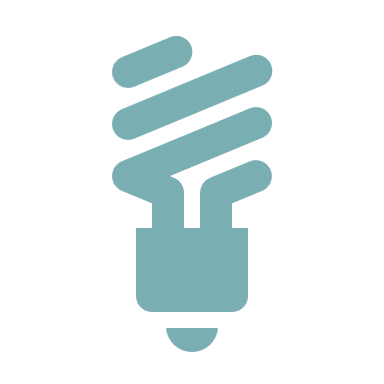
**Distribution boxes of electric power** - Software Architecture

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 7](#_Toc154425591)

[Acronyms used in the report 7](#_Toc154425592)

[Assumptions 7](#_Toc154425593)

[Problem architecture 8](#_Toc154425594)

[Introduction 8](#_Toc154425595)

[Use case diagram - "WHO" specification 9](#_Toc154425596)

[The team 12](#_Toc154425597)

Introduction & Theory

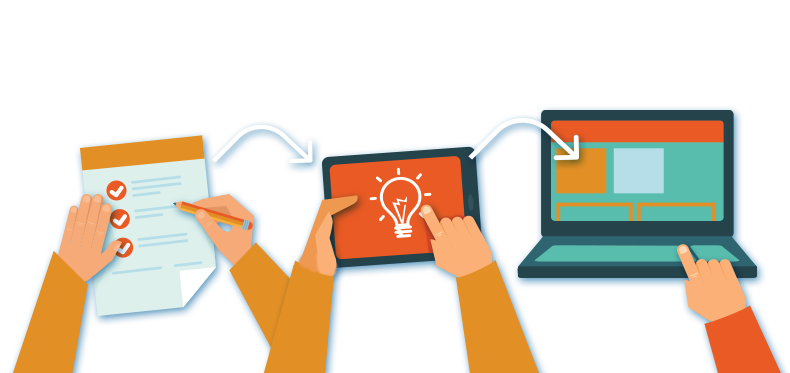
What is software architecture?

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 ci evidenzia 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.



**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.

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

**Sottostrutture***:*

*A close-up of a document

Description automatically generated*

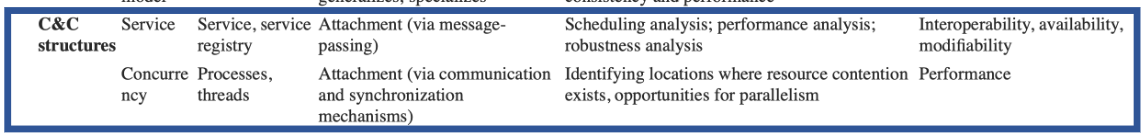
**NB**: nelle sottostrutture i moduli sono gli stessi, ma sono rappresentati in modi diversi per comprendere se tutti i vincoli vengono soddisfatti.

Le viste dei moduli sono comunemente mappate alle viste dei componenti e dei connettori. Le unità di implementazione mostrate nelle viste modulo hanno una mappatura con i componenti che vengono eseguiti in fase di esecuzione

* **Componenti e connettori**

Le **istanze dei moduli sono le componenti**. Componenti e connettori ci 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, …)

**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:

* + **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.).

* + …

**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.

Funzionalità

Le **funzionalità/requisiti funzioni sono alla base di un sistema**. Tuttavia, spesso i sistemi vengono modificati perché non vengono rispettati dei requisiti non funzionali (requisiti di qualità).

In generale, comunque, abbiamo:

* **Requisiti funzionali**. Questi requisiti evidenziano cosa il sistema deve fare e come deve reagire a degli stimoli;
* **Requisiti di qualità**. Questi requisiti esprimono requisiti di qualità sugli aspetti funzionali (*es: quanto veloce deve performare un’operazione o quanto è resistente agli errori*) o sul sistema nella sua interezza (*es: tempo di sviluppo*);
* **Vincoli**. Sono scelte imposte dall’architetto (es: usare Ruby perché tutti i programmatori lo usano) oppure scelte dettate dall’ambiente (es: l’azienda fa solo web app, l’azienda usa solo certi frameworks, …).

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).

In ogni caso, come abbiamo detto prima, queste qualità devono essere misurabili. Inoltre, dobbiamo riuscire a associare a determinati problemi solo una dimensione di qualità e senza avere ambiguità nella scelta.

How to identify a viable architectural solution?

Una tecnica (tra le tante) per progettare adeguatamente una soluzione architetturale 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 “dividere” 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 punto ci concentriamo sui componenti della struttura componenti e connettori deployati nell’environnement (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, come già evidenziato, di viste rappresentate tramite il modello UML.

Project track

Procediamo riportando la traccia del progetto…

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.

