POLITECNICO
MILANO 1863

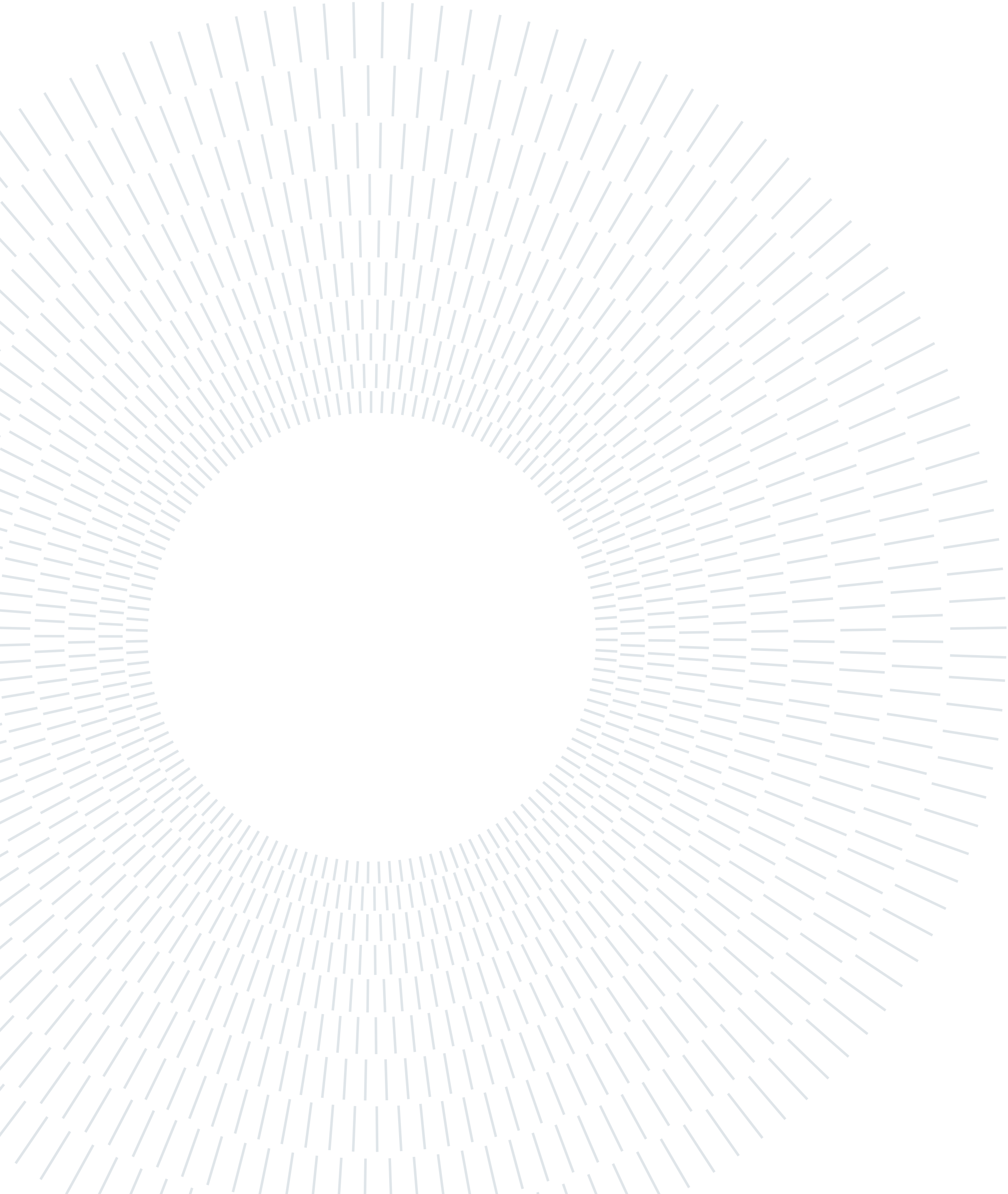
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Development of a User-Friendly Testing Tool for the Gas Meter Industry

TESI DI LAUREA MAGISTRALE IN
AUTOMATION AND CONTROL ENGINEERING - INGEGNERIA DELL'AUTOMAZIONE

Author: **Saul Andres Ibarra Tejera**

Student ID: 949802
Advisor: Prof. Matteo Giovanni Rossi
Co-advisors: Eng. Claudio Bianchi
Academic Year: 2021-22



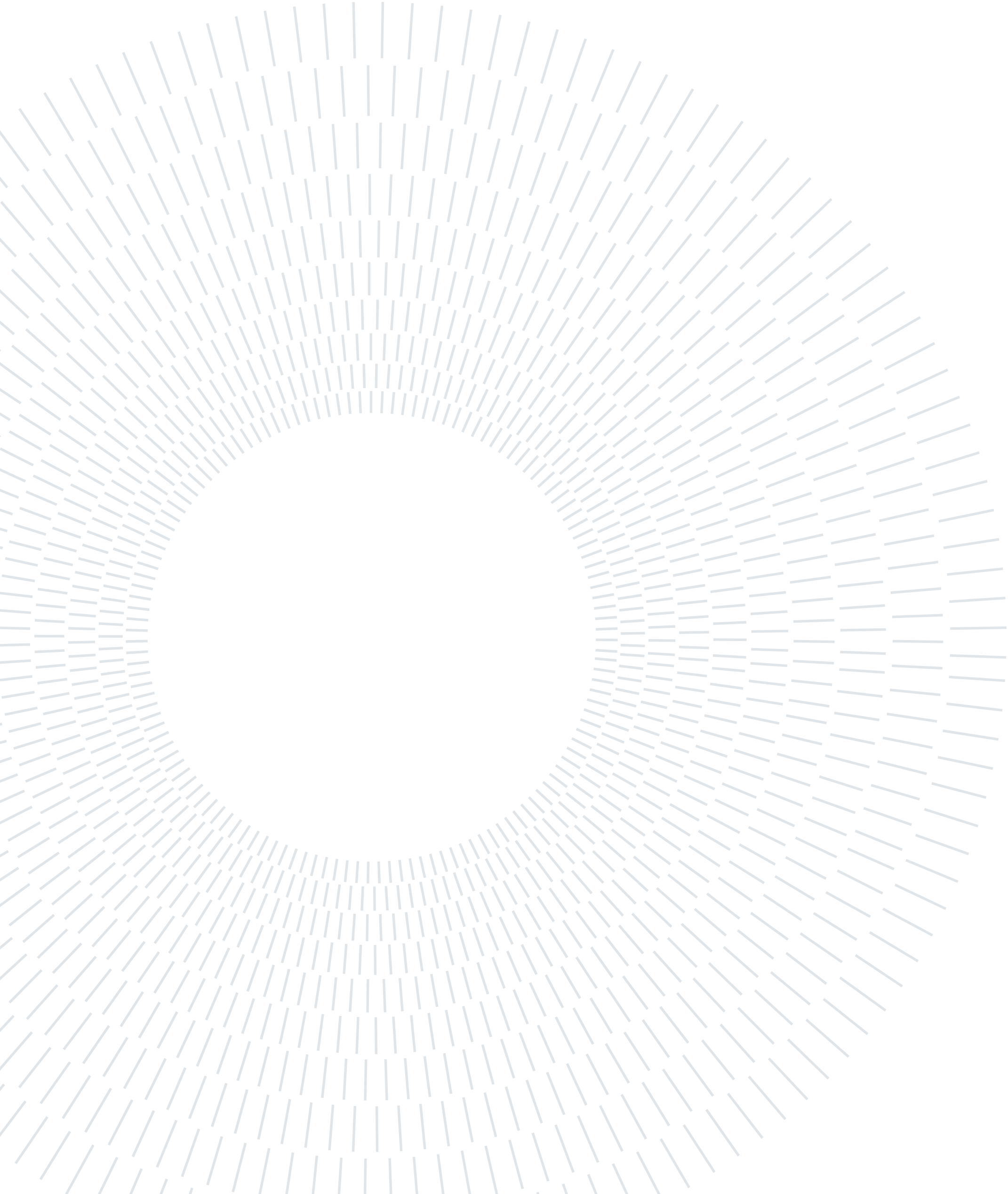
Abstract

The automatic testing processes are every day becoming more essential to increase productivity, reliability and save time during the production processes. Inside the gas meter industry, there is high competition to cover the existing and new demand due to the change of conventional meters for smart metering devices.

The goal of this thesis is to generate a new option to design and execute automatic testing for the gas meter industry. For this purpose, it is introduced a Domain Specific language (DSL) designed to be a simple and understandable way to generate scripts for automatic testing. To work along with the DSL it is developed a custom tool, which works as the link between the tester, meter, and scripts. The tool is in charge of interpreting the DSL script, executing the instructions, communicating with the meter, and presenting results to the user.

The thesis will introduce the DSL, present the resultant tool, discuss their interaction, and finally show the results of the validation. Additionally, through this text, there are going to be included some resultant scripts created for real testing, applying the DSL, and executed by the testing tool.

Keywords: DSL, testing tool, autonomous testing.



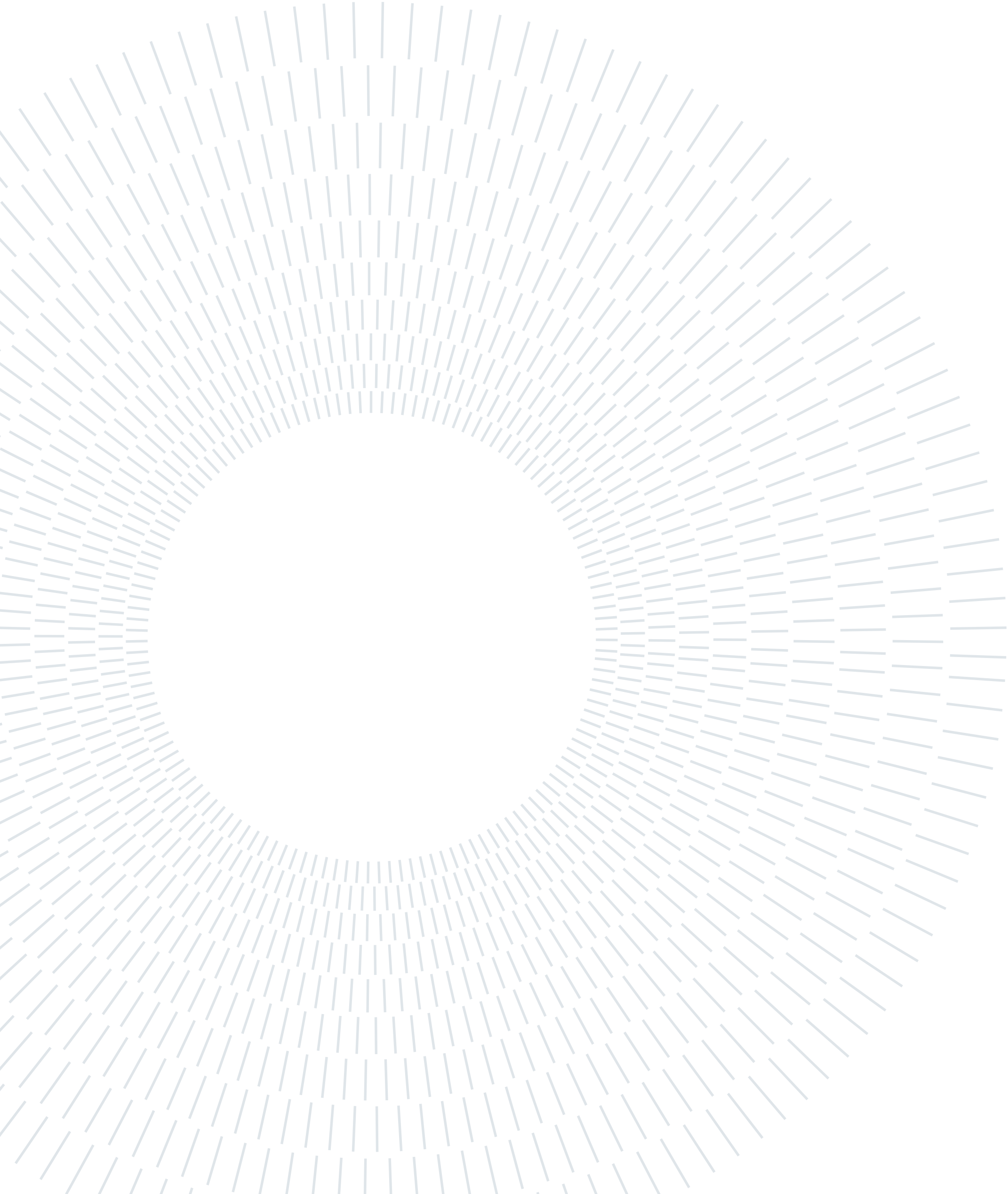
Abstract in lingua italiana

L'automazione dei processi di testing sta diventando un'operazione sempre più essenziale per migliorare la produttività e l'affidabilità dei processi produttivi e per ridurne i tempi nei processi produttivi. All'interno dell'industria dei contatori di gas c'è una alta competitività per coprire la domanda, esistente e nuova, a causa del ricambio di quelli tradizionali con contatori intelligenti.

L'obiettivo di questa tesi è quello di progettare un nuovo modo/metodo/procedimento per l'esecuzione di test automatizzati per l'industria dei contatori di gas. A tal fine, si propone un Domain Specific language (DSL, un linguaggio specifico di dominio) specificatamente progettato per essere un modo semplice e chiaro di creare scripts per test automatizzati. Per poter lavorare con il DSL è stato sviluppato uno specifico (custom) tool, che funge da elemento di collegamento tra i tester, i contatori e gli scripts. Il tool interpreta lo script DSL, esegue le istruzioni, comunica con il contatore e mostra i risultati all'utente.

In questa tesi verrà introdotto il DSL e presentato il tool finale. Verrà analizzata e discussa la loro interazione e, infine, verranno esposti i risultati ottenuti. In aggiunta, verranno inclusi alcuni script creati per test reali e i risultati doppo essere eseguiti dal tool.

Keywords: DSL, testing tool, test automatici.



Contents

1 | Introduction

In 2013, Italy started a massive and ongoing process of substituting all the existing gas meters for smarter versions. *The Energy Network and The Environment Regional Authority* known under the acronym ARERA in Italian stated in regulation N.631/2013/R/GAS that from 2013 the obligation for all local distributors to substitute the old and previously used devices with new smart gas meters [22]. This was the start of a complete market, with Italy as a leading country in the implementation of smart gas meters in the Europe Union and worldwide.

Consequently, the demand started to grow exponentially together with the requirements of the production companies. The testing processes are one of the more time-consuming activities during production and development. Many testing programs help the tester facilitate the testing procedure and reduce the time consumed. In some cases, these tools can reach a level of automation where they can carry out a complete test in their own.

This thesis also introduces a Domain Specific Language created to build scripts for automatic tests. It is presented how a DSL can work as a solution for companies that requires a high level of automation in the testing process. In addition, it offers highly comprehensible language, is easy to manipulate, and adapt, and it can be introduce in highly efficient teams.

For this tool, it is also introduced a Domain Specific Language created to build scripts for automatic tests. It is presented how a DSL can work as a solution for companies that requires a high level of automation in the testing process. Plus at the same time offer a highly comprehensible language, easy to manipulate, adapt and introduced in highly efficient teams.

It is explained how the tool was designed to interpret the DSL and how integrating the two parts can save time and make the testing process more reliable, and efficient. Moreover, how a real test can be reproduced in a completely autonomous mode.

Below is presented an overview of the content of each chapter of the thesis.

- Chapter 2 presents a review of the actual situation of the gas meter industry in Italy and the software testing.

- Chapter 3 introduces the basic concepts that are useful to understand the theory, regulations and protocols behind the thesis.
- Chapter 4 discuss how the project was planned and the initial considerations. Also, it treats the different requirements that the project is expected to meet at the end of this thesis.
- Chapter 5 introduces the Domain Specific Language designed for the gas meter industry. It shows the main elements, a breve description of all key words and how they can be used to generate the test scripts. It also presents two real test where the DSL were implemented.
- Chapter 6 describe the testing tool session by session as it was obtained at the end of the thesis. In addition, it will be shown how it is the interaction between tool and DSL.
- Chapter 7 provides all the instruments that were used to verify the correct functionality of the tool and language. It also present the results obtained concerning the initial requirements.
- Chapter 8 presents the conclusions after the development of the complete project, a small recap of the results, and the next steps to continue with this project.
- Appendix A, and B contains the log files generated by the tool as results of the example tests exhibit during the thesis.

2 | State of the art

This chapter presents the actual situation of the Italian Gas industry, which is the industry of relevance for this thesis. Additionally, the second part of the chapter covers the topic of software testing, going into detail on the main components and the different levels of automation. All of this is to generate a better understanding of the actual situation in which this project was involved and to reaffirm the relevance of the implementation of automation in the testing processes.

2.1. Italian Gas Industry

This section presents a brief review and description of the Italian gas meter. The goal of the chapter is to understand the current situation of the industry and to explain the standards required by the market.

