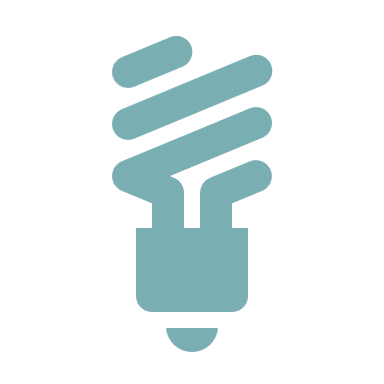
**Distribution boxes of electric power** - Software Architecture

Infor**MAT** - Mattia **Piazzalunga** & Matteo **Severgnini**



**Table of contents**

[Introduction & Theory 3](#_Toc154425585)

[What is software architecture? 3](#_Toc154425586)

[Funzionalità 5](#_Toc154425587)

[Attributi di qualità 5](#_Toc154425588)

[How to identify a viable architectural solution? 5](#_Toc154425589)

[Project track 6](#_Toc154425590)

[Studio iniziale 6](#_Toc154425591)

[Acronyms used in the report 8](#_Toc154425592)

[Assumptions 8](#_Toc154425593)

[Problem architecture 10](#_Toc154425594)

[Introduction 10](#_Toc154425595)

[Use case diagram - "WHO" specification 10](#_Toc154425596)

[The team 16](#_Toc154425597)

Introduction & Theory

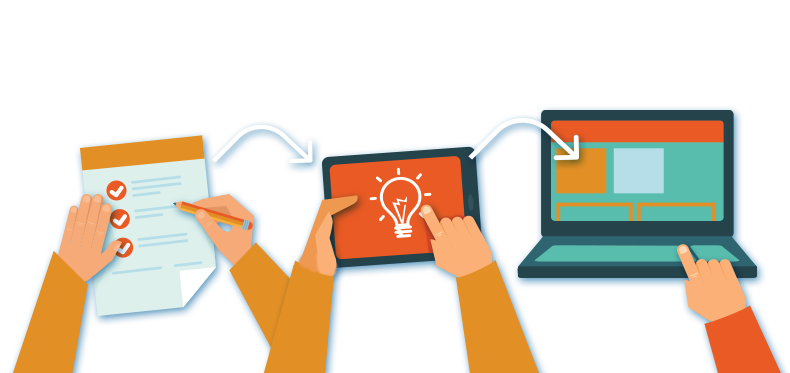
What is software architecture?

We start talking SW architecture in the early 1990s, with the emergence of the need to provide solid structure to the system before actual development. It is the starting point of any software development.

According to an IEEE definition, software architecture is the fundamental organization of a system, embodied in its components (computational components), their relationships to each other and to the environment, and the principles that govern its design and evolution. It is, therefore, a higher level of design than concrete design: we do not define how an element is represented, but we highlight its existence, under certain constraints. In general, therefore, there is no 1:1 mapping with representation, but certain observations about elements that are not useful for reasoning about the system are omitted. There is, however, no perfect architecture; it depends strongly on the type of system.

NB: the analysis shows us what the system is supposed to do, without the how. Opposed to concrete design, which, based on diagrams, highlights to me the final implementation.

Architecture, in general, matches elements with functional requirements, ensuring nonfunctional requirements.



**Structures**

**Structures** are the fundamental part of SW architecture. A structure is a **set of elements held together by a relationship**. It becomes **architectural** if it supports reasoning about the system and its properties.

We can, therefore, redefine the concept of architecture as a set of structures, each of which is a set of elements.

In the complexity of modern systems, we make use of **three types of structures based on the aspect to be modeled** (they capture different aspects of the system, I use them to verify aspects of different qualities):

* **Modules**

We partition the system into implementation units called modules, which are static and focus on how the functionality of the system is grouped.

We have software elements (classes, layers, feature groups) linked by relationships.

**NB**: We highlight, at this level, simply which modules interact ta them.

**Substructures***:*

*A close-up of a document

Description automatically generated*

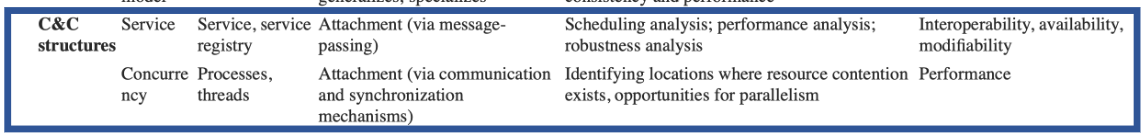
**NB:** In substructures, modules are the same, but are represented in different ways to understand whether all constraints are met.

