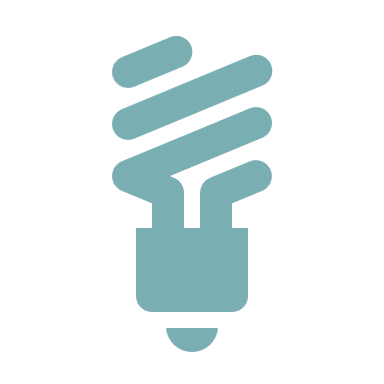
**Distribution boxes of electric power** - Software Architecture

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



**Table of contents**

[Introduction 3](#_Toc151915530)

[What is software architecture? 3](#_Toc151915531)

[Project track 3](#_Toc151915532)

[Acronyms used in the report 4](#_Toc151915533)

[Project organization 4](#_Toc151915534)

[How to identify a viable architectural solution? 4](#_Toc151915535)

[Studio iniziale 4](#_Toc151915536)

[Assumptions 4](#_Toc151915537)

[Problem architecture 4](#_Toc151915538)

[Introduction 4](#_Toc151915539)

[Conclusion 5](#_Toc151915540)

[The team 5](#_Toc151915541)

Introduction

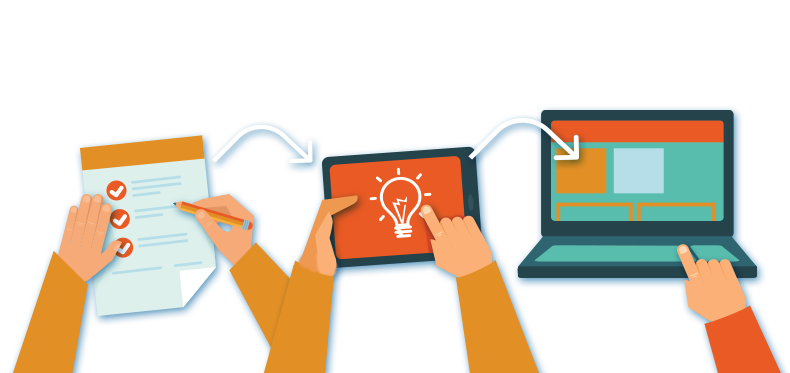
What is software architecture?

Si comincia a parlare architettura del SW all’inizio degli anni ’90, con la nascita della necessità di fornire struttura solida al sistema prima dello sviluppo vero e proprio. Essa è il punto di partenza di qualsiasi sviluppo software.

Secondo una definizione dell’IEEE, l'architettura del software è l'organizzazione fondamentale di un sistema, incarnata dai suoi componenti (componenti computazionali), dalle loro relazioni reciproche e con l'ambiente, e dai principi che ne regolano la progettazione e l'evoluzione. È, quindi, un livello di design superiore al design concreto: non definiamo come un elemento è rappresentato, ma ne evidenziamo l’esistenza, sotto certi vincoli. In generale, dunque, non vi è una mappatura 1:1 con la rappresentazione, ma vengono omesse alcune osservazioni sugli elementi che non sono utili al ragionamento sul sistema. Non esiste, comunque, un’architettura perfetta, dipende fortemente dal tipo di sistema.

NB: l’analisi dice cosa il sistema deve fare, senza il come. Opposto al design concreto che, basato su diagrammi, mi evidenzia l’implementazione finale.

L’architettura, in generale, mette in corrispondenza gli elementi con i requisiti funzionali, garantendo dei requisiti non funzionali.



Project track

Un’azienda si occupa della progettazione e installazione di centraline di distribuzione dell’energia elettrica. Le centraline sono sparse sul territorio e sono dotate di sensori per la misura istantanea della potenza erogata. Si deve progettare un sistema di gestione operativa che:

* acquisisce “in tempo reale” i dati dalle singole centraline. Il dato rilevabile dai sensori è la potenza istantanea erogata (in kw);
* controlla se si verificano situazioni anomale (picchi di potenza anomali rispetto ai limiti previsti per la centralina);
* nel caso di situazioni anomale, disattiva la centralina e notifica l’anomalia al servizio tecnico centrale;
* supporta le decisioni del servizio tecnico centrale per identificare l’operatore più adatto (per disponibilità, vicinanza geografica, competenze tecniche relative al tipo di centralina) a riparare il guasto;
* notifica all’operatore l’intervento da effettuare;
* consente all’operatore di comunicare al servizio tecnico l’avvio dell’intervento e il suo completamento;
* raccoglie i dati del consumo di potenza istantanea al fine di definire nuove politiche di distribuzione delle centraline.

Acronyms used in the report

Forniamo una lista di acronimi utilizzati in questo report:

1. SA, Software Architecture;
2. kWh, kilowattora;

How to identify a viable architectural solution?