After the new obligations set by *The Energy Network and The Environment Regional Authority (ARERA)*, the gas meter industry underwent a complete restructuring to adapt to the new requirements. These obligations said all gas meters in the Italian territory had to be replaced by smart metering devices. The term smart meters refers to the new devices that have remote metering and management capabilities. That generated an accelerated incursion of many companies into the IoT market.

Together with the obligations, all kinds of regulations and standards for this new smart product were published. This was all made to make it possible to standardize devices regardless of the country where it is located. A new product of this nature requires the standardization of regulations and protocols to ensure the quality and legal compliance of each product. This is especially important in the European Union context, where it is required to work with different legislation. The requirements published are communication protocols, specifications, and functionalities, among many others. In the beginning, a block diagram of the functional modules was established for all metering units. The block diagram is presented below in Figure 2.1.

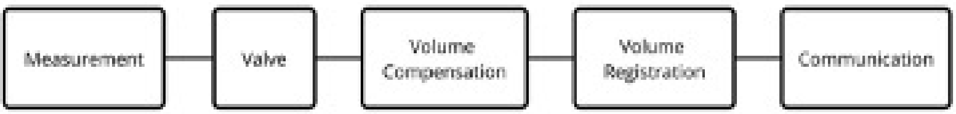


Figure 2.1: Functional modules for a metering system.

This diagram represents the main workflow of the meter's operation. The first component in the block diagram refers to the metering functionality that it can be done with any measurement technique (volumetric diaphragm, volumetric rotary, ultrasonic, or Coriolis). The valve in the second block is implemented to interrupt the gas supply. The volume compensation is made to compute the volume measured taking into consideration the gas temperature and referring it to the referent temperature. The next block is the registration of the obtained information. Finally, the transference of the information from the meter to the supplier company [7]. The following subsection looks into details the requirements and specifications of the communication block, which is a fundamental background to understand the scope of the thesis.

2.1.1. Communication requirements

The communication infrastructure used in all smart meters is an essential part of its functionality. In the process of telemanagement and telemetering, the communication system must function at all times with as few errors as possible. Directly related to its high importance, it has to be severely regulated.

All the regulations are presented and established in the normative UNI/TS 11291-12. It is divided into four parts that include use cases, data models, transmission networks, and requirement evaluations [8–11]. The technical requirements specified were established with the main scope of assuring the interconnectivity with all the measurement devices for natural gas on the market that works in the network point-to-point. Specifically, residential gas meters with a maximum flow rate of 10 m^3/h (<G10). This norm suggests the implementation of two different communication networks, GPRS/UMTS/LTE or NB-IoT.

The GPRS/UMTS/LTE are well-known cell phone networks. Each one of them works at different speeds and functionalities. But, in basic, the meters working with this technology have integrated a cell phone provider SIM card and a modem to connect with the network, in the way to use it to communicate with the SAC (Central access system).

The NB-IoT is a technology designed for devices with low power consumption. In particular, this characteristic is fundamental to fulfilling requirements such as the long battery life. In addition, the network was designed to be implemented in rural and deep indoor environments which require extended coverage, and low device complexity. The implementation price is in the same range as the previous technology, and with the rapid increase in popularity is expected a considerable decrease in price.

2.2. Software testing

The importance of software has increased, and it has become an essential part of almost all electronic devices appearing in daily basic activities. According to The Cambridge dictionary, the software is defined as *'The instructions that control what a computer does'* [24]. But with deeper research in the area and by using technical analysis of the topic, it would be possible to define software as the composition of programs, scripts, instructions, and routines related to the operation of a computer process [12].

One of the fundamental parts needed in the software development process is testing. Software testing is necessary to verify the requirements and specifications established. More specifically, software testing is the procedure implemented to verify and validate software functionality. Testing a program consists of much more than just running a code and looking for errors. It executes a program under defined conditions, verifying that it behaves as expected, detects anomalies, and finally validates the accomplishment of the client's requirements [23].

The testing of the software is the part that consumes most of the time, effort, and budget of the development. Until 50% of the software development costs are occupied only by the testing procedures [6]. Therefore, in the past years, companies have to pay more attention to this and the importance of software testing. This has increased investments by companies in testing teams, personnel, and tools. Using a good testing procedure, allows the companies to identify bugs and errors in the products before giving them to the final client, resulting in saving millions of dollars and time.

There are substantially two fundamental parts required for the procedure of software testing: the tester and the testing tools.

2.2.1. Tests engineer

The testers refer to the people in charge of and working with the testing activities. Most commonly, they are IT professionals, with significant knowledge of the domain and the product's functionality. They are, therefore, involved in the different specifications and requirements needed. The labor of the tester consists mainly of following a variety of procedures, observing the behavior, collecting data, analyzing results, and highlighting errors, bugs, or anomalies. Another important thing to specify is that the testing activities are not just tasks performed by the testing engineers. These activities are something that every person involved in the development process of a product or software must execute during his working activities. The person who is best suited for testing the process is always the one who has the most knowledge about it. Usually, this person is the one who created the process in the first place.

2.2.2. Testing tool

The testing tools are specialized programs used to validate the functionalities of a product. The features of the testing tools make it a perfect complement and partner for the tester to do the work more efficiently and reliably. The testing tools execute a sequence of commands, decode data, analyze results, and make it easier to evaluate the behavior of the tested software.

Testing tool divisions: Specialization level

There are many different variants of testing programs currently on the market, for free or by payment. Commercial tools are designed to be implemented by many types of companies. They are capable to adapt to the specifications of each case. For this reason, they must be generic enough to work in many environments, fields, and industries. On the other hand, there are cases where these tools are not enough, where they are too generic. These situations require specialized programs to fulfill the company or product requirements. Consequently, the companies are forced to develop their custom tool, which requires both time and resources, but gives the benefits of having it completely adapted to the task.

Testing tool divisions: Automation level

Another division of the testing tool is in the degree of automation. The first category, manual testing, refers to the tools that mainly cover the function of executing actions and showing the response to the operator. These results are shown in the same way as the software reply, without manipulation or validation in the results. The final test result is the complete responsibility of the tester. A benefit of this type of testing tool is the simplicity in its composition, but on the other hand, it requires a lot of work and time from the testing engineers.

The other category is instead tools that are designed for automatic testing. These have the same function as the manual testing tools, but with the difference that these tools have the possibility to be programmable with different tests that can be used multiple times in the future without further interaction of the operator. This function allows repeating the same test multiple times, assuring similar conditions, and reducing human error during the script execution [16]. It thereby reduces the workload for the testing engineers by making the testing simple and reducing the costs of the testing process. After the execution of single or multiple tests, the automatic testing tool gives the partial and final information to the testing engineer for further analysis. Additionally, this type of tool generally makes a preliminary analysis, and it can generate a successful or failed result concerning the parameter established during the test creation.

The main benefit of using automatic testing tools are thereby that it gives the possibility to save time and money. At the same time gives more reliable testing results and improves the quality and efficiency of the testing procedure. Compared with manual testing, it is possible to show that it decreases human error, generates more test suits with less effort, and reduces the testing engineer workload [21].

2.2.3. Software testing for embedded systems

For the variety of embedded systems, it is impossible to define a general testing method that satisfies the requirements of every device and market. Each type of device has different measures to test the variety of fail cases that may be presented in the system. That is why there is no point in deeply analyzing one generic testing method that works to test all systems and their demands. Even though it is possible to find many factors to get through different testing processes for the different embedded systems, many common issues can be tested and treated in similar ways. In the end, some specific tests can be executed for a variety of systems, taking into account the multiple adaptations required for the test plan to correspond to the requirements of each embedded system [13].

The TEmb method

TEmb is a method that facilitates the design of a suitable test path for a specific device. It does contain a set of strategies that start from the generic parts, that works for every test, then it takes into consideration the specific measures and all the proprietary characteristics of the system to get as result a truly custom test approach.

The basic procedure starts with the classical elements that work for the test path of any embedded device. These elements consist of life-cycle, infrastructure, techniques, and organization (known as "LITO"). These four components are the funds of structured testing. The test approach is not appropriate yet, because it still requires the implementation of the specification of the device and the kind of test to be carried out. These requirements can be concerning the design techniques, infrastructure, and tools to be applied in the testing process. The first step for this method is based on the measures decided, that are relevant for the test procedure. These measures are taken based on the risks and system characteristics of the device on the test. Concerning the risks, it is possible to select the measurements to be taken into account to evaluate the business risks. They are related to the wrong functioning of the product. Instead, the system characteristics are the ones that help to define the problems related to the technical properties. These may be the complexity of the software, safety issues, interference, etc [13].

For any structured testing process, it is required to solve the questions "what and when", "how", "by what", and "by whom". Each of these questions is solved by one of the elements conforming to LITO. The basic idea is that all four LITO bases need to be fully covered to obtain a successful testing process. Life-cycle is the core of the process, it works as the link between the other components (see Figure. 2.2). Depending on the stage of the life-cycle, it may change the effect of the other three key parts on the testing procedure. From the moment when it reached a point where all four cornerstones are successfully covered, it is possible to define our test process as structured. However, as a result of this procedure is not possible to obtain a trouble-proof test, always there will be unexpected events that take the test to an unknown area creating a failure in the process. Equally, the structured test processes are more likely to overpass those dark zones and carry out the test to an admissible ending after a small aftereffect [13].

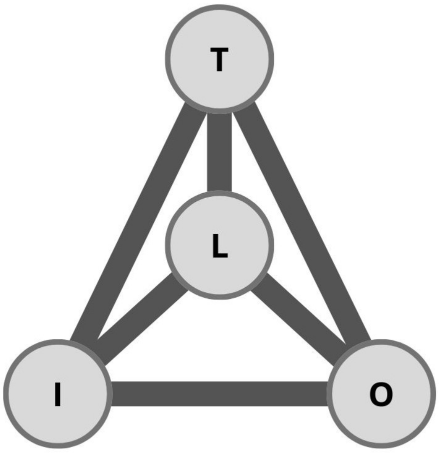
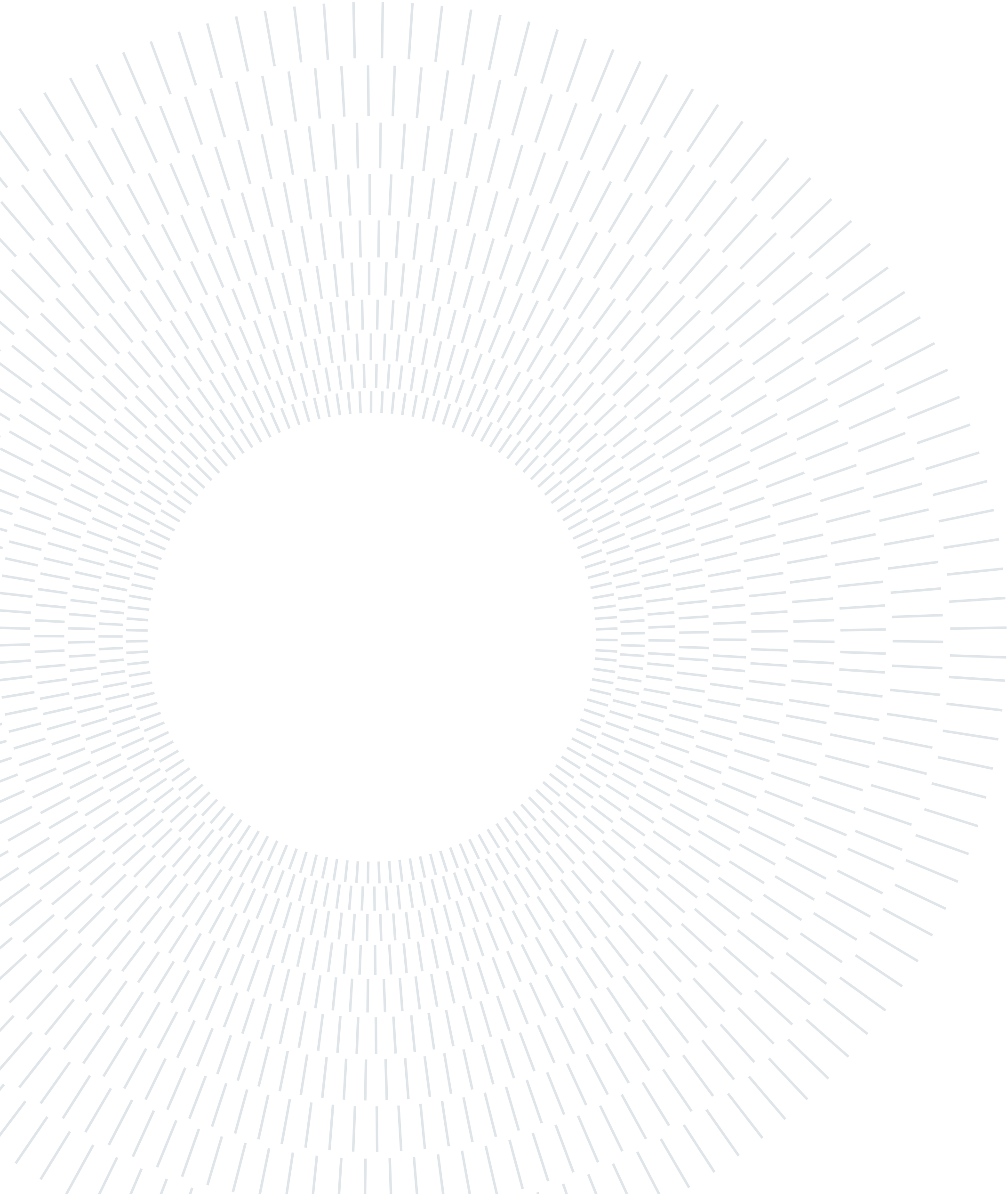


Figure 2.2: Components of a structured test process.

- **Life-cycle:** To generate the life-cycle model all the testing activities are divided into five subdivisions. These categories are Planing & control, preparation, specifications, execution, and completion. The main idea of the development of a life-cycle is to carry out as many activities as possible in less time required. This is the reason why it is a demand to avoid time losses, and everything should be planned and prepared in advance. Consequently, through these categories, it is required a detailed explanation of the complete test plan to be executed and how to carry it out.

- **Infrastructure:** This section requires to clarify all the facilities required for effective and optimal testing execution. Inside it has to specify the testing environment, tools, automation strategies, and the workspace where it will carry out all the steps. The testing environment is composed of hardware, software, test database, simulations, and measurement equipment. The tools and automation strategies have the scope to facilitate the tester's work, and it exists a wide variety of tools to help in every phase of the process. The workspace intend the physical location where the test is supposed to be executed, but it also includes all the facilities and amenities that are required by the testers to work in a productive environment.
- **Techniques:** It intends to contain all the resources to enable the tester to complete the procedure. It means that it has to be introduced the techniques that are required to execute and control the progress of the test. These can be classical techniques and other innovations that have been developed for the specific test case. To be specified the strategies for development, test design, safety analysis, automation, and task management of the complete process.
- **Organization:** This part focused on the interaction between the people involved in the testing process. Although, it specified the different structures of the team, the roles, responsibilities, and how all of them should work together.

The main part of this thesis is developed inside the Infrastructure and Techniques cornerstones of the method. Facilitating the creation of automatic test and an user-friendly testing tool to carry out a database of autonomous test with the minimum tester effort.



3 | Background

Before starting with the procedure and the essence of the thesis, it is fundamental to review the principal theory required to understand the contents. This chapter covers the main ideas of Domain Specific Modeling, DLMS, and HDLC. These three topics are essential to achieve the goal of this thesis.

3.1. Domain Specific Modeling and Domain Specific Language

Domain-Specific Model (DSM) is, as described in the name, oriented to a specific domain rather than to a general field. It deals with automating software development but is specifically oriented to a small area. The more specified and narrowed down the field can be, the easier it gets to automate programming work that, otherwise, would be manually performed. A narrow domain and an implication area are usually found within a company and are case-specific [17].

A well-known problem in the area is the lack of connection between the formalism of the business concepts and the implementation of it when the software is developed with general-purpose tools (GPT). The reason for this is the differences, for example, the language and way of thinking, between experts in the two different cases. DSM provides a solution using a Domain Specific Language (DSL). A DSL uses domain-specific concepts and existing business rules [18].

Another perspective on the importance of DSL is in the construction of big software systems. The main contribution that DSL gives in this context is enabling different software artifacts to be reusable through different DSL. Examples of that are language grammar, source code, software designs, and domain abstractions. An important category for reuse is the application generators that often used the DSL as input language. This further enables the reuse of semantics from the DSL without requiring the performance of detailed domain analysis [20].

DSM is not something new, it has been used in many companies since the end of the 1990s. But DSM was mainly used by big companies, because of the complexity and high initial cost of implementation. Later, in the mid-2000s it caught more attention from research communities. That was mainly due to the development of different platforms and modeling environments that made it easier to implement DSM and reduced the cost of using DSM [18].

A narrow field

That DSM has a narrow focus means that its application is limited to what the developers intended it to be. A small and narrow focus is usually found inside a company, and the DSM solution is usually only for a part of the entire company domain. It is not commonly possible for companies to use the same DSM solution. The basic concept between the companies in the same domain are usually the same and those can be used, but the details differ, and that is important to provide automation. The more specified the focus area can be, the easier it is to create an effective language and automation.