Module views are commonly mapped to component and connector views. Implementation units shown in module views have a mapping to components that are executed at runtime

* **Components and connectors**

**Module instances are the components**. Components and connectors give us insight into the dynamic nature of the system: how are constraints and nonfunctional requirements met at run-time? This framework focuses on how components services, peers, clients, servers, filters) interact via connectors (call-returns, process synchronization, pipes, protocols, API rests, ...)

**Substructures**:



* **Allocation structures**

They describe the mapping between software structures and environnements (between software structures and non-software structures such as CPUs, file systems, networks, development teams). They can be used for both modules and components. They allow, for example, different modules to be assigned to different development and deployment environments.

They can be:

* + **Organizational**

To which team assign the development of a module.

* + **Development**

How to split a module into files or where to save them (github for example)

* + **Deployment**

Which component to deploy and on which machine (server, etc.).

* + …

**Substructures**:

A close-up of a document

Description automatically generated

**NB**: we can see that, in structures, we always have elements and connections between them;

Structures, however, are related, representing as we said, the same system from different points of view.

In general, however, structures are "an idea." The "on paper" representation of a structure is called a view. A common format for representing the view is the UML model.

**NB**:Anything that does not have a constraint I do not treat at the architectural level!!! In architecture I do not treat superfluous elements, I leave them out (I have abstraction). Saying, for example, how classes communicate within a module is the task of concrete design.

**NB**: Documentation, especially architectural, is, in general, important to allow you to get your hands on a system.

Functionality

**Functionality/functional requirements are the basis of a system**. However, systems are often changed because nonfunctional requirements (quality requirements) are not met.

In generale, comunque, abbiamo:

* **Functional requirements.** These requirements highlight what the system should do and how it should react to stimuli.
* **Quality requirements.** These requirements express quality requirements on functional aspects (e.g., how fast an operation should perform or how error-resistant it is) or on the system as a whole (e.g., development time).
* **Constraints.** These are choices imposed by the architect (e.g., use Ruby because all programmers use it) or choices dictated by the environment (e.g., the company makes only web apps, the company uses only certain frameworks, ...).

Quality Attribute

A software architecture, if chosen correctly, must satisfy quality attributes. A quality attribute is a measurable or verifiable property of a system that is used to indicate the degree to which it meets the needs of stakeholders. It is important to point out at the outset that quality attributes often depend on the environment and the constraints it dictates.

In general, however, it is not possible to guarantee them all in a system; one must understand which aspects are predominant and select architectural solutions that are promoters of these requirements. The most important set of requirements is the ASR (Architectural Significant Requirements).

In each case, as we mentioned earlier, these qualities must be measurable. Moreover, we must be able to associate only one dimension of quality with certain problems and without having ambiguity in the choice.

How to identify a viable architectural solution?

One technique (among many) for properly designing an architectural solution is to design the architecture based on views. The steps to be identified next are (in order):

1. **Problem Architecture**

In questa fase si cerca di capire il problema, verificando la comprensione dei requisiti funzionali e identificando quelli non funzionali. In generale, ci si concentra su:

* + Cosa deve fare il sistema?
  + Chi sono gli attori?
  + Quali sono i dati? E quali trasformazioni subiscono affinché i dati di interesse per il sistema possano essere realizzati?

1. Logic Architecture

In this step we go on to decompose the functionality of the system into logical modules, i.e., indivisible blocks (separations of concern). In this step the architecture must guarantee the identified ASRs. In general, we focus on 'component identification: how to "divide" the system in the best possible way with respect to the various functionalities/functional requirements. In addition, comparative assessments are made between the choices;

1. Concrete Architecture.

In this step we are going to look at how the logical components interact (what protocols etc.) and what technological solutions allow us to realize the logical components;

1. Deployment Architecture.

In this point we focus on the structure components components and connectors deployed in the environnement (software components that need to be installed on the hardware), specifically the identification of processing nodes and what components are present on them;

1. Implementation Architecture (not addressed in this project).

In this step, technologies, platforms, programming languages are chosen as far as possible, …

Important, in any case, is to consider from the outset the constraints imposed on us.

To represent the system, in any case, we will make use, as already pointed out, of views represented through the UML model.

Project track

We proceed by reporting the outline of the project...