Studio iniziale

Acronyms used in the report

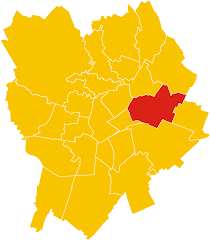
Forniamo una lista di acronimi utilizzati in questo report per avere, in ogni punto di questa relazione, chiari i riferimenti:

1. SA, Software Architecture;
2. kWh, kilowattora;
3. ECU, Electrical Control Unit.

Assumptions

Siccome ci sono state date poche informazioni dalla traccia, procediamo facendo delle assunzioni (nella realtà 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.
* 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).
  + Surriscaldamento.

Solitamente la temperatura di funzionamento di un ECU è tra i [-25%; +55%] gradi per le ECU di e-distribuzione. Per questo motivo una temperatura oltre il range fa scattare, dopo i classici tre controlli, la segnalazione <https://shorturl.at/cptO5>;



Questa assunzione è stata fatta perché, per un semplice problema di “picco di potenza” non avrebbe senso attivare l’uscita di un operatore (richiesta dalla traccia).

* Il sistema centrale interroga la centralina ogni 60 secondi per prelevare i dati utili alla definizione di nuove politiche;
* Il sistema in funzione sulla centralina, ogni 60 secondi (3 “blocchi” di dati generati), media gli ultimi dati (cercando di evitare dati “spuri”) e verifica se è presente un’anomalia (anomalia segnalata dai dati).
* Tutte le notifiche di anomalia del sistema vengono inviate istantaneamente quando generate, ma in modo asincrono;
* L’operatore viene identificato geograficamente da latitudine e longitudine (posizione) fornita da uno strumento GPS, questo perché può anche non trovarsi in un edificio e, dunque, non ha a disposizione un indirizzo di riferimento;
* Lavorando in Italia ed avendo associata una ECU per unità abitativa, andiamo ad associare in relazione 1:1 una ECU, che comunque verrà identificata con un’id interno al sistema, ad un indirizzo;
* Ogni 30 giorni avviene lo studio dei dati raccolti per la definizione di nuove politiche, dove per politica intendiamo l’inserimento, lo spostamento o il potenziamento della centrale di distribuzione.

… altre assunzioni meno importanti vengono esposte contestualmente alle viste esplicative.

Problem architecture

Introduction

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

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

**Digramma**: diagramma dei 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?

**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 concreti*);
* 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 + “WHERE” for the actors

link here

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.

Per gli attori è, inoltre, importante specificare il where: una persona in movimento o meno influenza diversamente il sistema. 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 dobbiamo capire che interfacce di comunicazione avere con lui.

NMB: se l’attore è interno al sistema necessito di un dato che me lo modella semanticamente.

Immagine che contiene testo, diagramma, schermata, disegno

Descrizione generata automaticamente

Andiamo a specificare del razionale per permettere comprendere a pieno le scelte meno intuibili dal diagramma.

|  |  |
| --- | --- |
| **Element** | **Responasability** |
| Data acquisition | Il sistema centrale acquisisce periodicamente i dati dalla centralina e ne mantiene uno storico persistente; |
| 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”.  *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 | Il sistema, se invocato dal “Central Technical Service”, permette l’identificazione di un operatore utilizzando alcune features. La posizione geografica real-time viene richiesta alla componente intelligente in uso dal tecnico tramite GPS. L’identificazione può terminare in una richiesta di intervento al tecnico (Sending action report). |
| Policy optimization | Il sistema, periodicamente, consulta lo storico persistente per delineare nuove politiche di distribuzione delle centraline, etc. ed, eventulmente, suggerisce alcune modifiche, tramite notifica (Notification policy), al “Central Technical Service”. |

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

link here

Per aiutarci a comprendere cosa è il modello dei dati, andiamo a differenziarlo dal modello di dominio. Il modello di dominio cerca di catturare con delle entità quello di cui stiamo parlando. È un concetto più ampio che rappresenta la comprensione approfondita di un particolare dominio o settore di attività. Si concentra sulla comprensione delle entità, dei concetti, delle regole e delle relazioni che caratterizzano il dominio in questione.

*Es: in un programma di foto avrò la foto, la foto ritoccata, la fotocamera, etc. Qui, inoltre, possiamo anche mettere le funzionalità: scatta foto, etc.*

Quando facciamo un **modello di dati**, invece, stiamo semplicemente trattando l’informazione: il dato. Il modello dei dati è una parte specifica del modello di dominio e si concentra principalmente sulla struttura dei dati utilizzati in un sistema o in un'applicazione. Avrò solo i dati di transito, tutto il resto lo tolgo. Qui, inoltre, **ho attributi che non hanno visibilità o tipo.**

*Es: non avrò il concetto di fotocamera, bensì la foto, la foto ritoccata, etc.*

Alcune avvertenze:

* gli attributi non hanno né qualità, né tipo;
* si possono anche modellare le relazioni tra posizioni;
* Un attore può essere portato all’interno del sistema modellandolo come un dato, con una rappresentazione che sia necessaria ai miei fini. Per esempio, mi interessa sapere l’origine dei dati possiamo associare un dato che è l’attore stesso.
* Per fare vedere che un dato è diverso a seconda del momento in cui viene trattato (più evoluzioni) si usa la **generalizzazione**;
* Possiamo **aggiungere le unità di misura**;
* Posiamo **aggiungere delle labels per caratterizzare meglio il dato**. Per esempio, specificare la dimensione in termini di bytes del dato ci permette di pensare, successivamente, a quanta banda è necessaria per il trasferimento. Importante è anche, per esempio, specificare la precisione per comprendere quanto possiamo tollerare i ritardi o la perdita di informazione: cambiare precisione ci obbliga, spesso, a cambiare tipo di sistema;
* Possiamo specificare anche **l’astrazione tramite etichetta**: da che aspetto di sistema vediamo l’oggetto per “chiamarlo” in un certo modo? A che livello di astrazione siamo (la semantica cambia)? Questa informazione è utile, poi, quando facciamo il partizionamento del sistema in moduli.

Immagine che contiene testo, diagramma, schermata, Piano

Descrizione generata automaticamente

|  |  |
| --- | --- |
| **Element** | **Responasability** |
| ECU | ECU è un attore modellato come un dato, fondamentale per conoscere l’origine del dato (ECUData), ma anche avere sempre il controllo del riferimento notifica – centralina. Ha un id univoco ed un indirizzo, anch’esso univoco, per conoscere l’esatta posizione di essa e permettere gli interventi nonché lo studio per la definizione di nuove politiche. |
| ECUdata | ECUData è il dato vero e proprio generato dai sensori della centralina (trasformato in digitale dalla componente intelligente della centralina). Il dato è composto, ad un certo timestamp, di temperatura e kilowatt/ora, entrambi utili a studiare le anomalie.  I dati temporaneamente si trovano sulla parte software del sistema in funzione sulla centralina (InECU) per lo studio dell’eventuale anomalia (vengono utilizzati i dati medi provenienti da tre estrazioni). Una volta studiati, questi dati, vengono dimenticati. *Lo evidenzieremo anche successivamente, ma è fondamentale che un “pezzo software” sia in funzione sulla centralina poiché in caso di anomalia, ma in presenza di problemi di comunicazione col server centrale, la centralina deve avere comunque la possibilità di essere disattivata.*  I dati vengono, ogni 60 secondi, prelevati dal sistema centrale per essere salvati nel dataset centrale (InSystem) e permettere studi successivi. Il blocco dati in transito riguarderà un migliaio di bytes che devono essere inviati con una alta precisione: non vogliamo un sistema rigidissimo, ma è importante che questi dati arrivino precisi, con perdite basse e latenza minima.  Questa semantica l’abbiamo espressa attraverso una generalizzazione che permette di comprendere come il dato, anche se è lo stesso, ha un significato diverso ed è trattato diversamente in base a dove si trova. |
| Notice | Tutti gli attori del nostro sistema si scambiano notifiche di un migliaio di bytes: è lo strumento di comunicazione centrale del nostro sistema. Tutte le notifiche sono associate ad un’ECU. Quelle, inoltre, che si interfacciano con un operatore ne avranno uno associato (al quale possono giungere più notifiche).   * AnomalyNotice che riguarda la notifica dell’anomalia al Central Technical Service deve avere un’alta precisione. La notifica deve essere segnalata prontamente e con minima perdita. Possiamo, comunque, tollerare piccolissime perdite: non si parla di sistemi critici. Qui abbiamo, ovviamente, un’anomalia associata. * Anche InterventionNotice, ovvero la segnalazione di anomalia al tecnico, deve avere un’anomalia associata e latenza bassa, benchè possiamo tollerare una precisione minore su questo tipo di notifica. * Le notifiche di inizio e fine intervento sono quelle di cui possiamo tollerare minor precisione, in quanto sono solo segnalazioni tecnico-Central Technical Service con pochi dati importanti.   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).  NB: non è necessario includere questo dato all’interno delle notifiche di inizio e fine intervento. |
| Operator | Ogni operatore ha i dati identificativi classici, nonché la conoscenza tecnica che lo contraddistingue (per operare) e un flag binario di disponibilità in quel dato istante.  Ad ogni operatore è associata una posizione (OperatorPosition) fornita da uno strumento GPS, motivo per cui abbiamo necessità di latitudine e longitudine (non afferisce in un dato momento ad un’abitazione, bensì ad un lungo nello spazio 2D).  La relazione tra Operator e OperatorPosition è una N-1 perché non ci interessa tenere uno storico di posizioni (non ha senso, non vogliamo N:N), ma l’operatore avrà sempre e solo la posizione associata nel dato momento fornita dal sistema GPS. |
| 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.