MIC1-SYS

Paolo Amato

Sommario

1) Processo di sviluppo

Per la realizzazione del sistema software da presentare in sede d'esame si è deciso di adottare un processo di sviluppo del tipo **Agile UP**, sviluppato da **Scott Ambler**. In sostanza, questo significa che si farà uso del framework che **UP** mette a disposizione e di conseguenza dei concetti da esso forniti come gli artefatti da creare, le differenti discipline, le fasi di ideazione ed elaborazione e così via. Tuttavia, tale framework viene utilizzato focalizzandosi su **un modo di lavorare agile**, vale a dire:

- Approcciare la realizzazione del software mettendo da parte l'idea di poter avere a disposizione, sin dalle prime fasi del progetto, di tutti i requisiti software funzionali e non da implementare. È importante rispettare le tre caratteristiche fondamentali di un tale processo di sviluppo: il software va sviluppato in modo incrementale, iterativo ed evolutivo. **I requisiti dovranno evolvere con il passare delle iterazioni.**
- **Eseguire in maniera frequente dei workshop di breve durata.** Questa caratteristica si presta evidentemente ad un lavoro di gruppo. Tuttavia, pur essendo da solo, ho approfittato di questi momenti per individuare nuovi requisiti, problemi nei requisiti già presenti, eventualmente andando a ragionare su possibili problematiche in termini di progettazione ed implementazione
- **Modellare secondo lo spirito agile.** Conformarsi al modo di lavorare agile non significa affatto non effettuare modellazione. L'obiettivo della modellazione è quello non di documentare, ma di aiutare a comprendere. Secondo questo spirito, la modellazione relativa ad aspetti meno importanti del sistema software risulterà leggera o completamente assente, mentre si riserverà una maggiore attenzione ed importanza a quegli aspetti del sistema che possono essere ritenuti più critici.
- **Concentrazione su attività ad elevato valore.** Sulla base di quanto già accennato con la modellazione agile, in generale si può affermare quanto segue: l'attenzione deve

concentrarsi sulle attività che contano realmente, e non su ogni cosa possibile che possa accadere durante un progetto.

- **Fare uso dello strumento denominato GitHub**, di modo tale da poter controllare in maniera efficace le diverse versioni degli artefatti di volta in volta prodotti e, dunque, gli output delle diverse iterazioni

Non ho potuto ovviamente non tenere conto della mia inesperienza pratica nel gestire la costruzione di un sistema software secondo quelle che sono le pratiche fondamentali di questo processo di sviluppo software, motivo per cui molto spesso ho dovuto ragionare attentamente sul corretto modo di procedere ed operare, realizzando, ogni volta, quella che mi è parsa la scelta più giusta.

Per concludere questo paragrafo riguardante la descrizione del processo di sviluppo, brevemente andrò ad indicare le pratiche agili fondamentali alle quali ho fatto riferimento:

- **Sviluppo iterativo,incrementale ed evolutivo**
- **Design semplice**
- **Versioning**
- **Refactoring**
- **Prioritization**: Lo sviluppo della soluzione può cominciare solo dopo aver messo in priorità gli obiettivi, dai quali deriveranno i requirements e le features
- **Semplicità**: semplicità nel codice, semplicità nella documentazione, semplicità nella progettazione, semplicità nella modellazione; i risultati così ottenuti sono una migliore leggibilità dell'intero progetto ed una conseguente facilitazione nelle fasi di correzione e modifica;

2) Fase di avvio del progetto

2.1) Vision

Il principale scopo del sistema software che si vuole costruire è quello di **emulare una architettura hardware**. Ciò che fondamentalmente il sistema fornisce è un emulatore, vale a dire una applicazione che, dalla prospettiva comportamentale, funziona in maniera del tutto analoga al dispositivo hardware che si vuole emulare. Ponendo attenzione al fattore complessità, e alle conoscenze pregresse di cui dispongo, sono giunto alla decisione di produrre un sistema software in grado di **emulare la IJVM e in particolare il MIC-1**, un processore inventato da Tanenbaum a scopo didattico in ambito universitario. Si tratta di una architettura da me studiata al corso di Calcolatori Elettronici II, che viene presentata per due motivi essenziali:

- mostrare come, usando elementi logici di base, sia possibile realizzare una microarchitettura che implementi un semplice ma completo set di istruzioni;
- mostrare come anche la realizzazione di un sistema hardware apparentemente complesso, come un processore, si riduca in realtà alla progettazione di un'unità operativa e di un'unità di controllo, e del modo in cui devono comunicare.

Un emulatore di una generica architettura hardware, in quanto tale, deve mettere a disposizione tutte le funzionalità dello strato hardware che sta emulando. Si tenga presente che, in generale, non è necessario, come già sottolineato in precedenza, realizzare una emulazione a livello elettrico, in termini di porte logiche, componenti combinatori e sequenziali. Lo stesso risultato, infatti, può essere ottenuto analizzando il funzionamento del dispositivo in termini comportamentali. Si terrà conto di entrambe le prospettive: quella elettrica infatti risulta utile per capire come i vari componenti del datapath della CPU collaborano consentire l'esecuzione delle istruzioni. Sarà, a questo punto, sufficiente replicare il comportamento dell'architettura hardware attraverso opportuni strumenti tecnologici e sfruttando una macchina sulla quale sarà presente l'ambiente di esecuzione attraverso il quale verrà eseguito lo strato software che sintetizza l'emulatore.

Più nel dettaglio, **si vuole produrre non soltanto un emulatore per tale architettura, ma anche un assemblatore** che sia capace di tradurre il codice sorgente scritto dall'utente in codice macchina eseguibile proprio dall'emulatore. Ovviamente, l'utente sarà "obbligato" a scrivere codice secondo quello che è il modello di programmazione del processore MIC-1.

L'utente deve poter interagire con l'assemblatore e costruire l'eseguibile da dare in pasto all'emulatore secondo quelle che sono le opzioni messe a disposizione dal sistema stesso.

All'utente quindi, sarà reso possibile utilizzare una qualsiasi istruzione appartenente all'ISA della CPU considerata.

Inoltre, l'utente potrà stabilire se caricare sull'emulatore un programma scritto con l'assemblatore appartenente al sistema software oppure un programma già compilato.

Si tenga presente che la IJVM è un processore virtuale a tutti gli effetti, quindi ha il processore, ossia il MIC-1, la memoria, un ambiente di esecuzione, e così via.

L'emulatore dovrà occuparsi non soltanto dell'effettiva esecuzione di istruzioni e dunque di linee di codice. A tale componente è affidata anche la visualizzazione, mediante un'interfaccia grafica, di alcune informazioni interne al dispositivo hardware emulato:

- **Comportamento del processore**
- **Comportamento dei registri**
- **Comportamento dell'unità di controllo**
- **Memoria principale**
- **Memoria di controllo**
- **Struttura dello stack e informazioni in esso salvate**

Per concludere il documento di "Vision", si può in definitiva affermare che **è centrale il desiderio di produrre un emulatore affidabile, ma è importante anche curare in maniera opportuna aspetti di contorno per l'emulatore stesso**, che in ogni caso hanno una loro importanza, soprattutto nell'ottica di facilitare l'utente a comprendere il modo di funzionare del sistema emulato.

2.2) Documentazione delle specifiche supplementari

In questa sezione ci si focalizza su tutto ciò che non sarà compreso nei casi d'uso, in particolare vengono analizzati quelli che sono i **requisiti non-funzionali** che il sistema software dovrebbe soddisfare.