A company is engaged in the design and installation of power distribution substations. The substations are scattered throughout the territory and are equipped with sensors for instantaneous measurement of the power delivered. An operational management system is to be designed that:

* Acquires "real-time" data from the individual power stations. The data detectable by the sensors is the instantaneous power delivered (in kw);
* Checks for abnormal situations (abnormal power peaks compared to the limits set for the control unit);
* In the case of abnormal situations, turns off the control unit and notifies the central technical service of the abnormality;
* Supports the decisions of the central technical service to identify the most suitable operator (by availability, geographic proximity, technical expertise related to the type of control unit) to repair the fault;
* Notifies the operator of the action to be taken;
* Allows the operator to notify the technical service to initiate the intervention and its completion;

Collects instantaneous power consumption data in order to define new power unit distribution policies.

Initial study

Acronyms used in the report

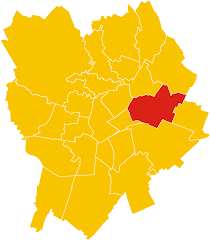
We provide a list of acronyms used in this report to have, at each point in this report, clear references:

1. SA, Software Architecture;
2. kWh, kilowatt-hour;
3. ECU, Electrical Control Unit.

Assumptions

Since we were given little information from the outline, we proceed by making assumptions (in reality to be verified with stakeholders):

* We assume to work, in terms of current regulations, in Italy and to Italian contractual situations;
* Assume that the control unit is viewed as a "single block" with its sensors. Such a control unit has a certain level of software "intelligence," such as:
  + Converting analog sensor signals to digital and sending them;
  + Accepting power on and off commands.
  + Checking sensor values to decide whether or not to disconnect power
* Suppose we operate in the Borgo Palazzo neighborhood area of the municipality of Bergamo, where there are, according to data from the Regulatory Authority for Energy Networks and Environment (ARERA), as of December 31, 2022, 10,800 housing units with active electric meters.



Again according to ARERA, that district refers to only one power distribution plant, assumption we will make for this project.



<https://shorturl.at/kwxI4>

* The power packs referred to in the trace we assume refer to domestic power packs, thus meters in people's homes, whose maximum peak power is 3kw;
* By abnormal situations we refer to:
  + power peaks compared to the limits provided for the control unit. As limits we use those provided by e-distribuzione, the largest company in Italy in the electricity Distribution and Metering sector serving more than 31.5 million Customers connected to the grid. A limit is referred to as 10% higher than the committed power subscribed at the contract level. For example, for a 3 kWh contract, it is possible to draw up to 3.3 kW without time limit. In addition, if you exceed 3.3 kWh, you are given the option to draw up to 4 kWh for three hours; at three hours, a warning is triggered. If more than 4 kWh is drawn, the power supply control device is triggered at two minutes. [shorturl.at/rHOT5](https://shorturl.at/rHOT5).
  + Overheating.

Usually, the operating temperature of an ECU is between [-25 C°; +55 C°] degrees for e-distribution ECUs. For this reason, a temperature beyond the range triggers, after the classic three checks, the signaling (<https://shorturl.at/cptO5> );



This assumption was made because, for a simple "peak power" problem, it would not make sense to activate the output of an operator (required by the trace).

* The central system queries the control unit every 60 seconds to take data useful for setting new policies;
* The system running on the control unit, every 60 seconds (3 "blocks" of data generated), averages the latest data (trying to avoid "spurious" data) and checks if an anomaly is present (anomaly reported by the data).
* All anomaly notifications from the system are sent instantaneously when generated, but asynchronously;
* The operator is identified geographically by latitude and longitude (location) provided by a GPS instrument, this is because he may not even be in a building and, therefore, does not have a reference address available;
* Working in Italy and having associated one ECU per housing unit, we are going to associate in a 1:1 relationship an ECU, which in any case will be identified with an id internal to the system, with an address;
* Every 30 days the study of the collected data takes place for the definition of new policies, where by policy we mean the insertion, shifting or upgrading of the distribution unit.

... other, less important assumptions are set out at the same time as the explanatory views.

Problem architecture

Introduction

The problem architecture must **define**:

* The **functionality that the system must offer to external actors** (functional requirements)  
  **Diagram**: UML use case diagram.
* **The types of information/data that the system needs to handle for the realization of functionality**. By now all systems work on the data where I have to express constraints, etc..

**Diagram**: UML class diagram.

* The information flows that the system must support to achieve the functionality defined in step 1 by operating on the data defined in step 2 (in/out). **How information evolves within my system**.

**Diagram**: UML activity diagram.

* Represent the control flows that the system must support to ensure functionality. They specify why certain things happen, why a component is activated, etc…

**Diagramma**: UML activity diagram.

**NB**:

* the same diagram with a certain syntax can have multiple semantics *(e.g., we can use the class diagram to model both data and concrete components)*;
* we have strongly data-driven systems that are therefore to be dealt with at the architectural level.