Una tecnica (tra le tante) è quella di progettare l’architettura sulla base di views. Gli steps da identificare successivamente sono (in ordine):

1. Architettura del problema.

In questo step cerchiamo di comprendere il problema, verificando la comprensione dei requisiti funzionali ed identificando i non funzionali. In generale, ci concentriamo nel:

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

1. Architettura logica.

In questo step andiamo a scomporre le funzionalità del sistema in moduli logici, ovvero blocchi indivisibili (separations of concern). In questo step l’architettura deve garantire gli ASR identificati. In generale ci concentriamo nell’ identificazione dei componenti: come spazzare il sistema nel miglior modo possibile rispetto alle varie funzionalità/requisiti funzionali.

1. Architettura concreta.

In questo step andiamo a vedere come le componenti logiche interagiscono (quali protocolli etc.) e quali sono le soluzioni tecnologiche che ci permettono di realizzare i componenti logici

1. Architettura di deployment.

In questo step ci concentriamo sui componenti della struttura componenti e connettori deployati nell’environement (componenti software che devono essere installati sull’hardware).

1. Architettura di implementazione (non trattata in questo progetto).

In questo step si scelgono, per quanto possibile, tecnologie, piattaforme, linguaggi di programmazione, …

Importante, in ogni caso, è considerare sin da subito i vincoli che ci vengono imposti.

Per rappresentare il sistema, in ogni caso, faremo uso di viste rappresentate tramite il modello UML.

Attributi di qualità

**Un’architettura software**, se scelta in maniera corretta, **deve soddisfare degli attributi di qualità**. Un attributo di qualità è una proprietà misurabile o verificabile di un sistema che viene utilizzato per indicare il grado di soddisfazione delle esigenze delle parti interessate. Importante è evidenziare, sin da subito, che spesso gli **attributi di qualità dipendono dall’ambiente e dai vincoli** che esso detta.

In generale, comunque, non è possibile garantirli tutti in un sistema, bisogna comprendere quali sono gli aspetti predominanti e selezionare soluzioni architetturali che sono promotrici di questi requisiti. L’insieme dei requisiti più importanti è l’ASR (Architectural Significant Requirements).

NB: gli ASR possono essere sia vincoli, che requisiti funzionale, che requisiti non funzionali.

In ogni caso, come abbiamo detto prima, queste qualità devono essere misurabili (es: veloce cosa vuol dire? Ciò che è fast per una tartaruga non lo è per un coniglio) (es: un sistema si deve modificare nella parte di rete per supportare il 5G e le successive tecnologie). Inoltre, dobbiamo riuscire a associare a determinati problemi solo una dimensione di qualità e senza avere ambiguità nella scelta (es: problemi di terminologia: un input cosa è? Un attacco al sistema, un evento, un input errato da gestire? Boh)

Strutture

Le **strutture** sono la parte fondamentale dell’architettura SW. Una struttura è un **insieme di elementi** **tenuti insieme da una relazione**. Essa diviene **architetturale** se supporta il ragionamento sul sistema e le sue proprietà.

Possiamo, quindi, ridefinire il concetto di **architettura come un insieme di strutture**, ognuna di esse è un insieme di elementi.

Nella complessità dei sistemi moderni, ci avvaliamo di **tre tipologie di strutture sulla base dell’aspetto da modellare** (catturano diversi aspetti del sistema, le uso per verificare aspetti di qualità diversa):

* **Moduli**

Partizioniamo il sistema in unità di implementazione chiamate moduli, che sono statici e si focalizzano su come le funzionalità del sistema sono raggruppate.

Abbiamo elementi software (classi, layers, gruppi di funzionalità) legati da relazioni.

A diagram of a computer

Description automatically generated

**NB**: evidenziamo, a questo livello, semplicemente quali moduli interagiscono ta loro.

**Domande**:

* + *Qual è la responsabilità funzionale primaria assegnata a ciascun modulo?*
  + *Quali altri elementi software può utilizzare un modulo?*
  + *Da quali altri software dipende?*
  + *Quali moduli sono legati ad altri moduli da relazioni di generalizzazione o specializzazione?*

**Cosa aiuta**: La modificabilità, perché più “spacchetto” più la aiuto.

**Sottostrutture***:*

*A close-up of a document

Description automatically generated*

**NB**: nelle sottostrutture i moduli sono gli stessi, ma sono rappresentati in modi diversi per comprendere se tutti i vincoli vengono soddisfatti.