Una applicazione di questo tipo, il cui compito è quello di emulare una già esistente architettura hardware, si rivela davvero utile nel momento in cui la sua **logica di emulazione è indipendente dalla maniera in cui i programmi sono predisposti per l'esecuzione**. Sin dall'inizio sarà fondamentale andare a realizzare una architettura fatta da **componenti e/o sottosistemi che**

siano facilmente riusabili e modificabili, così da poter conformare l'interfaccia utente all'emulatore stesso, senza inficiare quella che è la sua logica di elaborazione.

L'approccio che intendo seguire richiede che il sistema soddisfi la proprietà della **modularità**, disaccoppiando i due sottosistemi fondamentali da cui il sistema software è costituito: emulatore e assemblatore. Difatti, l'assemblatore non deve generare codice interpretabile ed eseguibile esclusivamente dall'emulatore appartenente a tale sistema.

Si tenga presente che si sta adottando un processo di sviluppo software **iterativo, incrementale ed evolutivo**. Dunque, i requisiti non-funzionali di **manutenibilità ed evolvibilità** sono importanti da soddisfare**. Emulatore ed assemblatore dovrebbero essere suddivisi in moduli tra loro scarsamente accoppiati**, consentendo così di adoperare un tipo di sviluppo focalizzato sulle parti più critiche del sistema, per poi estendere quando fatto nelle prime iterazioni.

In generale, un emulatore potrebbe risultare scarsamente **usabile**, ragion per cui le scelte di progetto effettuate dovranno in qualche modo far sì che il sistema costruito sia il più **interattivo ed user-friendly** possibile.

2.3) Glossario

Vengono di seguito presentati dei termini che compariranno di frequente all'interno di tale documento, fondamentali per capire in maniera adeguata le parti più tecniche e specifiche del dominio applicativo. Comincerò mostrando tutte quelle parole significative del dominio utilizzate nella prima iterazione, inserendone poi altre con il proseguire dello sviluppo.

- **CPU:** Unità centrale di elaborazione, corrispondente in questo caso al processore MIC-1, del quale verrà implementato l'ISA.
- **Unità operativa:** Questo blocco incapsula la parte inerente all'elaborazione dei dati ottenuti in ingresso sulla base degli ingressi di controllo forniti dalla unità di controllo, e risponde fornendo uno stato che sarà utile alla unità di controllo per scandire le prossime operazioni da comandare al blocco di elaborazione. Al termine delle elaborazioni, in uscita verrà rilevato il prodotto di queste ultime.
- **Unità di controllo:** Il blocco a cui è delegato il controllo della parte operativa è costituito dall'unità di controllo. Non sussiste unità di controllo senza unità operativa; se non è stata dapprima sviluppata la parte da controllare, non ha senso sviluppare il controllo di tale parte.
- **BUS:** canale di comunicazione che permette a periferiche e componenti di un sistema elettronico di interfacciarsi tra loro scambiandosi informazioni o dati di sistema attraverso la trasmissione e la ricezione di segnali.
- **Stack:** è un tipo di dato astratto che viene usato in diversi contesti per riferirsi a strutture dati, le cui modalità d'accesso ai dati in essa contenuti seguono una modalità LIFO. Lo stack è un elemento dell'architettura dei processori, e fornisce il supporto fondamentale per l'implementazione del concetto di subroutine
- **RAM:** Random Access Memory, si tratta della memoria principale utilizzata dal processore