Questions are attempted to be answered to capture the basic elements of the problem:

* **Static aspects**. It's those questions that allow me to identify, then, the forms.
  + **Who**. Who is part of the system? Who are the actors?
  + **What**. What are the data? What entities make up the system?
  + **Where**. Where are the data and the who of the system? Is the user sitting at the terminal or iterating with a tablet?
* **Dynamic aspects**
  + **How**. How do the transformations take place? How is information manipulated so that the system does what is required?
  + **Why**. Why do certain things happen? Why are activities activated? What are the mechanisms dietri the innate?
  + **When**. When do things need to be done? (delay, timing, ...) How often?

Use case diagram - "WHO" specification + “WHERE” for the actors

link here

Actor is everything I take for granted that already exists and that in no way I have to implement. Actors are not only human beings, but also pieces of software: if we exploit the services or command it we do. Actors are all external, as-is entities: we exploit it.

**NB**: there is no such thing as the system actor! The system is the whole piece of software, it is the set of use cases. The system is what needs to be realized/developed.

It is, moreover, important for actors to specify where: a person moving or not influencing the system differently. For this reason, each actor will have a labeling to indicate the where.

Another important aspect to consider is whether the actor is internal or external to the system itself: if the actor is not internal to the system we need to figure out what communication interfaces to have with it.

**NB**: if the actor is internal to the system I need a datum that models it to me semantically.

Immagine che contiene testo, diagramma, schermata, Piano

Descrizione generata automaticamente

Let us specify of the rationale to allow full understanding of the choices less intuitable from the diagram.

|  |  |
| --- | --- |
| **Element** | **Responasability** |
| Data acquisition | The central system periodically acquires data from ECU. Data acquisition è un passaggio fondamentale, necessario per far in modo che i dati acquisiti siano gli stessi usati durante ECU Check e durante Saving Data. Include serve proprio per esprimere che Data acquisition è un operazione obbligatorio per ECU Check e Saving Data. |
| ECU check | Il sistema verifica ed interpreta i segnali forniti dalla componente intelligente della centralina e, in caso di anomalie, ha la capacità di disabilitarla (Disable ECU) e notificare l’anomalia al “Central Technical Service”.  La centralina, composta da una parte intelligente, legge i dati, come specificato nelle assunzioni, e verifica che non ci siano anomalia. Qualora vengano rilevati malfunzionamenti, viene spenta la centralina e successivamente inviata una notifica al sistema tecnico centrale.  *Successivamente vedremo deployato l’insieme di questi casi d’uso sulla centralina stessa per disabilitare la centralina contestualmente all’anomalia, anche nel caso in cui non sia chiamabile da remoto.* |
| Operator identification | The system, when invoked by the "Central Technical Service," allows the identification of an operator using certain features. The real-time geographic location is requested from the intelligent component in use by the technician using GPS. The identification can end in a request for action to the technician (Sending action report). |
| Policy optimization | Il sistema, ogni 30 giorni come da assunzione, consulta lo storico persistente per creare delle statistiche e delineare nuove politiche di distribuzione delle centraline, come aggiunta o rimozione di una centralina, e successivamente notifica (Notification policy) il “Central Technical Service”. |

Data diagram - "WHAT" specification + “WHERE” for the data

link here

To help us understand what the data model is, let us differentiate it from the domain model. The domain model tries to capture with entities what we are talking about. It is a broader concept that represents an in-depth understanding of a particular domain or area of activity. It focuses on understanding the entities, concepts, rules and relationships that characterize the domain in question.

*Ex: in a photo program I will have the photo, the retouched photo, the camera, etc. Here, in addition, we can also put the functionality: take photo, etc.*

When we make a **data model**, however, we are simply dealing with information: the data. The data model is a specific part of the domain model and focuses primarily on the structure of the data used in a system or application. I will only have the transit data; everything else I take out. Here, also, I have **attributes that have no visibility or type.**

*Ex: I will not have the concept of the camera, but rather the photo, the retouched photo, etc..*

Some warnings:

* attributes have neither quality nor type;
* relationships between positions can also be modeled;
* An actor can be brought into the system by modeling it as a datum, with a representation that is necessary for my purposes. For example, I am interested in knowing the origin of data we can associate with a datum that is the actor itself.
* To show that a datum is different depending on when it is processed (multiple evolutions) we use **generalization**;
* We can **add units of measurement**;
* We can **add labels to better characterize the data**. For example, specifying the size in terms of bytes of the data allows us to think later about how much bandwidth is needed for the transfer. It is also important, for example, to specify precision to understand how much delay or loss of information we can tolerate: changing precision often forces us to change the type of system;
* We can also specify **abstraction by label**: from what system aspect do we see the object to "call" it a certain way? What level of abstraction are we at (semantics changes)? This information is useful, then, when we do partitioning of the system into modules.