Language sets the boundaries for what can and cannot be described. These limitations enable the narrow field, and it makes it possible the use generators that work with full code generation. Instead, if the domain is too broad, it is no longer possible to use the language for generating the needed code and artifacts. Therefore, it is important in the development to start with a small well-known and understood part and then extend it to cover the entire domain. At the start of the definition, it is usually easier to build it on the structure of an already existing or multiple languages.

Keeping a narrow focus allows the generators to provide efficiently and structured code. If sometimes it is needed to make changes in the code, the changes could be done by field experts [17].

Increasing level of abstraction

An increase in abstraction usually leads to a productivity increase, both in terms of time and resources needed for specifications and in terms of the maintenance work required. There are many ways to raise the abstraction, but most of them require that developers manually program and create a mapping using code. The DSM works by increasing the level of abstraction further than the currently used programming language. A key to this is to use a language that stands closer to the specific domain. This is done by defining a solution based on concepts that make part of the domain. The DSM thereby hides the complexity within and still guides the developers in the designs of the specific domain.

The language raises the abstraction using model elements that are taken from the domain rather than from the regular code world. In the DSM, not only text can be used but also other things like diagrams and matrices to enable it to become even closer to the domain.

3.1.1. Domain Specific Language

Sometimes Domain-specific languages (DSL) are instead called application-oriented, specialized, or task-specific languages among other notations. DSL is instead of being general, specific, and mapped to a limited domain. As the DSL provides notations and constructions mapped to a specific application domain it provides gains in expressiveness and utility in comparison to a GPL used in the same field. DSL also opens the application domain to a broader group of software developers; due to that, it reduces the required expertise regarding programming and the domain [20].

Usually, the specific notation established in the domain is not contained by a GPL. These notations should not be underestimated, and it is directly related to the increased productivity associated with the use of DSL. Also, suitable constructions and abstractions of the specific domain can not always, in a simple way, be mapped to functions and objects that can be placed in a library. The use of a DSL offers the possibility to perform things like analysis and verification by DSL constructs in an easier way than with a GPL. This is because the patterns of a GPL source code may be either too complex or not well defined.

Problem domain concepts

When developing the language, it is important to identify the relevant concepts. This is easier if you have previous experience in the domain and it is fundamental to consult people with higher experience because it is usually their insight and opinions that lay the bases for the creation of the DSM solution. The commonly used concepts are there for a reason, and people find them relevant and concise while talking about the product. If you start the development from an already existing vocabulary, it makes it easier cause people do not come up with solutions in coding terms, and they do not have to learn new concepts [17].

In the case of a DSM, the language itself identifies the concepts that are of relevance. Instead, in the case of a GPL, it leaves a big part of the decisions to the modeler, which makes things more difficult to deal with.

The domain concepts have several characteristics that make them suitable to be included in language development. The concepts are often already well known and used inside the development team. When a language is developed, it does not necessarily have to establish new concepts. Using problem domain concepts creates a natural connection to the problem domain, which makes the implementation of the model easier, and enables the reuse of elements. Mapping the model close to the domain also makes it easier to handle in many ways as reading and understanding. To make the language easy for people to learn and use it is better to keep it small.

Rules

The language must follow different rules and constraints linked to the domain. The rules give many benefits to the model and make it specific to the domain. One benefit is that it prevents early errors by making sure that models that are not desired cannot be created. It also guides the path towards desirable design patterns. Besides, the code generator requires a complete specification and it will be checked for and informed if parts are missing. The modeling work is also minimized hence the use of conventions and default values. Another benefit is that the specifications are consistent, that if an element is changed, it will be noticed elsewhere, and the model will have to be updated, or the inconsistency will have to be reported [17].

Many of the concepts used in the language also have rules which need to be recognized by the language as well. Some rules are identified from the sources of domain concepts, but others cannot be detected using the already existing material, then it is faster to ask the domain experts. The language's rules should work as nothing is permitted if it is not specified. That makes the language definition easier because new rules can be tested and implemented in steps.

In the beginning, it is not necessary to have all the rules defined, because both the model and the concepts can change. That is also the case for the notation and the generators. If they are implemented too early in the language, both time and effort can be wasted.

Changes over time

The language is not something fixed, and if the domain change, it should be possible to change. It will change and evolve, either because the domain has to change or because the modelers see new opportunities. No matter what change should be done, it is usually best to do it in a similar way to how it was developed. In the case of a large extension that will change many different parts of the model, it is good to make a pilot study before updating the language. Normally it is easy to add concepts to the language, and after the generator and the framework of the domain have been updated, the development can continue. Removing or modifying concepts that are existing is, on the other hand, more complicated. This is because the changes will have an impact on the already existing model [17].

3.1.2. Design for usage

Developing a DSL is not easy, it requires both expertise in the domain and language development, and very few people have both these criteria. The development techniques for a DSL are more diverse than the techniques to develop a GPL, which means that it requires consideration of all the present factors in the specific case. Furthermore, a lot of time is consumed developing the needed training material, support for the language, and creating the standards [20].

There are several ways that the design of a DSL can make it easier for experts in the specific domain that are unfamiliar with GPL. Generally, a DSL is needing to be changed more frequently than what is needed for a GPL. It is important to develop the language on parameterized building blocks to make it easy to change. The need of learning specific metalanguages for language development can also be reduced by the use of a user-friendly syntax, and the use of example sentences.

3.2. HDLC - DLMS/COSEM

3.2.1. DLMS/COSEM

The main purpose of a Device Language Message Service (DLMS) or the Companion Specification for Energy Metering (COSEM) is to standardize communication language for metering instruments and systems. The language is an object-oriented interface that gives the service of accessing the proprietary objects. It also specified a variety of communication profiles to enable message interaction through different communication media [15].

Thanks to the standardization of the COSEM interface classes, it provides a library of elements that can be used by manufacturers in the development process. It enables a huge variety while still securing the interoperability of the devices [14].

Today different protocol suits make interoperability between vendors possible, this means that the users are not locked into continuing with the previously used. The DLMS/-COSEM standard specifies an application layer for all metering devices. An advantage of the DLMS/COSEM standard is that it is supporting a variety of different communication technologies. DLMS is designed so that it is not dependent on any supporting layers. The DLMS specifies the data transfer procedure and the access to the information for the meters. It is also giving the possibility of interoperability between both the metering and the reading tools. And this, even if the models or vendors differ between the different devices [19].

Client and Server

The meter is controlled by the decisions of the COSEM application process (COSEM AP), and it works as a data server. Instead, the reading device is functioning as a data client. The reading device calls for a client application layer service to request something from the meter. The request that the meter can handle are data reading, data writing, and method calls. When the meter obtains a request, it processes it to decide with which implemented object it should interact. Then, the server application layer service is used to transmit the data or the results to the reading device.

DLMS/COSEM is creating an exchange of information between the meter and the data collection system. That is done using the major concepts of an open system interconnection (OSI). For the data exchange to be possible between the two systems it is of importance that the server and the client are both interoperable and interconnectable. Due to that the metering device and the collection system usually are on different devices the communication exchange of messages occurs through a protocol stack [15].

The use of COSEM for modeling metering enables to have semantic interoperability and hence a common understanding between the clients and servers even if different communication media are used. The semantic elements in the language consist of the COSEM objects, their logical name, the defined attributes, and methods.

Within DLMD/COSEM there are security mechanisms, that provide identification and authentication of both the clients and the servers. It also provides access rights to the object attributes within the COSEM and different methods in the application associations between the client and server.

3.2.2. COSEM objects

In Companion Specification for Energy Metering (COSEM) the meter is seen as a component in an advanced system for measurement and control. Therefore the meter must be capable to transport the measurement result between the measuring point and the point where it will be used. The meter also needs to be able to both give information to the consumer and handle consumption and local generation. To do that COSEM makes use of object modeling techniques to create the functions of the metering devices. The protocol lets the developers consider what functions will be needed and how they should be implemented. It does not specify either how the data should be transported. It, therefore, leaves a lot of freedom to the designer [14].

When you are defining a specific meter, it means that various specific objects are being chosen. The object consists of a group of attributes and methods, where attributes are the characteristics of the object. The information of the data contained by the attribute has an impact on the behavior of the object. The object can provide different methods to examine or modify the information in the attribute.

To refer to a specific object, the logical name specified by the Object Identification System (OBIS) code is used. The logical name is the first attribute of any object and should indicate what it is [19]. The OBIS is thereby an important part of the COSEM and is formed on DIN 43863-3:1997, Electricity Meters-Part 3: Tariff metering device as additional for electricity meters-EDIS-Energy Data Identification System. The existing OBIS codes have increased over time due to the new necessities of the market [14].

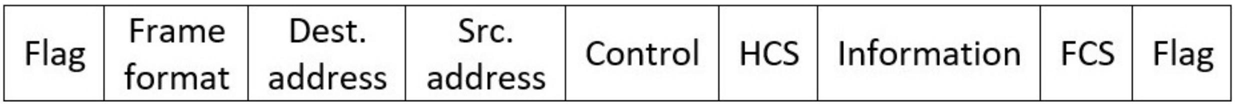
It exists many different interface classes and OBIS codes but it is not possible to implement them all on the meters. Therefore, the designers have to be selective in choosing which one to use to implement the needed objects on the meters [19].

3.2.3. Characteristics and benefits of using DLSM/COSEM

Using DLSM/COSEM for data exchange enables access to the metering devices for other parties such as clients and third parties. It also provides mechanisms that enable control of who can access the metering instrument and how each user can interact with it. Cryptography protection can be applied to both the DLMS messages and the COSEM data to guarantee the security and privacy of the usage. The usage of DLMS/COSEM also makes it possible to create a single access point, if more than one metering device is used [15].

Another important characteristic of the usage is that it provides efficiency and a low loss of resources because of the usage of mechanisms like selective access and compact encoding. The data exchange could occur either remotely or locally, and if the capability of the metering devices allows, it could occur at the same time without disturbing one another. Another characteristic of the usage of DLSM/COSEM is that different communication media can be used in different networks, such as local networks, neighborhood networks, and wide area networks.

3.2.4. High-Level Data Link Control (HDLC)



Flag	Frame format	Dest. address	Src. address	Control	HCS	Information	FCS	Flag

Figure 3.1: HDLC message structure

The metering system applies the High-Level Data Link Control (HDLC) frame format as communication protocol. In the structure presented in Figure 3.1, both the opening and the ending flag are 0x7e. The format type, frame segmentation, and frame length are expressed by the frame format. The destination address and the source address are instead different smart device MAC addresses. In the control position of the HDLC, the structure specified what type of message corresponded. The HCS and FCS sections are control checks corresponding to the header and the frame respectively. Finally, the information component is the DLMS frame containing the message [19].

4 | Development approach

Before start presenting the results of this project, it is necessary to look at the circumstances that lead to this project, the area where it takes place, and the initial considerations. This chapter generates an overview of the necessity, scope, and consideration taken to satisfy the requirements of the project.

4.1. Environment description

To start the procedure it is required to understand the specific task that should be solved and the environment in which it was executed. First of all, it is needed to remark that this project was developed during an internship period in a company that takes by the name MeteRSit. A company that produces and commercializes smart gas meters with innovative procedures in metering technologies and communication functionalities. All the steps for the release of a new product or the new features of the existing ones are made inside the department of Research and Development (R&D). After the planning, execution of the complete project is performed, and the potential final results arrive at the team of Integration and Validation (I&V).

The I&V team is in charge of executing all kinds of tests to verify the correct function of all the features implemented in the different products and to fulfill all the requirements established by law. When this project started, the tests were executed in a manual testing tool which required the complete supervision of a specialized tester engineer in charge all the time. It works mostly as a communication and interpretation tool. That means that to follow a specific test, the operator should introduce the commands required from the meter, wait for the response, analyze if it is the expected one, and then give the next command.

That testing method is time-consuming and wastes the capabilities of the specialized engineer. With the increase in demand and the insertion of the product in new markets, the company was unable to continue with the misuse of these resources. Consequently, it required a method to optimize and automatize the software testing procedures during the validation process.

This project intends to work specifically with the meter *Domusnext 2.0 G4 – G6 NB-IoT* (look in Figure 4.1) produced for the Italian market but with the opportunity to be implemented to work with meter with similar communication protocols.

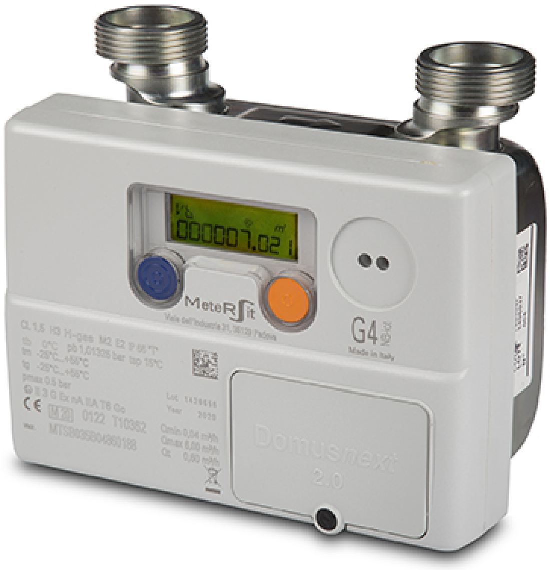


Figure 4.1: Gas meter Domusnext 2.0.

In figure 4.2 is presented the class diagram of the complete testing system. In which it can be remarked the main classes that interact in the complete process. The main classes are the gas meter, which contains the COSEM objects, and the test case that triggers all the interaction with the meter. Between these two main classes is located the commands block, that act as a communication bridge for all the interaction and gets to send all the messages to the gas meter and back in the correct format.

The COSEM objects conform to the structure of the gas meter, and they are the access to the data storage in the device. The final block presented in the diagram is the Profile which is an attribute of the gas meter and it can be public management or installer. This profile categorizes the rights obtained in each connection. All the methods and attributes contained are intended to be represented and accessible with the solution presented in this thesis.

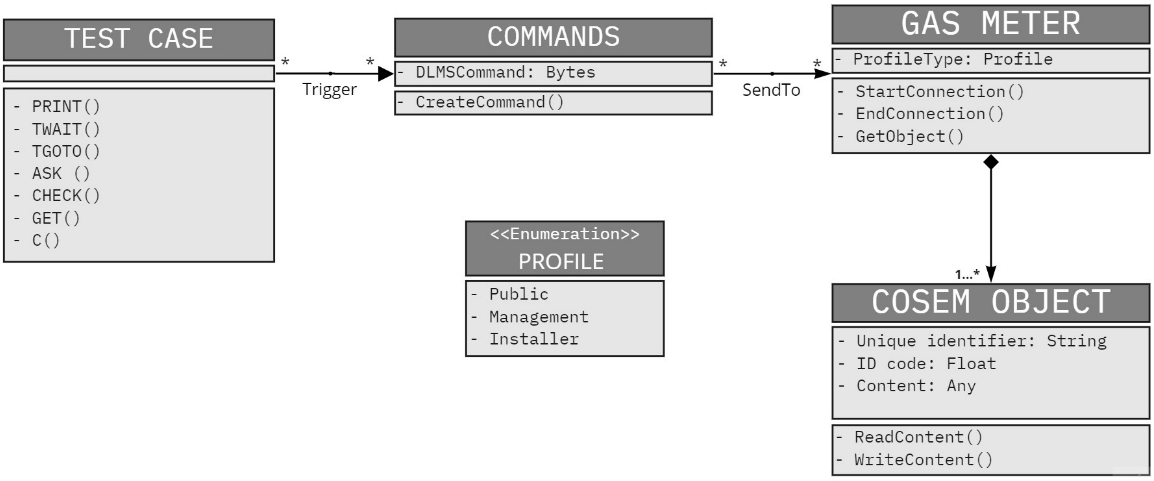


Figure 4.2: Tests class diagram.

4.2. Specifications of the desired result

Taking into account the level of specialization desired, and the specific characteristics of the device to work with, it came out that the best way to solve the necessity was developing a brand-new specialized testing tool. The tool intends to release the tester's responsibilities regarding following the complete process and the analysis of the results. The development of this project is looking to reduce the working charge on the testing engineers, saving time and resources.

It is expected, an instrument to maintain stable communication with the gas meters through the optical port. Be able to read and modify the information stored in the memory of the device with real-time automatic interpretation. Run a variety of tests previously specified with settable parameters. Save the track of the interaction with the meter of a log file for further analysis. And finally, generate a final report of the executed tests to evaluate stability during the time.

It is required to have a database with the different tests that should be possible to modify, delete and create new ones. The tests are planned to be recreated in scripts to be executed on multiple occasions under the same circumstances. Additionally, it is intended to be possible for people without extensive programming knowledge to create and interpret the scripts that reproduce the test cases.

As an additional objective, it was set desired to reach compatibility with different devices that share the same communication protocols. As was mentioned before, this project is designed to work with the meter *Domusnext 2.0 G4 – G6 NB-IoT* but a variety of devices share the DLMS/COSEM protocol (see Chapter 3). Consequently, The tool should be optimized to work with the initially considered technologies, but it may be possible to be compatible to communicate with other devices' references.

4.3. Workflow

To conclude the project design, it was tried to generate a clear idea of the link between the tester, the tool, and the meter. A high-level interaction representation can be observed graphically in Figure 4.3.