- **Opcode:** codice operativo in base al quale l'unità di controllo comanda l'unità operativa e stabilisce cosa quest'ultima deve fare
- **Istruzione:** tale concetto ingloba in sé due tipi di informazione: la codifica binaria dell'Opcode da far eseguire alla CPU, e la codifica binaria degli operandi da usare.
- **Programma:** Blocco di istruzioni che il MIC-1 è in grado di interpretare
- **Microprogrammazione:** tecnica specifica utilizzata per la realizzazione dell'unità di controllo. Si contrappone alla logica cablata
- **Memoria di controllo:** si tratta di una ROM utilizzata nelle unità di controllo costruite in logica microprogrammata. Essa contiene per ogni istruzione dell'ISA, la relativa microprocedura
- **Microistruzione:** parola ad n bit che fornisce i valori che i segnali di controllo, mandati dall'unita di controllo all'unità operativa, devono assumere
- **Microprocedura:** la sequenza di microistruzioni relativa ad una data istruzione dell'ISA costituisce la microprocedura che implementa quella istruzione.
- **ALU:** questo termine identifica la parte del processore che si occupa di svolgere calcoli matematici e operazioni logiche
- **Shift Register:** registro a scorrimento collegato in uscita all'ALU
- **PC:** registro interno alla CPU utilizzato per mantenere l'indirizzo della successiva istruzione da eseguire
- **IR:** registro della CPU che immagazzina l'istruzione in fase di elaborazione.
- **IJVM:** architettura di elaborazione che prende in considerazione il sottoinsieme delle istruzioni della JVM che lavorano soltanto sui numeri interi.
- **Architettura a stack:** si tratta di un particolare tipo di struttura di processore, nel quale si presuppone che se devo eseguire una qualsiasi istruzione, gli operandi di cui essa necessita si trovano in cima allo stack del processore stesso.

3) Specifica dei requisiti

In questa parte della documentazione l'attenzione è posta sulla realizzazione di artefatti rappresentativi di use-case che siano incentrati sugli obiettivi degli utenti che andranno a fare uso del sistema software in questione. Proseguendo, ci si focalizzerà sulle differenti funzionalità messe a disposizione dall'applicazione, utilizzabili dall'utente, anche se non collegate ad un determinato obiettivo o scopo che l'utente stesso intende perseguire.

3.1) Attori e obiettivi

Attori:

- **Utente**

Obiettivi utente:

- **Esegui Programma**
- **Assembla Codice**
- **Inizializza Emulazione**
- **Mostra Informazioni Architettura**

Di seguito viene riportata una descrizione dei casi d'uso secondo quello che in UP è definito "formato breve":

- **EseguiProgramma:** Lo sviluppatore, dopo aver selezionato un programma dal file system, potrà eseguirlo in modalità normale oppure step-by-step.

- **AssemblaCodice:** Consente di compilare il codice scritto generando nel file system locale un file assemblato che l'emulatore è in grado di eseguire.
- **InizializzaEmulazione:** Prima di caricare ed eseguire un programma, l'utente sarà in grado di vedere e modificare proprietà tipiche dell'emulazione come il microprogramma memorizzato nella memoria di controllo, le istruzioni ISA supportate, i valori iniziali dei registri ecc.
- **MostraInformazioniArchitettura:** Il programmatore potrà visualizzare informazioni su specifici elementi dell'architettura, di modo tale da verificare il comportamento di tali componenti architetturali.

3.2) Modello dei casi d'uso

3.2.1) EseguiProgramma

Nome caso d'uso	EseguiProgramma
Attore primario	Utente
Portata	Livello Utente
Parti interessate ed interessi	Scegliere il programma e mandarlo in esecuzione
Pre-condizioni	Modo di esecuzione selezionato
Post-condizioni	Visualizzazione dell'esecuzione del programma
Scenario principale	1) Scegli un programma dal file system 2) Avvia programma
Estensioni/Scenari alternativi	2a) L'utente mette in pausa il programma 2b) L'utente termina il programma

3.2.2) MostraInformazioniArchitettura

Nome caso d'uso	MostraInformazioniArchitettura
Attore primario	Utente
Portata	Livello Utente
Parti interessate ed interessi	Visualizzare come lo stato dell'architettura e dei suoi diversi componenti evolve
Pre-condizioni	Un programma è già stato selezionato ed avviato
Post-condizioni	Visualizzazione informazioni riguardanti lo stato del dispositivo hardware
Scenario principale	1) Scegli componente

