:
        'once' term=WhileTerm;

WhileTerm returns expressions::Term:
        RepetionTerm
        ({expressions::WhileTerm.left=current} 'while' '(' right=OrBooleanTerm ')')?;
```

```
RepetionTerm returns expressions::Term:
	SequenceTerm
	(({expressions::RepeatTerm.term=current} 'repeat' '(' nrep=INT ')')
		| ({expressions::RangeTerm.term=current} 'range' '(' lowerBound=INT ',' upperBound=INT ')'))?;

SequenceTerm returns expressions::Term:
	OrEventTerm
	({expressions::FollowedByTerm.left=current} '->'
		right=ConditionalTerm
	)*;

OrEventTerm returns expressions::Term:
	AndEventTerm
	({expressions::OrEventTerm.left=current} 'or' right=AndEventTerm)*;

AndEventTerm returns expressions::Term:
	(NotEventTerm | ConditionalTerm)
	({expressions::AndEventTerm.left=current} 'and' right= (NotEventTerm | ConditionalTerm) )*;

NotEventTerm returns expressions::Term:
	{expressions::NotEventTerm}
	'not' term=ConditionalTerm;

/*
 * Conditional Operator
 */

ConditionalTerm returns expressions::Term:
	OrBooleanTerm
	({expressions::ConditionalTerm.terms+=current} '?' terms+=OrBooleanTerm ':'
terms+=ConditionalTerm)?;


/*
 * Logical Terms
 */

OrBooleanTerm returns expressions::Term:
	AndBooleanTerm
	({expressions::OrBooleanTerm.left=current} '|' right=AndBooleanTerm)*;

AndBooleanTerm returns expressions::Term:
	(NotBooleanTerm | RelationalTerm)
	({expressions::AndBooleanTerm.left=current} '&' right= (NotBooleanTerm | RelationalTerm) )*;

NotBooleanTerm returns expressions::Term:
	{expressions::NotBooleanTerm}
	'!' term=RelationalTerm;

/*
 * Relational Term
 */
```

```
RelationalTerm returns expressions::Term:
        AdditionTerm (({expressions::EqualTerm.left=current} '=='
                | {expressions::LessThanTerm.left=current} '<'
                | {expressions::GreaterThanTerm.left=current} '>'
                | {expressions::LessEqualTerm.left=current} '<='
                | {expressions::GreaterEqualTerm.left=current} '>='
                | {expressions::NotEqualTerm.left=current} '!='
        )
        right=AdditionTerm)?;

/*
 * Arithmetic Terms
 */

AdditionTerm returns expressions::Term:
        MultiplicationTerm
        (({expressions::AddTerm.left=current} '+' | {expressions::SubTerm.left=current} '-')
right=MultiplicationTerm)*;

MultiplicationTerm returns expressions::Term:
        PrimaryTerm
        (({expressions::MultTerm.left=current} '*' | {expressions::DivTerm.left=current} '/') right=PrimaryTerm)*;

/*
 * Primary Terms
 */

PrimaryTerm returns expressions::Term:
        '(' Term ')' |
        ConstTerm |
        VarTerm |
        FunctionTerm;

VarTerm returns expressions::VarTerm:
        variable = [datatypes::TypedVariable | QualifiedName];

ConstTerm returns expressions::ConstTerm:
        value = TypedValue;

/*
 * Functions
 */

FunctionTerm returns expressions::FunctionTerm:
        ArithFunction | AggregationFunction | PatternFunction;

ArithFunction returns expressions::ArithFunction:
        (terms+=VarTerm "'")? name=ArithFtnEnum ('(' (terms += Term (',' terms += Term)*)? ')')?;

enum ArithFtnEnum returns expressions::ArithFtnEnum:
        POW = 'pow' | SQRT = 'sqrt' | EXP = 'exp' | ABS = 'abs';

AggregationFunction returns expressions::AggregationFunction:
```

```
        (terms+=VarTerm "'")? name=AggregationFtnEnum ('(' (terms += Term (',' terms += Term)*)? ')')?;

enum AggregationFtnEnum returns expressions::AggregationFtnEnum:
        AVG = 'avg' | MIN = 'min' | MAX = 'max' | COUNT = 'count' | SUM = 'sum' | DECREASING = 'decreasing' |
INCREASING = 'increasing' | STABLE = 'stable';

PatternFunction returns expressions::PatternFunction:
        (terms+=VarTerm "'")? name=PatternFtnEnum ('(' (terms += Term (',' terms += Term)*)? ')')?;

enum PatternFtnEnum returns expressions::PatternFtnEnum:
        EVENT_WHEN = 'eventWhen' | UPDATE = 'update' | PERIOD = 'interval';


/* *********************************
 * Auxiliary grammar
 * *********************************/

EString returns ecore::EString:
        STRING | ID;

EDouble returns ecore::EDouble:
        '-'? INT ('.' INT)?;

EBoolean returns ecore::EBoolean:
        'true' | 'false';

QualifiedName returns ecore::EString:
        ID ('.' ID)*;

LiteralQualifiedName returns ecore::EString:
        ID ('::' ID)+;

@Override
terminal ML_COMMENT :
  ('/*' !'*') -> '*/';

@Override
terminal STRING :
  '"' ( '\\'('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|'"') )* '"';
```