Starting from the user, the one who starts the testing process. It required a user-friendly graphic interface that is a simple way to introduce all the options to interact with the device to be tested. This tool presents all the methods and the different profiles to connect with the meter, all the objects to be read, interpreted, and modified, plus the way to trigger all the tests previously designed. Therefore, the testing tool works as the central connection between the tester and the meter and offers access to the test database.

Continue with the workflow, from the central interface there are sent the commands corresponding to the user actions through a serial port connected to the metering device. The communication between the computer and the meter makes use of a probe connected to the optical port implemented in the meter. In the same way, all the responses to the user's request are sent back to the tool. The messages received from the meter through the serial port are decoded and interpreted by the tool to be shown on the computer screen for the tester's further analysis.

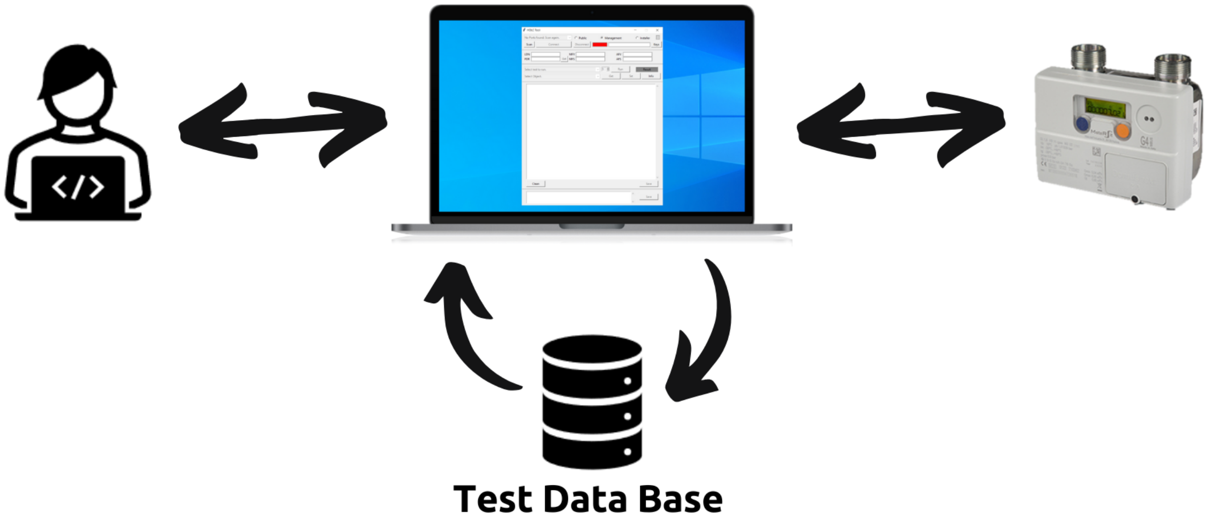


Figure 4.3: High level user meter interaction.

Finally, the last block in this graphic is the one that corresponds to the database. This database corresponds to a dedicated folder where are collected all the scripts to be available inside the tool. The mentioned scripts represent the functionality tests written in the Domain Specific language design for the testing procedure. The testing tool interprets the scripts and generates the corresponding commands to proceed with the testing process. At the end of the execution of all the steps specified in the script, the tool generates a final result depending on the partial results of all the check controls included inside the scripts.

A high level description of the testing process would start with the user, who uses the tool to trigger the communication with the meter. In the next step, the user selects the test to be executed and makes it start. Now is the tool's responsibility to take the test script and interpret line by line the instructions contained. Each instruction will represent a command to be sent to the meter and a response to be received. To conclude, the tool generates a preliminary test result based on all the evaluations included in the test description.

The basic interactions with the meter are usually of two types. The GET is the first one in which the tool requests the information contained in a specific COSEM object. Instead, the SET consists in modifying the content. Combining those it is possible to obtain the main components of a test. All tests are based on request and responses procedure. It is common to verify different features request the content of an object, then try to modify it for a feasible or unfeasible value, and finally, reading again to verify if the action was executed or rejected properly.

4.4. Engineering decisions and considerations

During the planning phase of the developing process, some considerations were made. In agreement with the specialist, the initial information to be requested as soon as the tool is launched was chosen.

During the discussion about the test scripts, it was decided to implement a domain-specific language. Looking for the flexibility, simplicity of the scripts, and facility to read given by the DSL. In addition, it allows specialists unfamiliar with programming languages to create and modify the scripts. The following sessions will show how the DSL was implemented to work together with the tool.

Additionally, it was decided to implement the creation of two types of files after the end of each testing session. The first is a text file (.txt) that saves all the interactions between the tool and the meter. The second one is an excel file (.xlsx) that works for stability analysis of the tests executed.

The next chapters approach how all blocks of the Figure 4.3 were implemented to achieve the desired functionalities.

5 | Domain Specific Language for gas meter testing

This chapter introduces the DSL designed for the specific case of gas meter testing. The motivation for the development, the main components, and the way to implement it. At the end of this chapter, it is presented and explained two different test scripts that apply this language and are currently implemented and tested in the field. Before starting, it is important to remark that the DSL exposed in the following section is the first version, this testing language is still growing to cover a wider range of tests and the complete domain.

After reviewing all the specifications required by the company to have a specialized tool for the testing division, it was noticed that the level of specification was hard to meet with a generic language. This problem was introduced due to the petition of creating something with which people without extensive programming language can interact. The capability to create, understand, and modify are essential actions that have to be provided by the language. The capability to adapt to new requirements and norms is what makes this language so powerful.

5.1. Domain

Inside the Gas meter industry, there are high standards to satisfy. Plus all the client requirements there are many legislation and regulations to follow. For these reasons, companies are forced to spend a lot of time and resources on the testing of new features and products. Usually, the testing team designs a variety of tests to cover all the functionalities to be tested in the products. This process creates tests that work as route paths for the testers to repeat multiple times to arrive at consistent results. In the end, this process concludes in repetitive tasks that consist in following a test plan multiple times and evaluating the results.

The Domain Specific Language introduced in this chapter is looking to offer a suitable way to automatize the repeatable steps during the testing procedures. With the idea to generate scripts that can execute every step designed for testing a product features. Flexible scripts that along with the tester can recreate all the repeatable procedures that are currently made during the testing process. Additionally, it is fundamental that the generated scripts are easy to interpret and reproduce by specialists without extensive coding knowledge.

During the design process of a new Domain Specific Language, it is important to clearly define the domain where is intended to work and to what category of people or specialists is oriented. In this aspect, this testing DSL is specifically designed to work with the testing tool also designed during this thesis and presented in Chapter 6. Therefore, as part of its work, the testing tool interprets the scripts created with the DSL. Finally, the target users are categorized as testing specialists in the gas meter industry.

5.2. Concrete syntax

The main idea during the development of the DSL was to create a specialized language with enough flexibility to create scripts that automatize a huge variety and complexity level of tests applicable to this industry. This language should be in a simple syntax with the known words and the main terms which are familiar to the specialists. Accordingly, the language is based on keywords easily understandable for all users, even for those without any programming language or expertise. This property makes the complete scripts easy to read and understand, with only a basic introduction to the language plus the background knowledge in the field.

During the analysis of a representative selection of the test, it was shown that the number of actions to execute is not too many, but the alterations of those actions result in a wide variety of different results. In this sense, it was tried to reproduce this behavior in the architecture of the language, using a few functional keywords that modify their behavior depending on the parameters set. With the interaction between the keywords and the special words, it is possible to achieve flexibility and reach to recreate a huge variety of tests. A basic explanation of each keyword is given in the following paragraphs.

- **PRINT:** Is used to show a message on the main screen of the tool, which helps to make it easier to understand the procedure for the user. It may also be used to present the progress or notes in order to make the results more understandable.
- **ASK:** Is designed as a way to ask the tester for parameters when they are required. Following these keywords are accepted as many parameters that are needed, this

parameter is going to be assigned a position identification to call the user inputs on other keywords. When the tool finds this keyword in a script it automatically generated a Pop-up window with as many input boxes as the parameters specified.

- **C:** Is the main identification to send DLMS commands to the meter. The DLMS command is a series of bytes that generates an interaction with the objects. This keyword accepts as input a DLMS command that can contain the parameters set previously by the user.
- **TWAIT:** It generates gaps of wait that are required in some tests. This one gets input in the format DD.HH.MM.SS, referring to the number of days, hours, minutes, and seconds respectively.
- **TGOTO:** This one is another time control keyword. This one receives a specific date and time as input, and it pauses the test execution until the specified time arrives. The input goes in the format of YYYY/MM/DD HH:MM respective to year, month, day, hour, and minutes.
- **CHECK:** It is created to generate partial evaluations. The result of this action is always going to be Success or Fail. The inputs to these keywords are indications to execute validation concerning the last response received. This Keyword works along with others like *SET* and *VALUE*, the first indicates the validation of a set executed previously, and the second validates a mathematical operation with the values of the last response.
- **GET:** This keyword is used to show the user the information contained by an object clearly. For this keyword, there are additional identifications names for each object which specify what information is intended to be read. As a result, the tool shows the complete interpretation of the object as it is stated in the specifications. All the objects previously mentioned are specified and described in the technical specification regulating the interchangeability of the metering devices [9].

In the table 5.1 it is introduced an example of each keyword previously presented.

Key words	**Example**
PRINT {Text}	PRINT Message to be written in the text box
ASK {Parameters}	ASK Param_1 Param_2 Param_3
C {DLMS}	C C00141000800000100000FF0200
TWAIT {Time}	TWAIT 03.02.05.15
TGOTO {Date}	TGOTO 2022/07/22 17:30
CHECK {Action}	CHECK VALUE !00
GET {Object}	GET UNIX

Table 5.1: DSL key words

All the mentioned keywords can be combined and placed in many different combinations. Each one of these combinations results in a different routine, which in our domain are considered tests. Apart from the parameter to be chosen by the test programmer, there is a variety of additional words that give extra value to the main commands. Additionally, it was designed with a specific word to refer to each one of the objects implemented in the meters (detailed explanation in [9]). These object references work with **GET** to interpret the response and deliver the information to the user as clearly as possible.

Two example of the object interpretations are presented in the Listing 5.2. These results are the product of the combination of the keyword **GET** with the object identifiers *SYNCHRONIZATION_REGISTERS* and *NB_IOT_PARAMETERS*. The objects identifiers correspond to the objects *Synchronization Registers* and *NB-IoT Current Parameters* respectively (see [9]). The actual response of the request of these objects are *C4 01 41 00 02 03 06 00 00 00 09 06 00 00 06 F4 06 00 00 1E 8A* and *C4 01 41 00 02 05 11 14 06 00 00 56 B9 0A 08 4E 42 2D 49 6F 54 20 20 12 00 00 12 00 20* but after the analysis executed automatically by the tool the user received the information in the way presented in the Listing 5.2.

```

5.4.5.3.3 - Synchronization Registers
    SYNCHRONIZATION REGISTERS
         Number of synchronizations:    9
         Increased seconds:             0
         Decreased seconds:             244


5.4.16.30 NB-IoT Current Parameters
    NB_IoT Current parameters:
       Band:                     20
```

```
           PLMN:                 22201
           Access technology:    NB-IoT
           RRC (sec):            0
           T_3324 (sec):         32
```

Listing 5.1: Examples objects interpretation.

5.3. Implementation

All scripts should be structured with the template presented in the Listing 5.3. It consists of an initial optional section to request parameters if they are needed from the user. Then multiple command blocks, as many as needed, for all the interaction with the metering device. This command block is conformed by one or many instructions generated with the Keywords described before, plus a partial check to close the block.

```
< Begin script >

    < Parameters request > (Optional)

    < Begin commands block 1 >
        < Command 1 >
        ...
        < Command i >
        < Partial check >
    < End commands block 1 >
    < Commands block 2>
    < Commands block 3>
    ...
    < Commands block N>

< End script>
```

Listing 5.2: Scripts template

After the tool executes all command blocks, it is presented a final result that combines all partial checks. For a better understanding, and to clearly explain the implementation of the testing language, a real example is presented and will be explained in detail in Listing 5.3.

```
    PRINT TEST Installer/maintainer setup


// Get parameters
    ASK Permissions(1B) Schedule_time(Unix)(4B) Preset_duration(2B)
```

```
6
7  // Set "installer setup"                0-0:94.39.30.255 att.2
8      PRINT -- Setting the Installer window.
9      C C10141000100005E271EFF0200020311 (1) 06 (2) 12 (3)
10     C C00141000100005E271EFF0200
11     CHECK SET (1) 06 (2) 12 (3)
12
13 // Wait to test before the Schedule_time arrive.
14     TWAIT 00.00.00.10
15
16 // Get "installer remaining time"       0-0:94.39.31.255 att.2
17     PRINT -- Test that the window is not started
18     C C00141000300005E271FFF0200
19     CHECK VALUE h(3)
20
21 // Wait 10 mins after the Scheduled_time set
22     TGOTO (2)
23     TWAIT 00.00.10.00
24
25 // Get "installer remaining time"       0-0:94.39.31.255 att.2
26     PRINT -- Test that the window already start
27     C C00141000300005E271FFF0200
28     CHECK VALUE !00000000
29
30 // Wait to pass the Preset_duration set
31     TWAIT h(3)
32
33 // Get "installer remaining time"       0-0:94.39.31.255 att.2
34     PRINT -- Test that the window have been closed after
35         the Preset_duration
36     C C00141000300005E271FFF0200
37     CHECK VALUE 00000000
```

Listing 5.3: Installer/Maintainer setup script.

This one was the first test recreated and automatized using the DLS developed. It consists of validating the correct behavior of a time window configuration, during this window the meter accepts connections with the installer/maintainer profile. For a better understanding of the script, general comments were included in each block of actions. The two slashes (//) are the identifier of the comment lines. The rest of the lines start with a singular word, in capital letters, that are the mentioned Keywords.

The first one presented in Line 1, is a simple message included in the log file for a better understating of the current state of the test. Then in Line 5, **ASK** requests all the parameters required to set the installer/maintainer window to the user in a pop-up window like the one shown in Figure 5.1. In this case, they require three-parameter, *Permissions, Schedule_ time, Present_ duration.* The first one represents the rights that will obtain the user connected with the installer/maintainer profile, it is represented by 1 byte as it is specified by the normative. The second one is the date and time in which this mentioned window is going to start. This one is 4 bytes that represent the Unix time in hexadecimal. The third parameter is the duration of the activation of the window. That means starting from the *Schedule_ time*, for how long will be possible to connect with the meter through the installer profile. All these parameters are going to be called during the complete script with an identification respected to the position in which they were asked.

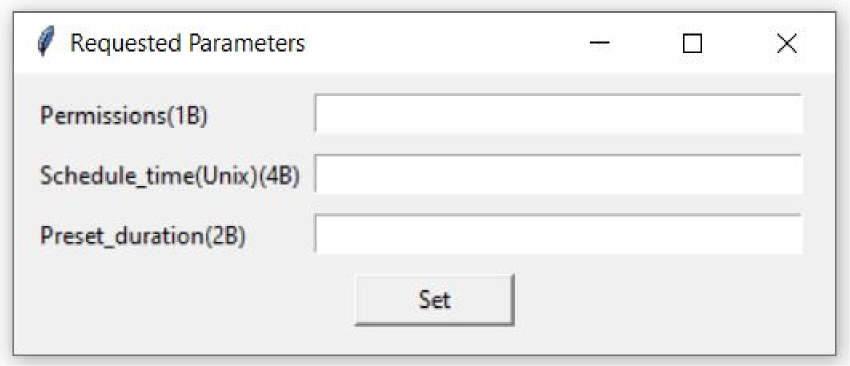


Figure 5.1: Parameters requested Pop-Up window

After that, it arrives at another block of commands, the first one, line 9, is the set DLMS command for the object *Installer setup*. It is possible to notice that the command includes the identifiers (1), (2), and (3) that represent the three parameters requested respectively, identified by their positions. These commands are composed of a header, an identification class, an identification object, and an attribute. for a better understanding lets analyse the command in line 10, *C0 01 41 00 01 00 00 5E 27 1E FF 02 00.* The three first bytes are the header where is set the type of message that is being sent, *C0* means that it is sending a Get, instead *C1* would be a Set. *00* says that is a normal request, other options would be a request in lists or blocks. *41* is a programmer's choice to represent the priority of the request, for the convention is used *41* or *C1* when there are no special requests. After that, the two following bytes represent the class identifier that points to a specific group of objects in which the object of interest is contained, in this case, *00 01.* Then it comes to the OBIS which are six bytes that represent the object required, *00 00 5E 27 1E FF* is the hexadecimal of 0-0:94.39.30.255 which is the OBIS of the *Installer setup* object. The next byte *02* is the attribute inside the object. Finally, the last byte *00* is used in case it is going to be included a special requirement, for example, a specific range or entry. For further explanation of the DLMS commands it can be referred to the Blue Book [14].

After the set is executed in line 9, the object is read in line 10 and verified that the values saved are the same as the values set. Along with the **CHECK** keyword, the word SET is used to verify the correct set of parameters.