Estensioni/Scenari alternativi	1a) Se viene scelta la CPU 1) Visualizza i valori dei registri 1b) Se viene scelta la memoria centrale 1) Visualizza stack,constant pool, method area 1c) Se viene scelta la memoria di controllo 1) Visualizza le microprocedure e le microistruzioni ad esse appartenenti

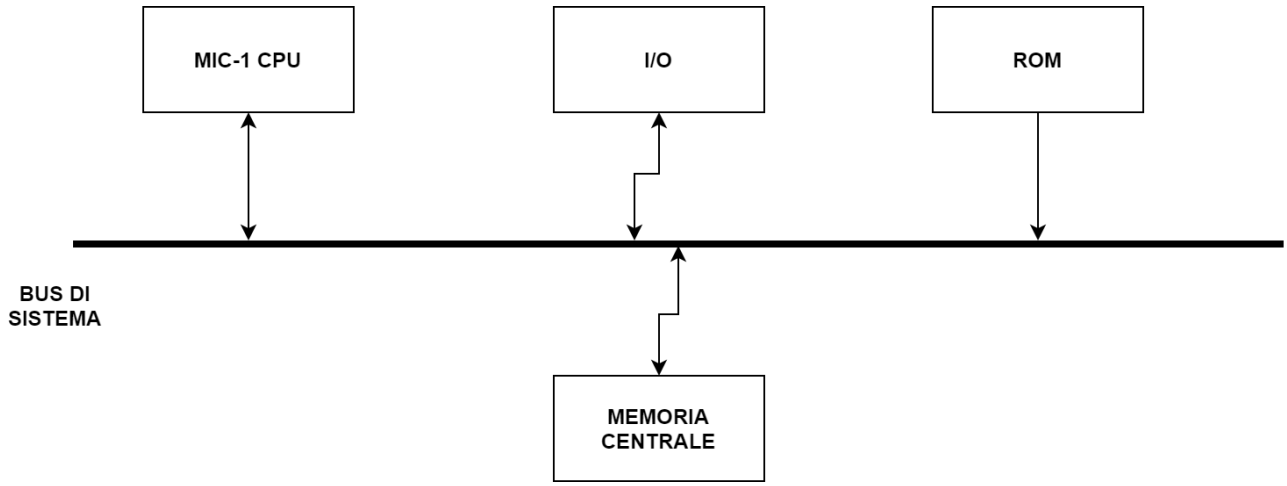
3.2.3) AssemblaCodice

Nome caso d'uso	AssemblaCodice
Attore primario	Utente
Portata	Livello Utente
Parti interessate ed interessi	Produrre un file eseguibile a partire dal codice scritto
Pre-condizioni	L'utente dispone di un file contenente codice
Post-condizioni	File eseguibile generato
Scenario principale	1) Carica file contenente codice 2) Compila codice
Estensioni/Scenari alternativi	2a) Se il codice è valido, viene restituito un file eseguibile 2b) Se il codice è errato, viene restituito un messaggio di errore

3.3) Lista delle funzionalità

3.3.1) Emulazione dell'ambiente IJVM

Questo è sicuramente un requisito funzionale, che presenta al suo interno uno specifico insieme di altre features. Il nostro interesse è rivolto all'emulazione di una CPU, in particolare del processore MIC-1, ma è evidente che un qualsiasi processore, preso così com'è, non possa funzionare correttamente. Ha bisogno senza dubbio di una memoria centrale con la quale effettuare scambi di dati in lettura e/o scrittura. Tra l'altro, il MIC-1 è costruito in logica microprogrammata, ragion per cui un ruolo di fondamentale importanza è svolto dalla memoria di controllo, che può essere considerata alla stregua di una ROM. Di seguito viene riportata, in forma semplificata, l'architettura della IJVM:



La ROM è stata riportata in maniera esplicita, sebbene sia da ritenere un componente interno alla CPU, appartenente alla sua unità di controllo. Tenere bene a mente questa architettura sarà di aiuto nel realizzare un componente emulatore la cui logica applicativa rispecchi in maniera adeguata l'hardware emulato. Scopo principale dello sviluppo secondo la proprietà dell'iteratività è capire quali sono gli elementi più importanti e focalizzarsi sulla loro comprensione, per poi passare allo sviluppo completo del sistema.