* **Componenti e connettori**

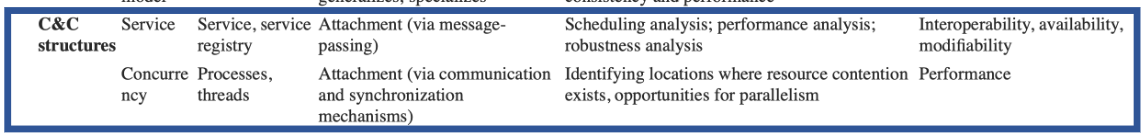
Le **istanze dei moduli sono le componenti**. Componenti e connettori mi forniscono un’idea della dinamicità del sistema: come i vincoli ed i requisiti non funzionali vengono soddisfatti a run-time? Questa struttura si concentra su come i componenti servizi, peers, clients, servers, filters) interagiscono tramite connettori (call-returns, process synchronization, pipes, protocolli, API rest, …)

**Domande**:

* + *Quali sono i principali componenti di esecuzione come interagiscono in fase di esecuzione?*
  + *Quali sono i principali archivi di dati condivisi?*
  + *Quali parti del sistema sono replicate (per garantire certe qualità)?*
  + *Come si muovono i dati nel sistema?*
  + *Quali parti del sistema possono essere eseguite in parallelo?*

**Cosa aiuta**: Performance, sicurezza e disponibilità.

**Sottostrutture**:



* **Strutture di allocazione**

Esse descrivono il mapping tra le strutture software e l’environnement (tra strutture software e strutture non-software come CPUs, file systems, networks, development teams). Possono essere utilizzate sia per i moduli che per i componenti.

Possono essere (*e nota se ci riferiamo a moduli e componenti*):

* + **Organizzativa**

A quale team, per esempio, assegnare lo sviluppo di un modulo.

* + **Di sviluppo**

Come, per esempio, dividere un modulo in files o dove salvarli (github per esempio)

* + **Di deployment**

Quale componente, per esempio, deployare e su quale macchina (server, etc.).

* + …

**Domande**:

* + *Su quale processore viene eseguito ogni elemento del software?*
  + *In quali directory o file viene memorizzato ogni elemento durante lo sviluppo, test e la costruzione del sistema?*
  + *Qual è l'assegnazione di ciascun elemento software ai team di sviluppo?*

**Cosa** **aiuta**: Performance, Sicurezza, Disponibilità, ma anche la valutazione di costi: se devo affidare aspetti a team interni/esterni.

**Sottostrutture**:

A close-up of a document

Description automatically generated

**NB**: possiamo notare come, nelle strutture, abbiamo sempre elementi e connessioni tra essi.

Le **strutture**, comunque, **sono in relazione**, rappresentano come abbiamo detto, lo stesso sistema sotto punti di vista diversi.

In generale, comunque, **le strutture sono “un’idea”.** La **rappresentazione “su carta” di una struttura è chiamata vista.** Un formato comune per rappresentare la vista è il modello UML.

**NB**: tutto ciò che non ha un vincolo non lo tratto a livello architetturale!!! Nell’architettura non tratto elementi superflui, li tralascia (ho astrazione). Dire, per esempio, come le classi comunicano all’interno di un modulo è compito del design concreto.

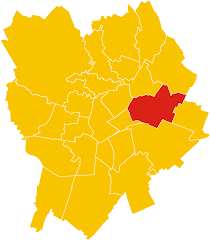
**NB**: la documentazione, soprattutto architetturale, è, in generale, importante per permettere di mettere mani ad un sistema.

Studio iniziale

Assumptions

Siccome ci sono state date poche informazioni dalla traccia, procediamo facendo delle assunzioni che poi saranno da verificare con gli stakeholders:

* Assumiamo di lavorare, in termini di norme vigenti, in Italia e alle situazioni contrattuali Italiane;
* Assumiamo che la centralina sia vista come un “blocco unico” con i suoi sensori. Tale centralina ha un certo livello di “intelligenza” software, quale:
  + Conversione di segnali analogici dei sensori in digitale ed invio;
  + Accettare dei comandi di attacco e stacco di corrente;
  + QUI CHE TIPO DI ALTRE ROBEPER ANOMALIA TIPO TEMPERATURA
* Supponiamo di operare nell’area del quartiere di Borgo Palazzo del comune di Bergamo, in cui sono presenti, secondo i dati dell'Autorità di Regolazione per Energia Reti e Ambiente (ARERA), al 31 dicembre 2022, 10.800 unità abitative con contatore elettrico attivo.



Sempre secondo l’ARERA, tal quartiere fa riferimento ad una sola centrale di distribuzione elettrica, assunzione che faremo per questo progetto.