For this test to be successful it has to control the behavior of the access window before, during, and after it is open. Before the first check, 10 seconds are left to pass after the control set, Line 14. Then it is verified by calling another object called *Installer remaining time*, this object stores how much time is left for the connection window, in seconds. In Line 19 we are proving that this value is equal to the parameter that specifies the duration of the window, as the window has not begun it should not have decreased any second.

In line 22, the keyword **TGOTO** is used to put the test on pause until the time to start the window arrives. In this case, parameter number two set the starting time, so the tool is going to wait until the meter clock is equal to the time set. Then it waits ten more minutes to make the next check inside the time frame of the window.

Finally, there is a wait for the full-time set in parameter number 3, and then the final check is performed. At this moment, it is verified that the *Installer remaining time* value is equal to 0 due to the window being already closed. With this last control the test is completed, and the final result will be computed automatically by the tool taking into account all the partial results.

Now it will be presented a second example, this one to show some variety and how the language can be adapted to the different tests. The test script introduced in the Listing 5.3 validates the correct functionality of the closure of the valve and the billing period.

```
    PRINT TEST Valve and billing period closure


// Get "Disconnect control"                     0-0:96.3.10.255 att. 2
    PRINT -- Reading stated of the valve and checking that it is open.
    C C00141004600006003OAFF0200
    GET DISCONNECT_CONTROL 2
    C C00141004600006003OAFF0200
    CHECK VALUE 01

// Get "Billing/snapshot period counter"        7-0:0.1.0.255 att. 2
    PRINT -- Saving billing period counter
    C C001410001070000010OFF0200
    GET SAVE(1) BILLING_COUNTER
```

```
15
16 // Set "Disconnect control action schedule"     0-0:15.0.1.255 att. 2
17     PRINT -- Configuring script for valve closure
18     C C10141001600000F0001FF020002020906000000A006AFF120003
19     C C0014100160000F0001FF0200
20     CHECK SET 03
21
22 // Set "Disconnect control action schedule"     0-0:15.0.1.255 att. 4
23     PRINT -- Setting an execution time in the past for immediately
24         valve closure
25     C C10141001600000F0001FF0400010102020904110000000090507E1090705
26
27 // Wait for valve change of state
28     TWAIT 00.00.00.15
29
30 // Get "Disconnect control"                     0-0:96.3.10.255 att. 2
31     PRINT -- Reading stated of the valve and checking if it is close.
32     C C0014100460000060030AFF0200
33     GET DISCONNECT_CONTROL 2
34     C C0014100460000060030AFF0200
35     CHECK VALUE 00
36
37 // Get "Billing/snapshot period counter"         7-0:0.1.0.255 att. 2
38     C C001410001070000100FF0200
39     GET SAVE(1) BILLING_COUNTER
40     CHECK VALUE SAVED[2] > SAVED[1]
```

Listing 5.4: Valve and billing time closure script

This script follows the standard structure described before, but in this case, it was not required any parameters in the context of the test. In the first block of commands, it checks the initial state of the valve, proving that it is open when the test starts. The keyword **GET** is used to clearly present the result of the object requested in the line before with the object identifier to the user. In this case, thanks to the identifier *DISCONNECT_ CONTROL 2*, the tool knows how to interpret the response.

Then in Line 11, start the second block, in which it is analyzing the value of the object *Billing/snapshot period counter* the one containing the number of billing periods already closed in the meter memory. In the GET command, it was used the special word SAVE to store a value for further analysis after the test. Inside the parenthesis it is specified the number of bytes that wants to be saved, they are taken for the last bytes of the previous response.

In the next block of commands, starting in Line 16, attribute two of the object *Disconnect control action schedule* is set. In this attribute, it is specified the action to be executed at the moment to trigger this action. The modalities can be a closure of the valve, opening of the valve, or closure of the valve and billing time together. As it is being tested for this last modality, we set the last parameter of the object equal to *03*.

Then it is trying to force the execution, setting a scheduling time in the past to generate an immediate trigger of the modality previously set. In line 28, there includes a 15 seconds delay to wait for the mechanical change of state of the valve. Then it controls the state of the valve again, this time checking if it is closed.

Finally, the last block reads the *Billing/snapshot period counter* and saves the final value one more time. A last partial result is generated due to the value check of the two values that were saved during the test, verifying that the second one is greater than the first one. That means that whit the closure of the valve it also closed a billing time as it established the modality set in Line 18.

These examples intend to show the keywords and how they can be structured to recreate a real test applied in the gas meter industry. The result of both tests presented before is contained in Appendix A and B respectively. Combining all the keywords, and the complements implemented it is possible to recreate a wide variety of tests in the field. In this way, a specialist can create new scripts for each test and store them in a database to be reused whenever it is required. The database mentioned is a dedicated folder inside the tool in which all the tests are stored for future executions. At this stage, the language is composed of 7 keywords, and more than 180 special words between the ones that modify the functionality of the keywords and the DLMS object identifiers. For the integration of the language in the testing processes, all the specifications and explanations were compiled in a guideline file shared inside the company.

In chapter 7 it is shown how the DLS language was implemented to be interpreted by the tool and how it is finally applied to test a meter with some real cases and results.

6 | Testing Tool

After all considerations, and the planning session the development proceeds with the constant support of different specialists. All this support was essential in the creation process to obtain as result a truly functional tool to be included in the work path without huge resistance to change. After some tries and different tests, there were made some additional considerations and decisions looking for a better way to approach the final steps of the development of the software presented ahead. The following sections present the tool obtained with this thesis.

6.1. Tool structure

In the initial decisions for the development of this tool, Python was chosen as the programming language [4]. Considering that it is an open-source language, the flexibility and the simplicity of its implementation. Also, it was considering the capability to create simple and interactive graphic interfaces, making use of the library Tkinter [5]. Other libraries that were essential for the development process of this tool were Pyserial [3], Pycryptodome [2], and Multiprocessing [1].

The structure of the back-end of this software was divided into three processes running in parallel. This enables to have different operations to occur at the same time without interruption, it also enables running the processes in separate cores of the computer generating a faster response and increasing the efficiency of the tool. The first process is in charge of the GUI and the complete interaction with the tester, the second one takes all the internal computational processes like encryption, creation of messages, and interpretation of scripts, and the last one control all the communication with the meter, sending and receiving all messages.

The first process, starting from left to right in the diagram presented in Figure 6.1, is the graphic interface of the tool designed mainly with Tkinter, this part will be analyzed deeper in the following section. The second process is the main one, in which all computation and the interpretation of the different scripts are addressed. Additionally, this

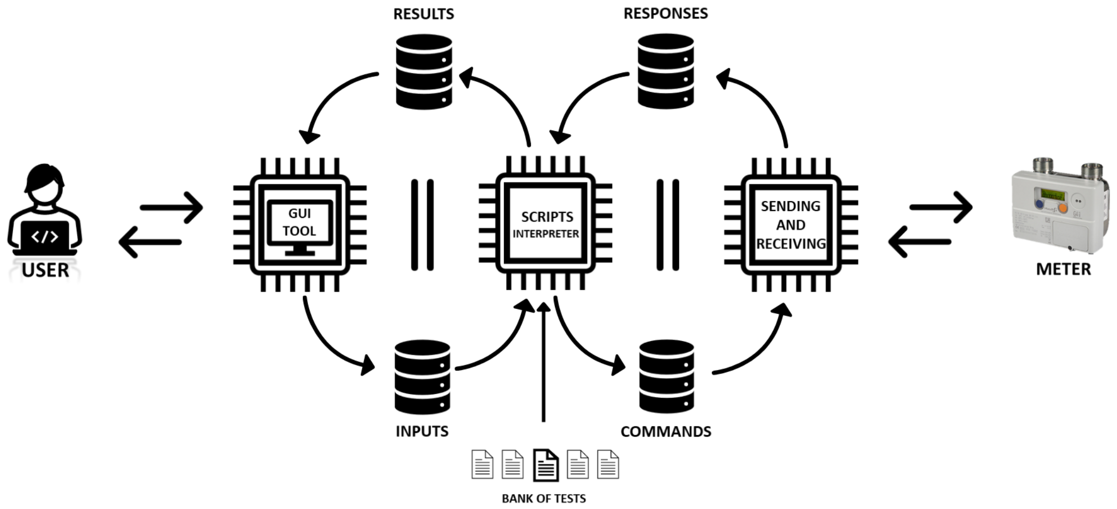


Figure 6.1: Backend tool structure

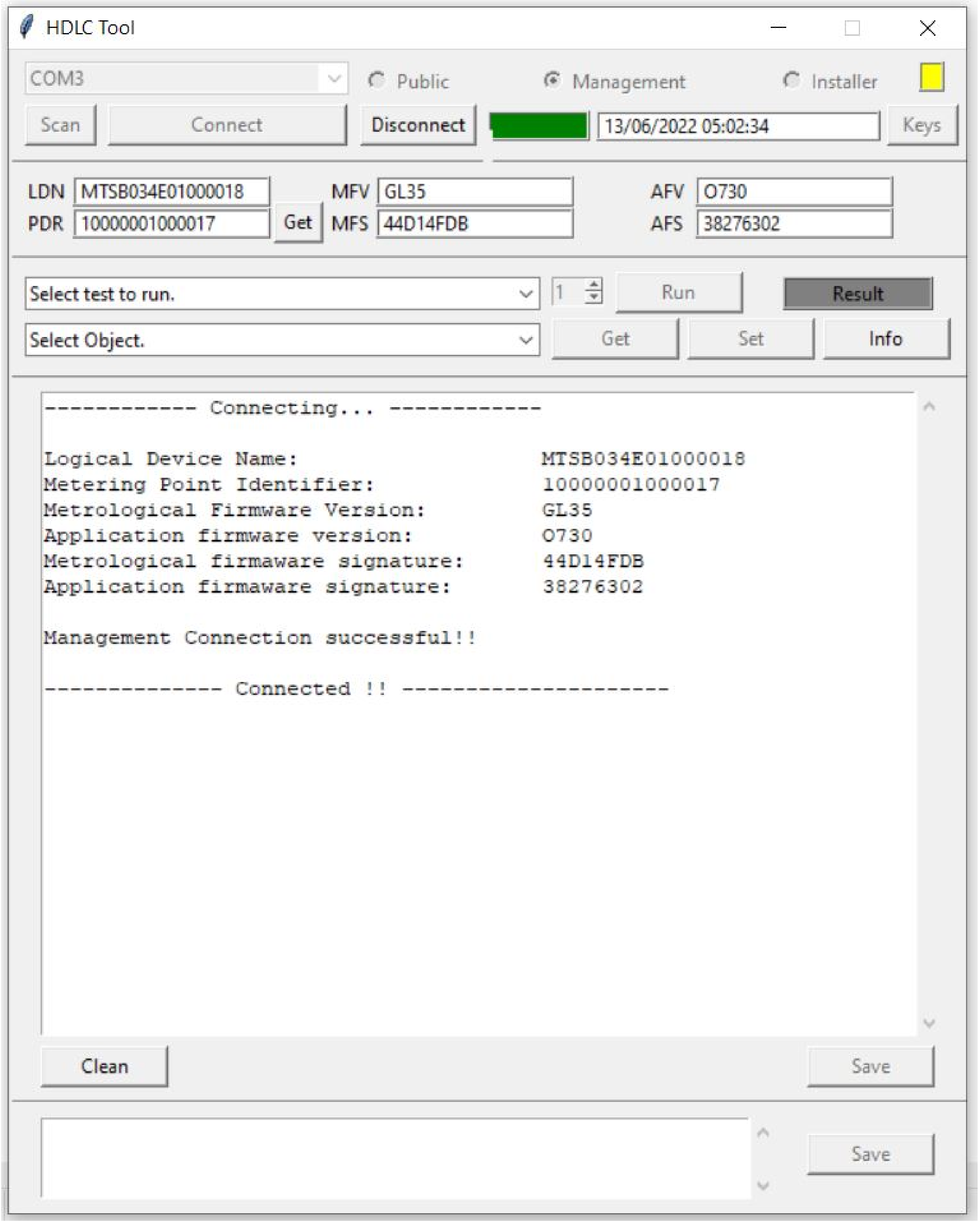
process is in charge of the encryption and decryption of all messages, consequently, the library Pycryptodomein is fundamental in this process. The last one is mainly working for the communication through the serial port, sending and receiving all messages with Pyserial, and includes some control methods to avoid as many as possible communication problems.

These three processes are completely independent of each other, they do not share the same memory. Consequently, it was required to apply a set of queues to address the data transference. That is also helpful to reduce information loss during data transfer because every single process takes the information for its input queue and put the result in its output. During data processing, the other entries wait in the queue until they can proceed for analysis.

The last item of the diagram in Figure 6.1 is the bank of tests, also called the test database. That is a collection of scripts created with the Domain Specific Language (see Chapter 5). After the script creation, it should be included in a specific folder where the tool looks for the available tests. The tool takes the script from the database, parsed it, analyses it, and automatically executes the respective test.

6.2. Tool review

The resultant testing tool is presented in Figure 6.2, which is one main screen divided into different sessions concerning the functionalities. This section will take each part and explain its content and functionalities.



COM3

○ Public ● Management ○ Installer

13/06/2022 05:02:34

LDN MTSB034E01000018

PDR 10000001000017

MFV GL35

MFS 44D14FDB

AFV 0730

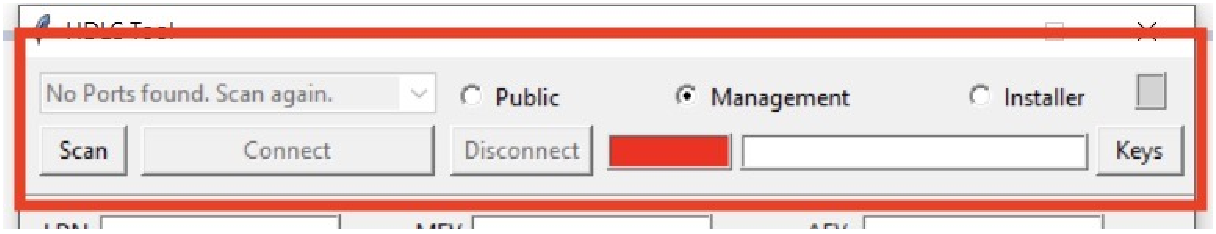
AFS 38276302

1

```
------------ Connecting... ------------

Logical Device Name:                    MTSB034E01000018
Metering Point Identifier:              10000001000017
Metrological Firmware Version:          GL35
Application firmware version:           0730
Metrological firmaware signature:       44D14FDB
Application firmaware signature:        38276302

Management Connection successful!!

-------------- Connected !! ---------------------
```

Figure 6.2: Main screen HDLC tool

In the connection session (shown in Figure 6.3), it contains all the options to configure the connection desired. The first option is the selection of the serial port through a drop-down menu generated after the scan of the serial ports available that satisfied the specification by a connection with the optical port. On the right, it is found the profile selector. The profiles implemented are public, management, and installer profiles. These profiles are the different users with which it is possible to associate with the devices, each one of those has a different connection procedure and different permissions to interact with the meter. Additionally, the association with management or installer profiles includes encryption communication, while in public everything is clear. Finally, the last button in the bottom right corner aloud to manage the system keys.

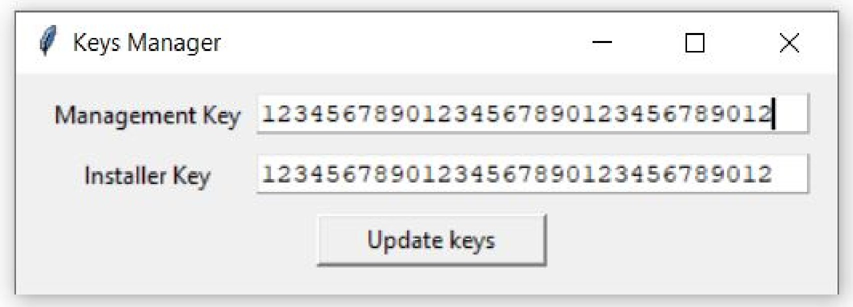


No Ports found. Scan again.

○ Public ● Management ○ Installer

Figure 6.3: Connection session tool HDLC.

The Keys button generates a pop-up window, like the one we can see in Figure 6.4, where is shown the actual set of keys to access and encrypt the communication with each profile. As well, it is possible to modify the present keys and save them for future communication sessions.



Keys Manager

Management Key 12345678901234567890123456789012

Installer Key 12345678901234567890123456789012

Figure 6.4: Keys Pop-Up window.

The second session on the main screen contains crucial information to characterize the device. Also, it is shown the software version running in the meter. It included a dedicated button to request the PDR (Metering Point Identifier) at any moment while the communication is active by a request of a testing specialist.

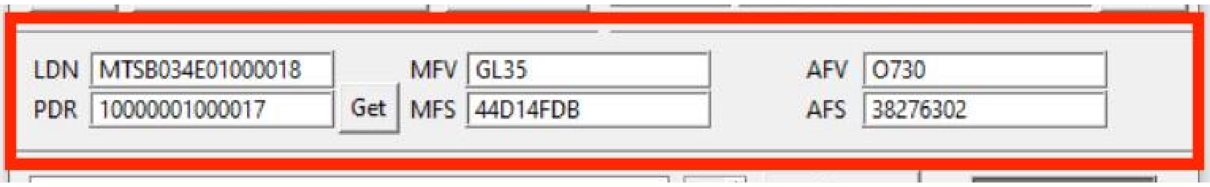

LDN MTSB034E01000018
PDR 10000001000017
MFV GL35
MFS 44D14FDB
AFV 0730
AFS 38276302

Figure 6.5: Information session tool HDLC.