- Linguaggio assembly del MIC-1: L'architettura appena mostrata è dotata di un processore. Come per un qualsiasi altro processore, anche il MIC-1 ha un comportamento che può essere descritto attraverso il ciclo di Von Neumann. È possibile interagire con tale CPU sfruttando il suo modello di programmazione. Un processore altro non è che un interprete: presa in ingresso una istruzione, la esegue e fornisce in uscita dei risultati. Siamo interessati a capire come la CPU, grazie al suo modello, riesce a rispondere alle istruzioni ad essa fornite. È inoltre di rilievo andare ad esplodere la struttura della singola istruzione: siamo nel caso di un processore realizzato in logica microprogrammata; dunque è presente una memoria di controllo che, per ogni istruzione dell'ISA, memorizza la corrispondente microprocedura. Ogni microprocedura è fatta da un insieme di microistruzioni, e ogni microistruzione governa il datapath per un singolo ciclo di clock. Quando una microistruzione viene letta, ciascun segnale di controllo assume il valore del bit corrispondente.

- Linguaggio MAL: Scrivere a mano le microistruzioni che formano la microprocedura di una data istruzione dell'ISA è perfettamente possibile, ma è molto semplice commettere errori; inoltre, il microprogramma così ottenuto sarebbe tedioso da comprendere e modificare. Il linguaggio usato per semplificare la scrittura del microprogramma è denominato MAL (MicroAssembly Language). L'emulatore, come sappiamo, deve essere in grado di interpretare il linguaggio assembly del MIC-1, ossia deve saper interpretare le istruzioni del'ISA della IJVM. Questo significa che tale componente deve essere in grado di interpretare le microistruzioni delle diverse microprocedure e, dunque, codice scritto in linguaggio MAL.

3.3.2) Compilazione

Anche questo è un requisito funzionale. L'utente, alla consegna del sistema, potrà interagire con esso per generare un eseguibile interpretabile dall'emulatore dopo aver caricato dal file system un file contenente codice conforme al linguaggio assembly del MIC-1.

3.3.3) Interfaccia grafica

Il sistema software in questione deve essere capace di gestire dinamicamente le interazioni con l'utente, ragion per cui, nella lista delle funzionalità, compare l'interfaccia grafica. L'utente, alla consegna dell'applicazione, potrà interagire con il sistema per stabilire proprietà dell'emulazione, la modalità di esecuzione ecc. L'utente potrà caricare il file macchina da eseguire prelevandolo dal file system.

Le due disponibili modalità di esecuzione sono:

- Utente: in questa modalità l'unico scopo dell'utente è quello di poter usare l'emulatore come semplice esecutore di programmi, avendo soltanto la possibilità di visualizzare lo stack per prelevare i risultati di una elaborazione.
- Esperto/Programmatore: Questa è una modalità molto più tecnica. Si potrà non soltanto fare tutto ciò messo a disposizione nella modalità utente, ma anche vedere le informazioni

in continuo cambiamento nei registri della CPU, nella memoria principale, modificare il microprogramma nella memoria di controllo, utilizzare l'assemblatore e così via. Sarà possibile inoltre interrompere l'esecuzione, riprenderla, terminarla o eseguirla passo per passo. L'esecuzione step by step, che potrà essere fatta in termini di operazioni interne alle microistruzioni, singole microistruzioni, o singole istruzioni dell'ISA, è utile nel momento in cui si vuole analizzare nel dettaglio il comportamento del programma caricato da parte del programmatore.