Immagine che contiene testo, diagramma, schermata, Piano

Descrizione generata automaticamente

|  |  |
| --- | --- |
| **Element** | **Responasability** |
| ECU | ECU is an actor modeled as a datum, which is essential to know the origin of the datum (ECUData), but also to always have control of the notification - ECU reference. It has a unique id and an address, also unique, to know the exact location of it and allow interventions as well as study for new policy definition. |
| ECUdata | ECUData is the actual data generated by the ECU sensors (transformed into digital by the intelligent component of the ECU). The data consists, at a certain timestamp, of temperature and kilowatt-hours, both of which are useful for studying anomalies.  The data temporarily sits on the software part of the system running on the control unit (InECU) for the study of any anomaly (average data from three extractions are used). Once these data are studied, they are forgotten. *We will also highlight this later, but it is essential that a "software piece" is running on the ECU because in the event of an anomaly, but with communication problems with the central server, the ECU must still have the ability to be disabled.*  The data are, every 60 seconds, taken from the central system to be saved in the central dataset (InSystem) and allow subsequent studies. The data block in transit will involve a thousand bytes that must be sent with a high precision: we do not want a very rigid system, but it is important that these data arrive accurate, with low losses and minimal latency.  We expressed this semantics through a generalization that allows us to understand how data, even if it is the same, has a different meaning and is treated differently depending on where it is located. |
| Notice | All actors in our system exchange notifications of a thousand bytes: it is the central communication tool of our system. All notifications are associated with an ECU. Those, in addition, that interface with an operator will have one associated with it (to which multiple notifications can reach).   * + AnomalyNotice which is concerned with the notification of the anomaly to the Central Technical Service must have high accuracy. Notification must be reported promptly and with minimal loss. We can, however, tolerate very small leaks-we are not talking about critical systems. Here we have, of course, an associated anomaly.   + InterventionNotice, i.e., reporting of anomaly to the technician, must also have an associated anomaly and low latency, although we can tolerate lower accuracy on this type of notification.   + Notifications of the start and end of intervention are the ones we can tolerate less accuracy of, since they are only technical-Central Technical Service reports with little important data.   Qui la generalizzazione è utile non a specificare la posizione del dato, bensì ad accumunare e differenziare tutte le notifiche con un id progressivo: l’intento è quello di esprimere le diverse tipologie di notifica che il sistema può gestire. |
| Anomaly | Il sistema tiene un pool di anomalie a cui fare riferimento (la componente intelligente e le notifiche si basano su questo pool possibile di anomalie segnalabili). Ogni anomalia ha un id univoco, nonché una descrizione ed una macro-categoria di riferimento (0 - temperatura, 1 – kilowatt, … ).  (esempio: Anomalia id:001, macro-category:1, description: superamento kWh massimi)  NB: non è necessario includere questo dato all’interno delle notifiche di inizio e fine intervento. |
| Operator | Each operator has id and basic information (the classical identification data), as well as the technical knowledge that distinguishes it (to operate) and a binary availability flag at that given instant.  Each operator is associated with a position (OperatorPosition) provided by a GPS instrument, which is why we need latitude and longitude (not afferent at a given time to a home, but to a long in 2D space).  The relationship between Operator and OperatorPosition is an N-1 because we are not interested in keeping a history of positions (it does not make sense, we do not want N:N), but the operator will always have only the associated position at the given time provided by the GPS system. |
| PolicyStatisticsNotice | Come sappiamo, ogni 30 giorni, il sistema tenta di segnalare se vi sono politiche attuabili per approvvigionare meglio le ECUs, come l’inserimento, lo spostamento o il potenziamento della centrale di distribuzione. Qual ora una nuova politica risulti applicabile (e solo in questo caso) il sistema invia una notifica al Central Technical Service con alcuni indicatori statistici di base, dei dati sulle centraline analizzate, un’intensità della necessità di attuare tale politica (scala in cinque valori), data di inizio e fine dei dati analizzati, nonché la posizione delle centraline analizzate (identificata da un’area definita da centro e raggio). |

Activity diagram – “HOW” & “WHY” specificationsConclusion

The team

This report is maintained by the InforMAT Software Architecture course group, whose members are:

* MATtia **Piazzalunga** - 851931;
* MATteo **Severgnini** - 851920.