The next session, shown in Figure 6.6, is characterized by two main drop-down menus. The first one is used to choose the test desired to run from the test scripts database. At the moment to start the tool, it is making an automatic search in the folder containing all test scripts to be presented in this drop-down menu. On the right there is the option to select the number of times that a test wants to be executed consecutively, limited to 99, to evaluate the resulting stability for a single test. The tests can be run from one to a hundred times without requiring human interaction. And the last item in that row is a label that shows the result (PASS or FAIL) of the last test concluded.

In the second row, the first thing found is another drop-down menu. Where it is possible to choose between all the objects implemented in the meter from the UNI/TS 11291-12-2 [9]. It is possible to read or modify the selected object from the list. The last button on the right redirects the user to the normative, to get specific information about the different objects.

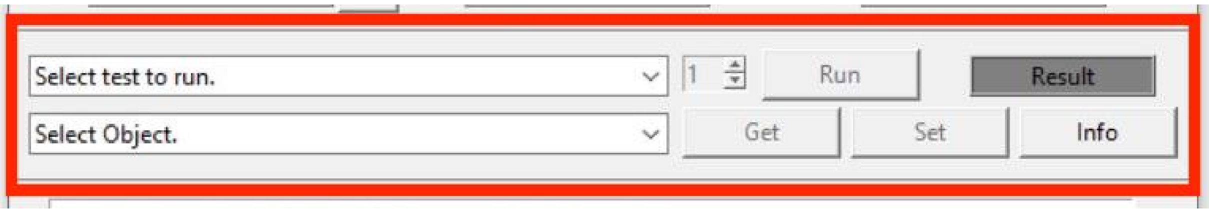
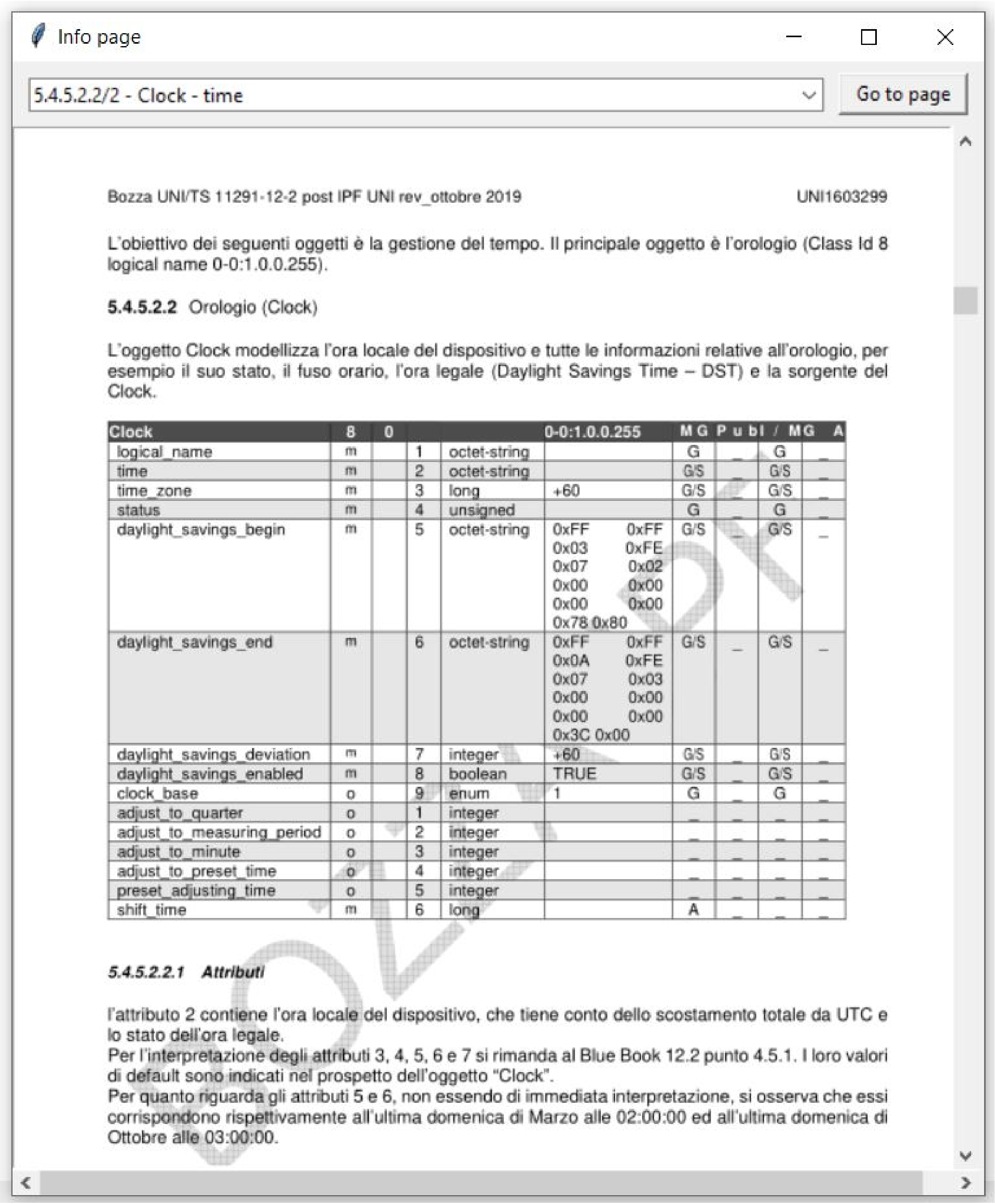


Figure 6.6: Control session tool HDLC.

After pressing the info button, it is generated a new pop-up window with the normative (see Figure 6.7). It also gives the option for a quick search to have another drop-down menu to choose the object required and take the user to a specific page.

The following session in the main window is dedicated to showing the current interaction with the meter (look at Figure 6.8). This one is the main information window in the tool, it is designed to deliver to the user all the information requested. It helps the following of the communication session. All the command sequences executed during a test, there are also shown in this box to offer the opportunity for further analysis in case it is required. At the end of each test, it explicitly presents the general result as an analysis of all singles



Bozza UNI/TS 11291-12-2 post IPF UNI rev_ottobre 2019 UNI1603299

L'obiettivo dei seguenti oggetti è la gestione del tempo. Il principale oggetto è l'orologio (Class Id 8 logical name 0-0:1.0.0.255).

5.4.5.2.2 Orologio (Clock)

L'oggetto Clock modellizza l'ora locale del dispositivo e tutte le informazioni relative all'orologio, per esempio il suo stato, il fuso orario, l'ora legale (Daylight Savings Time – DST) e la sorgente del Clock.

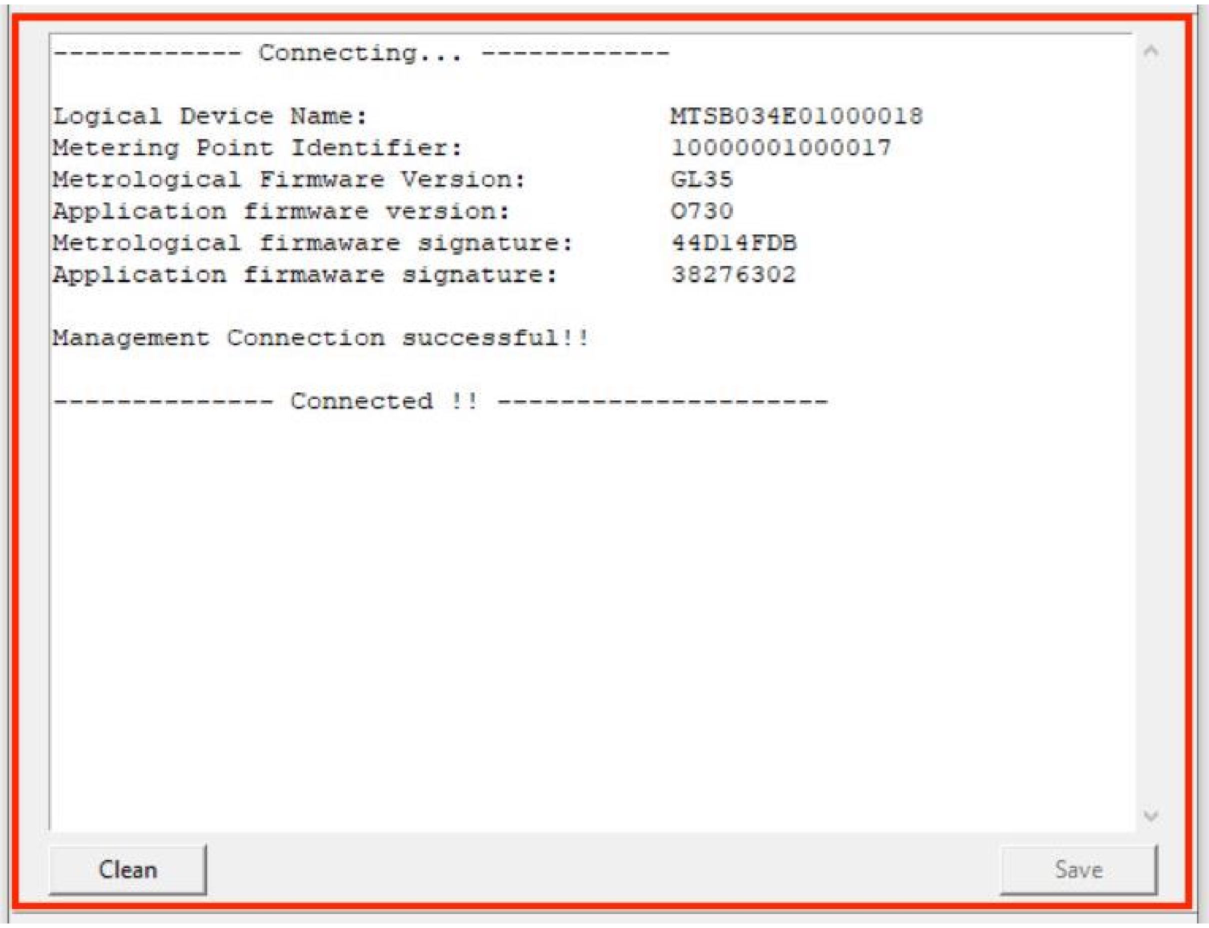
Clock	8	0			0-0:1.0.0.255	M G	P u b l	/ M G	A
logical_name	m		1	octet-string		G	_	G	_
time	m		2	octet-string		G/S	_	G/S	_
time_zone	m		3	long	+60	G/S	_	G/S	_
status	m		4	unsigned		G	_	G	_
daylight_savings_begin	m		5	octet-string	0xFF 0xFF 0x03 0xFE 0x07 0x02 0x00 0x00 0x00 0x00 0x78 0x80	G/S	_	G/S	_
daylight_savings_end	m		6	octet-string	0xFF 0xFF 0x0A 0xFE 0x07 0x03 0x00 0x00 0x00 0x00 0x3C 0x00	G/S	_	G/S	_
daylight_savings_deviation	m		7	integer	+60	G/S	_	G/S	_
daylight_savings_enabled	m		8	boolean	TRUE	G/S	_	G/S	_
clock_base	o		9	enum	1	G	_	G	_
adjust_to_quarter	o		1	integer		_	_	_	_
adjust_to_measuring_period	o		2	integer		_	_	_	_
adjust_to_minute	o		3	integer		_	_	_	_
adjust_to_preset_time	o		4	integer		_	_	_	_
preset_adjusting_time	o		5	integer		_	_	_	_
shift_time	m		6	long		A	_	_	_

5.4.5.2.2.1 Attributi

l'attributo 2 contiene l'ora locale del dispositivo, che tiene conto dello scostamento totale da UTC e lo stato dell'ora legale.
Per l'interpretazione degli attributi 3, 4, 5, 6 e 7 si rimanda al Blue Book 12.2 punto 4.5.1. I loro valori di default sono indicati nel prospetto dell'oggetto "Clock".
Per quanto riguarda gli attributi 5 e 6, non essendo di immediata interpretazione, si osserva che essi corrispondono rispettivamente all'ultima domenica di Marzo alle 02:00:00 ed all'ultima domenica di Ottobre alle 03:00:00.

Figure 6.7: Normative Pop-Up window.

checks contained in the procedure. Additionally, on the button of this session, there are two buttons, the first one is a simple cleaner for the text box and the one called SAVE generates a text file with all the information presented in the text box during the testing session. This last button is just active to work after concluding a connection with the meter.



```
------------ Connecting... ------------

Logical Device Name:                   MTSB034E01000018
Metering Point Identifier:             10000001000017
Metrological Firmware Version:         GL35
Application firmware version:          0730
Metrological firmaware signature:      44D14FDB
Application firmaware signature:       38276302

Management Connection successful!!

-------------- Connected !! ---------------------
```

Figure 6.8: Interaction session tool HDLC.

The last session, which is possible to see in Figure 6.8, is dedicated to keeping track of the final results of all the tests executed. This box will just show the crucial information for the stability analysis of the software in the test. The information presented is the execution time, software version, name of test executed, and result. On the right, there is also a save button to store that information in an excel file. This specific file is useful for the stability analysis of a test executed multiple times under the same circumstances.

Figure 6.9: Tracking test session tool HDLC.

That is the complete graphic environment developed in this thesis for the testing department. Following all the requests and with the permanent control of specialists to fulfill the necessity and to be as user-friendly as possible for the final user.

6.3. DSL and tool interaction

The main reason for the creation of this tool is for the execution of autonomous testing. For this purpose, the interaction of the Domain Specific Language and the tool play a major role. As was mentioned before, this interaction started in the drop-down intended to select the test to run (see Figure 6.6), where it is possible to select between all the scripts included in the database folder at the moment to start the tool.

The test starts as soon as the button *RUN* is clicked. The tool opens the file, and a custom parsing mechanism starts. That process takes line by line to analyze them. During the analysis, the first thing is to check if the syntax of the commands is correct, if not the tool would notify the user that there is a syntax problem in the script selected. If this control check is passed then the tool will execute the action expected for the keyword and presents the result to the user through the interaction window (see Figure 6.8).

In the interaction window, it is possible to visualize all the crucial information to keep track of the testing process, and the information that can help the posterior analysis of the log file. This information is composed of the start and end of the test, informational messages, request and response messages with their respective times, objects of interest interpretation, and partial and final results. All the information can be saved at the end of a test session on a text file (examples in Appendix A and B).

Table 6.1 compares how commands are noted in the scripts, and how is the result shown to the user. That intends to generate a clear idea of how the user experience would be during the execution of an automatic test.

Script:	PRINT TEST Valve and billing period closure
Tool:	`TEST Valve and billing period closure`
Script:	ASK Permissions(1B) Schedule_time(Unix)(4B) Preset_duration(2B)

Tool:	`Waiting for parameters ...`
	`Permissions(1B): BF`
	`Schedule_time(Unix)(4B): 62AB04FC`
	`Preset_duration(2B): 0001`

Script:	C C00141000100005E271EFF0200

Tool:	`16/06/22 12:26:49 Req:  C00141000100005E271EFF0200`
	`16/06/22 12:26:50 Res:  C4014100020311BF0662AB04FC120001`

Script:	TWAIT 00.00.10.00

Tool:	`WAITING 10.0 MINUTES ...`
	`TIME OVER!!!!`

Script:	TGOTO (2)

Tool:	`WAITING UNTIL: 2022/06/16 12:35`
	`IT IS TIME!!!!`

Script:	TWAIT 00.00.10.00

Tool:	`WAITING 10.0 MINUTES ...`
	`TIME OVER!!!!`

Script:	CHECK VALUE 01
Tool:	`Check VALUE: Successful!!`
Script:	GET DISCONNECT_CONTROL 2
Tool:	`OUTPUT STATE: Open or connected`

Table 6.1: Tool interpretation of DSL commands

These examples were taken from the result files of the scripts introduced in Chapter 5. The complete version of the test's resultant files can be found in Appendix A and B.

The second file generated by the tool is a tracking result computation table. The tracking file contains just each final result of all the executed tests, this is especially important for the analysis of stability in the iterative test of a feature. Usually, to verify that a feature is working correctly, the same test has to be executed multiple times. With this track report, it is possible to easily access the final results to do a stability evaluation. Table 6.2 recreates the track report resultant after an example test executed 10 times.

No.	**STARTING TIME**	**VERSION**	**TEST**	**RESULT**
1	22/06/22 10:12:25	O730	Example	PASS
2	22/06/22 10:13:25	O730	Example	PASS
3	22/06/22 10:14:25	O730	Example	PASS
4	22/06/22 10:15:25	O730	Example	PASS
5	22/06/22 10:16:25	O730	Example	PASS
6	22/06/22 10:17:25	O730	Example	PASS
7	22/06/22 10:18:25	O730	Example	FAIL
8	22/06/22 10:19:25	O730	Example	PASS
9	22/06/22 10:20:25	O730	Example	PASS
10	22/06/22 10:21:25	O730	Example	PASS

Table 6.2: Tracking report file.