<https://shorturl.at/kwxI4>

* Le centraline a cui si riferisce la traccia assumiamo che facciano riferimento a centraline domestiche, dunque contatori nelle case delle persone, la cui potenza di picco massima è di 3kw;
* Con situazioni anomale ci riferiamo a:
  + picchi di potenza rispetto ai limiti previsti per la centralina. Come limiti utilizziamo quelli forniti da e-distribuzione, la più grande società in Italia nel settore della Distribuzione e Misura di energia elettrica al servizio di oltre 31,5 milioni di Clienti connessi alla rete. Un limite è indicato come un 10% superiore rispetto alla potenza impegnata, sottoscritta a livello contrattuale. Per esempio, per un contratto da 3 kWh è possibile prelevare senza limiti di tempo fino a 3,3 kW. Inoltre, se si superano i 3,3 kWh, viene data la possibilità di prelevare fino a 4 kWh per tre ore, a tre ore scatta la segnalazione. Se si preleva una potenza superiore ai 4 kWh, il dispositivo per il controllo della fornitura di energia elettrica scatta a due minuti. [shorturl.at/rHOT5](https://shorturl.at/rHOT5).



* ???????????? ALRI
* Dopo 4 minuti senza possibilità di prelievo di dati, la centralina viene segnalata come guasta.
* Il sistema interroga la centralina ogni 20 secondi per prelevare i dati;
* Il sistema, ogni 60 secondi, media gli ultimi dati (cercando di evitare dati “spuri”) e verifica se è presente un’anomalia (anomalia segnalata dai dati).
* ??????????????quantità di informazioni report in byte e banda;
* Tutte le notifiche del sistema vengono inviate istantaneamente quando generate, ma in modo asincrono;
* L’operatore viene identificato geograficamente da latitudine e longitudine (posizione), questo perché può anche non trovarsi in un edificio e, dunque, non

Problem architecture

Introduction

L’architettura del problema deve **definire**:

* Le **funzionalità che il sistema deve offrire agli attori esterni** (requisiti funzionali)

**Digramma**: casi d’uso UML

* I **tipi di informazioni/dati che il sistema deve gestire per la realizzazione delle funzionalità**. Ormai tutti i sistemi lavorano sui dati in cui devo esprimere vincoli, etc.

**Diagramma**: diagramma delle classi UML

* I **flussi informativi che il sistema deve supportare per realizzare le funzionalità definite al punto 1 operando sui dati definiti al punto 2** (in/out). Come l’informazione evolve all’interno del mio sistema.

**Diagramma**: diagramma delle attività UML

* Rappresentare **i flussi di controllo che il sistema deve supportare per garantire le funzionalità**. Perché certe cose accadono? Why?

**Diagramma**: diagramma delle attività UML

**NB**:

* uno stesso diagramma con una certa sintassi può avere più semantiche (*es: possiamo usare il diagramma delle classi per modellare sia i dati che i componenti conctreti*);
* abbiamo sistemi fortemente basati sui dati che sono quindi da trattare a livello architetturale.

Si cerca di dare risposta a delle **domande per catturare gli elementi fondamentali** del problema:

* **Aspetti statici.** Sono quelle domande che mi permettono di identificare, poi, i moduli.
  + **Who**. Chi fa parte del sistema? Chi sono gli attori?
  + **What**. Quali sono i dati? Quali entità costituiscono il sistema?
  + **Where.** Dove sono i dati ed il chi del sistema? L’utente è seduto al terminale o iterante con un tablet?
* **Aspetti dinamici**
  + **How.** Come avvengono le trasformazioni? Come si fa a manipolare l’informazione affinché il sistema faccia quello che è richiesto?
  + **Why**. Perché certe cose accadono? Perché le attività vengono attivate? Quali sono i meccanismi dietri l’inneso?
  + **When.** Quando le cose devono essere fatte? (delay, timing, …) Con quale frequenza?

Use case diagram - "WHO" specification

link here

Il Diagramma dei Casi D’uso è un diagramma UML comportamentale, dedicato alla descrizione delle funzioni e dei servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Questo diagramma, seppur spesso sottovalutato, è molto importante perché influenza l’analisi e la progettazione orientata agli oggetti.

Attore è tutto ciò che do per scontato già esistente e che in alcun modo devo implementare. Gli attori non sono solo esseri umani, ma anche pezzi di software: se sfruttiamo i servizi o lo comandiamo noi è così. Gli attori sono tutti entità esterne, as-is: lo sfruttiamo.

NB: non esiste l’attore sistema!!!!! Il sistema è tutto il pezzo di software, è l’insieme dei casi d’uso.

* Where per gli attori.

Esso è importante per gli attori: una persona in movimento o meno influenza diversamente il sistema (Se l‘attore è in movimento, per esempio, posso pensare ad un sistema di notifiche tramite vibrazione). Un altro aspetto importante da considerare è se l’attore è interno o esterno al sistema stesso: se l’attore non è interno al sistema devo capire che interfacce di comunicazione avere con lui (es: in Google Maps l’utente è interno al sistema e fondamentale: geolocalizzarlo è il cuore della recensione).

NMB: se l’attore è interno al sistema necessito di un dato che me lo modella semanticamente. (es: i ripetitori in un sistema wifi sono attori, ma sono interni perché fondamentale e modellati dalla loro posizione = dato)

Conclusion

The team

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

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