This table contains the fundamental information to evaluate the reliability and stability of the results. This simulation presents a FAIL result of the ten tries. With the identification number of the repetition, it can be addressed to the log file and make an exhaustive analysis about what happened during that iteration to get a FAIL result.

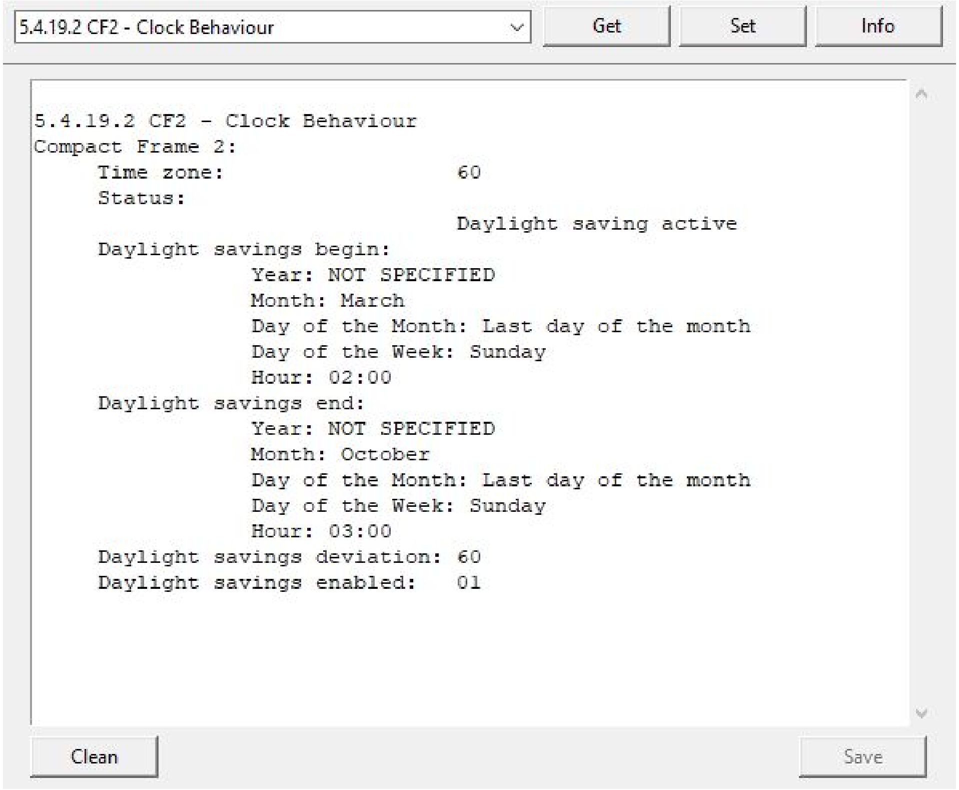
7 | Validation and final results

This chapter presents the validation process carried out in this thesis. In the tool, it was tested the communication stability, the variety of objects implemented, and the degree of independence from the tester. From the Domain Specific Language, it was evaluated with the specialists the capability to recreate real tests, and the simplicity to interact, understand and modify the scripts generated.

7.1. Tool results

With the tool, the validation begins by testing the communication stability. This is essentially important for long-duration tests that can be carried out during hours or days. An error in the communication can cause a wrong evaluation of the results, or it may cause the necessity to restart the complete test losing multiple hours. For this purpose, it was intended to prove that the tool can keep a connection with a meter for a long period without interruption. It was possible to assure a stable connection for at least twelve consecutive hours, which is considered enough for the majority of the cases in which the tool will work. Additionally, it was executing a middle duration test, around three hours of total time execution, to verify the stability during test execution.

Concerning the functionalities and the objects implemented it was checked the possibility to read, interpret and set the different objects and attributes implemented in the device. When the term interprets is mentioned it is referring to decoding the responses to show to the user a result completely understandable to any user, as can be seen in Figure 7.1. For the Get commands, 99% of all objects are implemented for easy access through the tool. For the setting procedures, the percentage decreased to 96%. The mentioned objects are contained in the technical specifications UNI/TS 11291-12-2 [9].



```
5.4.19.2 CF2 - Clock Behaviour
Compact Frame 2:
     Time zone:                   60
     Status:
                                  Daylight saving active
     Daylight savings begin:
                 Year: NOT SPECIFIED
                 Month: March
                 Day of the Month: Last day of the month
                 Day of the Week: Sunday
                 Hour: 02:00
     Daylight savings end:
                 Year: NOT SPECIFIED
                 Month: October
                 Day of the Month: Last day of the month
                 Day of the Week: Sunday
                 Hour: 03:00
     Daylight savings deviation: 60
     Daylight savings enabled:   01
```

Figure 7.1: Example GET object interpretation

The degree of automation for the processes concerned with the tool is completely autonomous, the tester is just required to trigger the connection and to ask for any action required. This autonomy is going to be better appreciated during the interaction with the scripts designed with the DSL.

Finally, concerning the compatibility with other devices, there were made some additional tests to prove if the tool was able to connect with devices that apply other technologies while keeping the same protocols. For this test, it was used the meter *Domusnext 2.0 G4 – G6 MBus*, a similar meter developed by the company including different communication technology. With this device, stable communication was achieved without major problems. It required a simple modification to create perfect communication compatibility. About the object interpretation, there were found some format differences that generate errors with the automatic interpretation. In conclusion, it was obtained multiple device compatibility for communication and testing, but it is not completely compatible with the automatic interpretation of the objects.

After the performance tests of the initial releases of the tool, different teams show their interest to start using it for basic communication and debugging. That is why before concluding this thesis, the tool is already being used inside R&D and after-sales teams.

By the end of this thesis, the software is being frequently used by around 5 people, collecting more than two hundred hours of use.

7.2. DSL validation

During the Domain Specific Language validation two tests were created to verify requirements fulfillment. The first and second ones to be created are presented in the Listings 5.3 and 5.3. These two tests were used to prove the possibility to recreate real tests applying the DSL.

Both tests were executed multiple times obtaining satisfactory results. The tests failed and passed according to the desired results. The resultant files are included in Appendix A and Appendix B respectively. These files were essential for analyzing the level of autonomy during the testing procedures.

Each of these scripts has been put to test on different devices, multiple times to assure the correct functionality. Each test has been carried out at least thirty times to find bugs or failures in the DSL and the interpretation of the scripts. Additionally, the results were analyzed to take the conclusion exposed in the next paragraphs.

The degree of automation was considered an important measure of the fulfillment of the requirements. This measure was particularly taken as the time of completely autonomous testing over the total time of execution. In the first test case, it achieved a 99% of autonomy due to that it requires the user's intervention to set the initial parameter as the first step of the test. Instead, the second one is completely autonomous as a result of the possibility to be executed by the tool without the interaction of anyone. This measure is going to change from test to test, but the important factor to point out is that the language designed have the power to recreate test even completely autonomous.

After the complete implementation of this testing method, the engineer will be mainly responsible for choosing the test to run, setting the parameters, if they are required, and further analysis of the final result if it is required. This represents more than 90% of time saved. The time that the tester is no longer required for the execution of the test. As consequence, it generates a significant increase in the tester's productivity.

As the language was designed all the time side by side with the specialist testing team, it completely meets their requirements. It is found flexible and easy to work with and understand. After a brief demonstration, the testers were able to start to interact with the tool and the existing scripts. The first teams put into contact with the language were the Integration and Validation (I&V), and the Software Development teams. For both teams, there were made short demonstrations of the tool functionalities and an introduction to the Domain Specific Language.

It has been noticed a great acceptance and easy adaptation from all the people in contact with the system. The teams have expressed a clear interest in implementing the DSL for the future execution of a huge variety of tests, with suggestions and recommendations to make it more efficient and intuitive for the next improvements to the language. There are intentions to create a database containing more than 100 test transcripts with the DSL developed in the next months.

8 | Conclusions and future developments

In this thesis, it was developed a specialized tool for the gas meter industry to work as a user-friendly automatic testing instrument for engineers. Applying a Domain Specific Language to create an option to transcript the variety of tests required. It helps to generate an accessible, flexible, and understandable language for people with little coding expertise.

In other words, the purpose was to create a new testing tool that can interact and test the different functionalities of the gas meter Domusnext 2.0. This tool is orientated for specialists in the industry. For this, it was required to create a specific language easily understandable for the target users. With this language, it is supposed to be recreated all the tests to be interpreted and reproduced by the tool.

As a result of the thesis, it was obtained a testing tool specifically designed to work with a DSL, all together created for the gas meters industry. The tool reaches to keep a stable communication, read, interpret and modify a big majority of the COSEM objects implemented. On the other hand, the Domain Specific language is composed of 7 keywords and more than 180 special words to work in collaboration. The scripts generated are completely interpreted by the tool with the result of the execution of an automatic test.

It was obtained a tool capable to generates a stable connection and interacting with a metering device through an optical port. The testing tool was designed to work with the meter*Domusnext 2.0 G4 – G6 NB-IoT*, and it proved the compatibility to interact additionally with other meters that make use of the same protocols. 97% of all interactions with the objects are automatized with a user-friendly interface to interact with the users. The informative window takes to the normative to those not specialized testers. The tool is capable to interpret the test scripts and execute all the instructions automatically. Additionally, It can carry out tests multiple times without additional interaction with the user. It provides a final log file and a results file at the end of the testing session that can be used for further analysis.

On the other hand, the Domain Specific Language was designed with a set of keywords and special words to work together. With a special word for each one of the objects and the designed keywords, it is possible to recreate a big variety of tests. It proved the flexibility to create test scripts, that can be completely autonomous or require a minimum interaction of the user to set the parameter of the tests.

For future developments, the first improvement to do is to complete the implementation of the objects missing for interpretation in the tool. The DSL has to be continuously enriched with new keywords, during the process to create new tests there are going out new requirements that have to be implemented in the language. Also, it is planned to integrate actuators to recreate physical actions that are required for a group of the set, such as pushing the bottoms, disconnecting batteries, or creating a gas flow. Finally, it is contemplated the expansion of the tool for remote testing, sending the testing commands through the different communication technologies implemented in the meters.

Bibliography

[1] Multiprocessing — process-based parallelism. URL `https://docs.python.org/3/library/multiprocessing.html#module-multiprocessing`.

[2] Welcome to pycryptodome's documentation. URL `https://pycryptodome.readthedocs.io/en/latest/`.

[3] Welcome to pyserial's documentation. URL `https://pythonhosted.org/pyserial/1/`.

[4] Python 3.10.5 documentation. URL `https://docs.python.org/3/`.

[5] Tkinter — python interface to tcl/tk. URL `https://docs.python.org/3/library/tkinter.html`.

[6] P. Ammann and J. Offutt. *INTRODUCTION TO SOFTWARE TESTING*. Cambridge University Press, New York, The Edinburgh Building, Cambridge CB2 8RU, UK, 2008. ISBN 978-0-511-39330-3.

[7] ARERA. *UNI/TS 11291-6*. Autorità di Regolazione per Energia Reti e Ambiente, 12 2013.

[8] ARERA. *UNI/TS 11291-12-1*. Autorità di Regolazione per Energia Reti e Ambiente, 10 2019.

[9] ARERA. *UNI/TS 11291-12-2*. Autorità di Regolazione per Energia Reti e Ambiente, 10 2019.

[10] ARERA. *UNI/TS 11291-12-3*. Autorità di Regolazione per Energia Reti e Ambiente, 10 2019.

[11] ARERA. *UNI/TS 11291-12-4*. Autorità di Regolazione per Energia Reti e Ambiente, 10 2019.

[12] Britannica. Software. Encyclopedia Britannica, Jan. 2021. URL `https://www.britannica.com/technology/software`.

[13] B. Broekman and E. Notemboom. *Testing Embedded Software*. Pearson Education Limited, Edinburgh Gate, Harlow CM20 2JE, UK, 2003. ISBN 0-321-15986-1.

[14] DLMS User Association. Blue book: Cosem interface classes and obis object identification system. Technical Report 13, DLMS User Association, May 2019.

[15] DLMS User Association. Green book: Dlms/cosem architecture and protocols. Technical Report 9, DLMS User Association, May 2019.

[16] H. V. Gamido and M. V. Gamido. Comparative review of the features of automated software testing tools. *International Journal of Electrical and Computer Engineering*, 9(5):4473–4478, Oct. 2019.

[17] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*, pages 45–62, 227–266. The name of the publisher, 2007. doi: 10.1002/9780470249260.ch3.

[18] M. Lethrech, A. Kenzi, I. Elmagrouni, M. Nassar, and A. Kriouile. A process definition for domain specific software development. In *2015 Third World Conference on Complex Systems (WCCS)*, pages 1–7, 2015. doi: 10.1109/ICoCS.2015.7483261.

[19] S. Limphapayom, D. H. Le, and W. Pora. An emulation of data concentrator units conformed to dlms-hdlc protocols. In *2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, pages 1–6. Institute of Electrical and Electronics Engineers, 2013. doi: 10.1109/ECTICon.2013.6559622.

[20] M. Mernik, J. Heering, and A. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37:316–344, Dec. 2005. doi: 10.1145/1118890.1118892.

[21] M. Prasanna, S. N. Sivanandam, R. P. Venkatesan, and R. Sundarrajan. A survey on automatic test case generation. 2011.

[22] G. Rotigliano. The italian gas smart metering obligation: A strategic analys of implications and opportunities. Master's thesis, Politecnico di Milano, 2011/2012.

[23] S. K. Singh and A. Singh. *Software Testing*. Vandana Publications, UG-4 Avadh Tower, Naval Kishor Road, Hazratganj, Lucknow-226001, India, 2019. ISBN 9788194111061.

[24] C. A. L. D. . Thesaurus. Software. Cambridge University Press, 2022. URL `https://dictionary.cambridge.org/dictionary/english/software`.

A | Appendix A

Here it is presented the log file of the second test case presented. It correspond to the execution of *"Installer setup"* test.

```
------------ Connecting... ------------

Logical Device Name:                   MTSB034E01000020
Metering Point Identifier:             10000001000017
Metrological Firmware Version:         GL35
Application firmware version:          0730
Metrological firmaware signature:      A1919F0B
Application firmaware signature:       E8AB938D

Management Connection successful!!

-------------- Connected !! ---------------------

-------------- Start Test ---------------
TEST Installer/maintainer setup

Waiting for parameters ...

Permissions(1B):  BF
Schedule_time(Unix)(4B):  62AB04FC
Preset_duration(2B):  0001

-- Setting the Installer window.

16/06/22 12:26:48
    Req: C10141000100005E271EFF0200020311BF0662AB04FC120001
16/06/22 12:26:49 Res: C5014100

16/06/22 12:26:49 Req: C00141000100005E271EFF0200
16/06/22 12:26:50 Res: C4014100020311BF0662AB04FC120001

Check SET: Successful!!

```

```
WAITING 10 SECONDS ...
TIME OVER!!!!

-- Test that the window is not started

16/06/22 12:27:00 Req: C00141000300005E271FFF0200
16/06/22 12:27:01 Res: C40141000600000E10

Check VALUE: Successful!!

WAITING UNTIL: 2022/06/16 12:35
IT IS TIME!!!!

WAITING 10.0 MINUTES ...
TIME OVER!!!!

-- Test that the window already start

16/06/22 12:45:51 Req: C00141000300005E271FFF0200
16/06/22 12:45:52 Res: C40141000600000BAE

Check VALUE: Successful!!

WAITING 60.0 MINUTES ...
TIME OVER!!!!

-- Test that the window have been closed after the
    Preset_duration

16/06/22 13:46:34 Req: C00141000300005E271FFF0200
16/06/22 13:46:35 Res: C40141000600000000

Check VALUE: Successful!!


  -------------------------------------
  |       TEST RESULT: PASS           |
  -------------------------------------


--------------- End Test ------------------

----------------- Disconnected !! -----------------
```

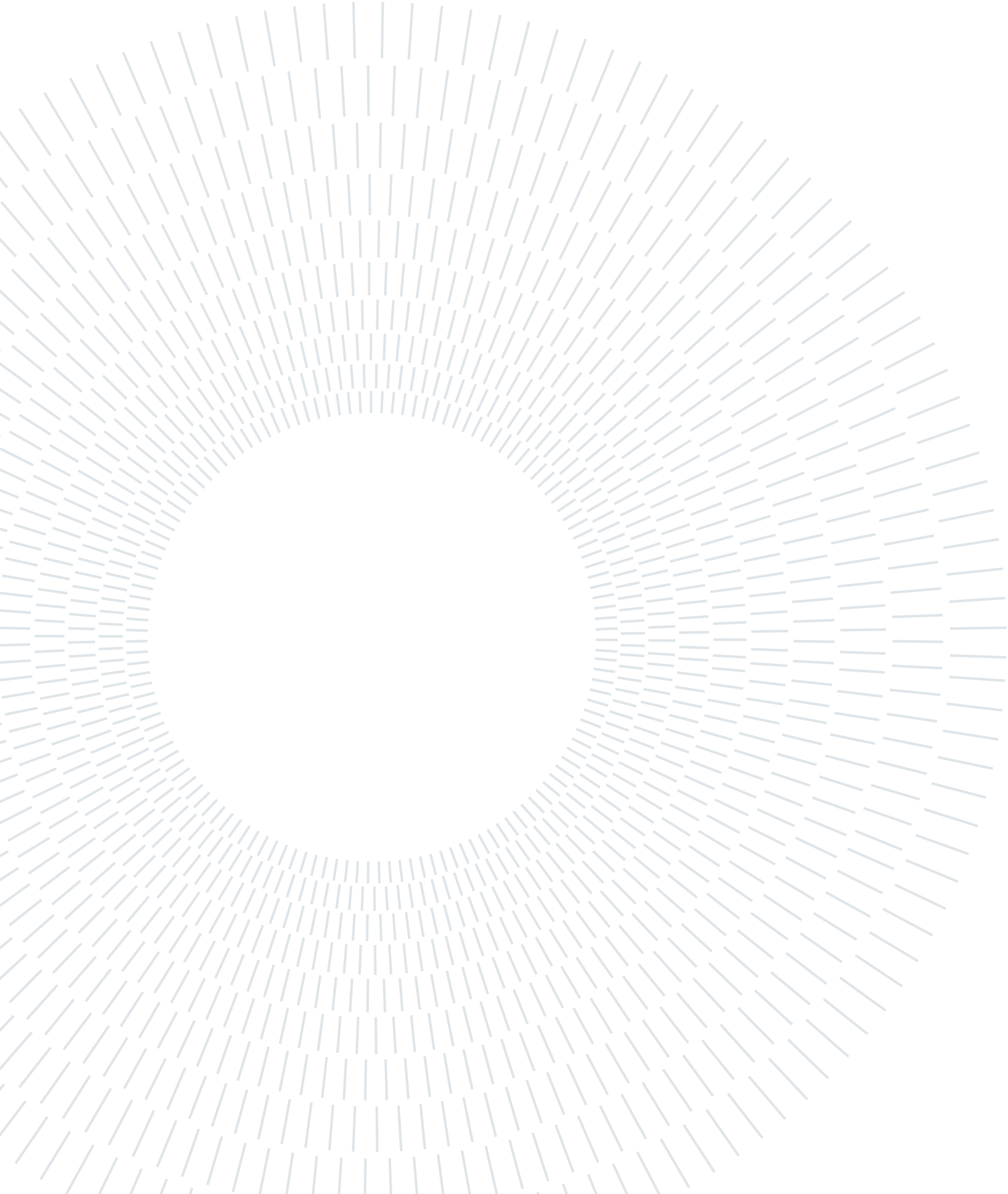
B | Appendix B

Here it is included the output file saved from the tool after the execution of the test *"Valve and billing time closure"*.

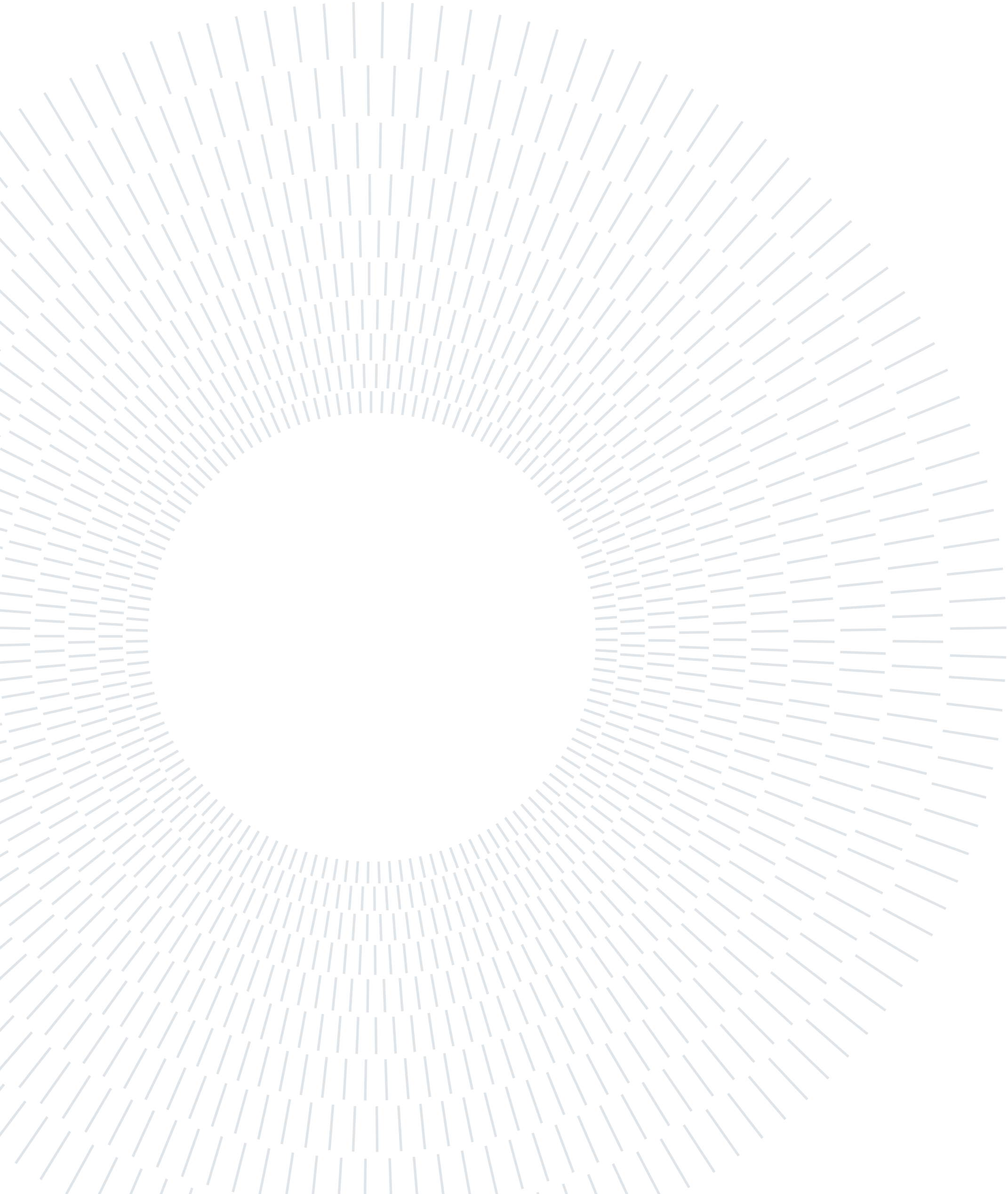
```
------------ Connecting... ------------

Logical Device Name:                    MTSB034E01000020
Metering Point Identifier:              10000001000017
Metrological Firmware Version:          GL35
Application firmware version:           0730
Metrological firmaware signature:       A1919F0B
Application firmaware signature:        E8AB938D

Management Connection successful!!

-------------- Connected !! ---------------------

-------------- Start Test ---------------

TEST Valve and billing period closure

-- Reading stated of the valve and checking that it is open.

16/06/22 10:06:57 Req: C001410046000060030AFF0200
16/06/22 10:06:58 Res: C40141000301

OUTPUT STATE:        Open or connected

16/06/22 10:06:58 Req: C001410046000060030AFF0200
16/06/22 10:06:59 Res: C40141000301

Check VALUE: Successful!!

-- Saving billing period counter

16/06/22 10:06:59 Req: C0014100010700000100FF0200
16/06/22 10:07:00 Res: C40141001101
```

```

Value saved:     01
Billing / Snapshot Period Counter:     1

-- Configuring script for valve closure

16/06/22 10:07:00
    Req: C10141001600000F0001FF02000202090600000A006AFF120003
16/06/22 10:07:01 Res: C5014100

16/06/22 10:07:01 Req: C00141001600000F0001FF0200
16/06/22 10:07:02 Res: C40141000202090600000A006AFF120003

Check SET: Successful!!

-- Setting an execution time in the past for immediately valve
    closure

16/06/22 10:07:02
    Req: C10141001600000F0001FF040001010202090411000000000090507E1090705
16/06/22 10:07:03 Res: C5014100

WAITING 0.25 MINUTES ...
TIME OVER!!!!

-- Reading stated of the valve and checking if it is close.

16/06/22 10:07:18 Req: C001410046000060030AFF0200
16/06/22 10:07:19 Res: C40141000300

OUTPUT STATE:       Closed or disconnected

16/06/22 10:07:19 Req: C001410046000060030AFF0200
16/06/22 10:07:20 Res: C40141000300

Check VALUE: Successful!!

16/06/22 10:07:20 Req: C0014100010700000100FF0200
16/06/22 10:07:21 Res: C40141001102

Value saved:     02
Billing / Snapshot Period Counter:     2

Check VALUE: Successful!!
```

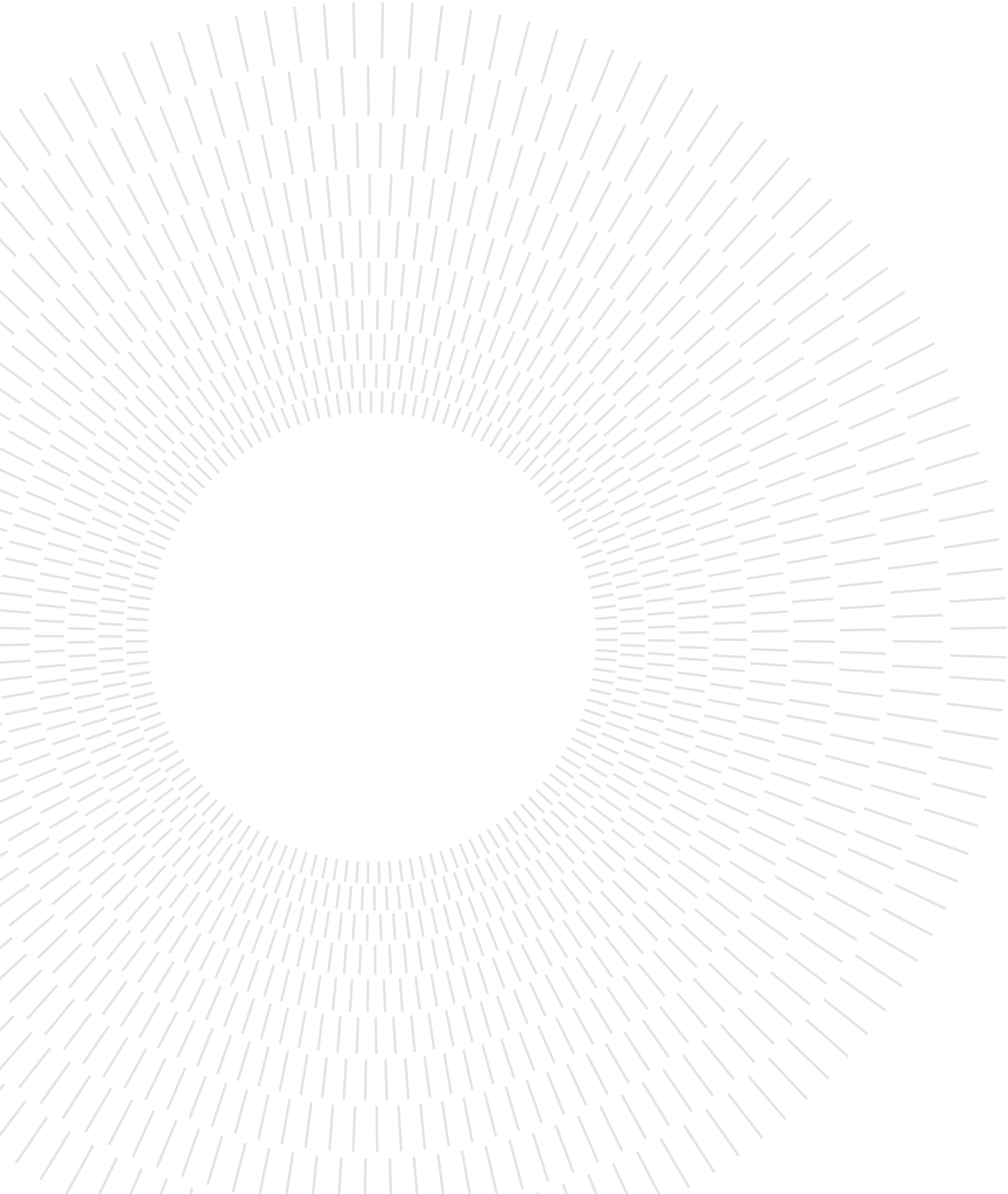
```

  -------------------------------------
  |       TEST RESULT: PASS           |
  -------------------------------------


--------------- End Test ------------------

----------------- Disconnected !! -----------------
```



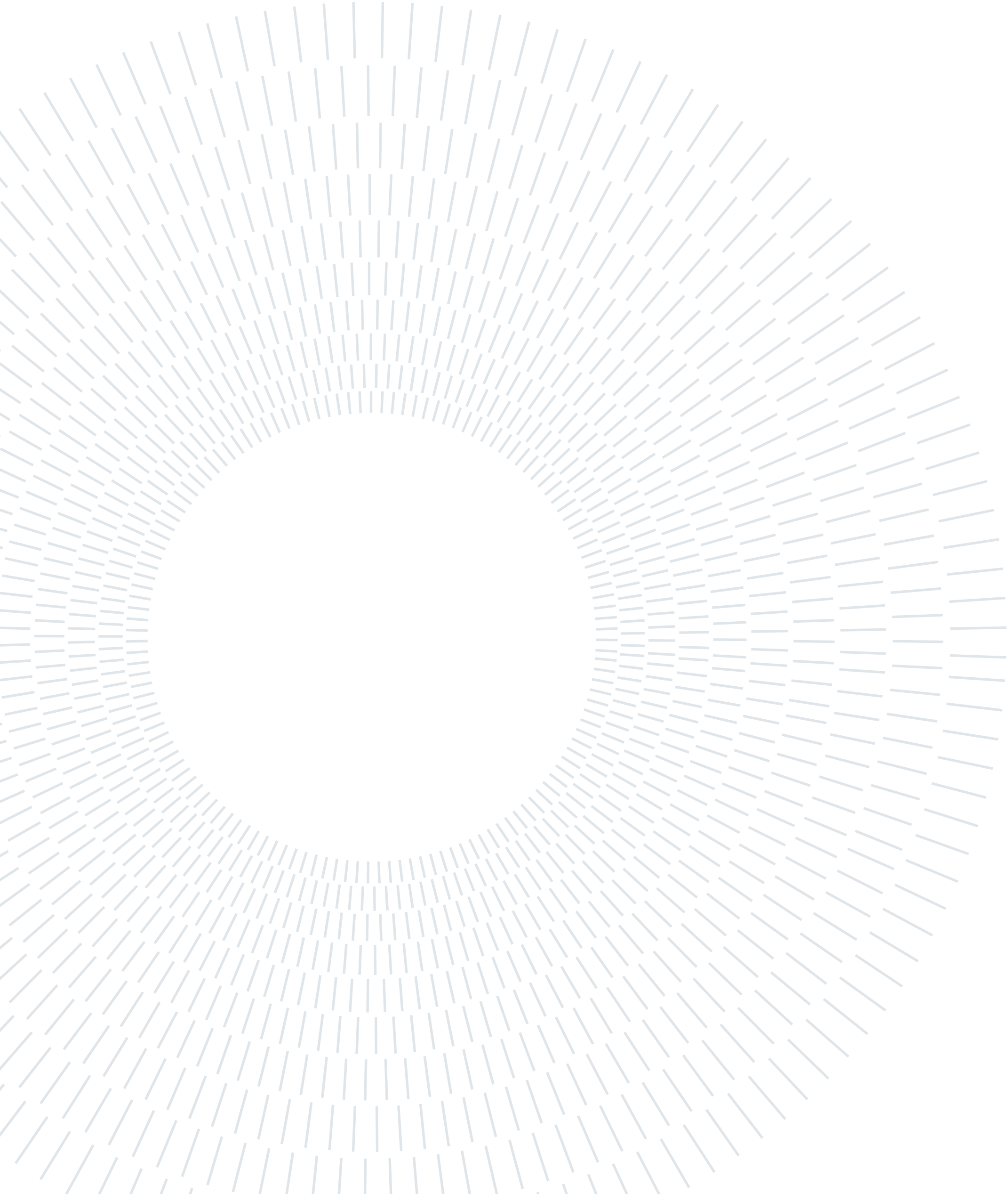
List of Figures



List of Tables



Listings



Acknowledgements

To conclude, I would like to thank Claudio Bianchi, Michela Passerini, Andrea Sulsenti, and all the R&D team of MeteRSit. This team took me as a member and guided me to achieve this goal. Additionally, I want to thank Prof. Matteo Rossi for guiding me through this thesis.

To all my friends, each one of you makes this possible and something memorable to remember for life. To my girlfriend, thank you for keeping me company and for all the help with this project.

Finally, thanks to my parents and sister, the people who raised, supported, and were by my side throughout whole my life. Without them, I would never arrive where I am today, and to present this project as the last requirement to conclude my studies. They have been in each step and taken part in every decision taken during my life. This achievement is as much mine as theirs.