Data una prima interfaccia di configurazione, in base alla modalità selezionata, verrà mostrata una seconda interfaccia la cui struttura dipende dal particolare modo di funzionamento selezionato (Utente o Programmatore).

4) Analisi dei requisiti

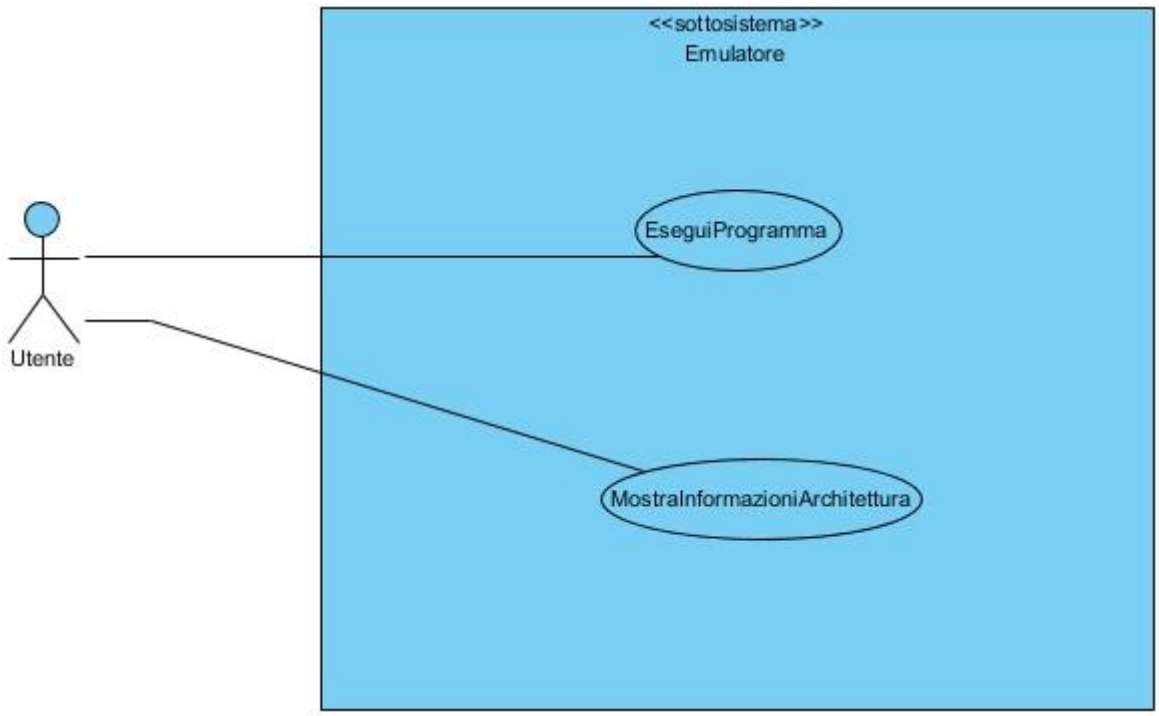
L'analisi dei requisiti è stata effettuata all'inizio di ogni iterazione e ogni volta si è fatto riferimento ad un sottoinsieme di requisiti funzionali da realizzare e requisiti non-funzionali da soddisfare e rispettare. Data la complessità del sistema da realizzare, alcuni requisiti hanno richiesto uno studio e un approfondimento sicuramente più corposi di altri.

4.1) Prima iterazione

La prima cosa da fare è stata individuare gli elementi critici del sistema software da dover sviluppare. Dall'inizio molta enfasi è stata posta sulla logica del componente emulatore appartenente ala sistema software che si sta costruendo. L'elemento principale dell'architettura e anche quello più critico è senza dubbio il processore. Prima di questo, però, il sistema è stato organizzato secondo una prospettiva statica e strutturato per soddisfare i requisiti non-funzionali.

4.1.1) Vista casi d'uso

MIC1-SYS si presta ad avere pochi casi d'uso associabili a degli obiettivi dell'utente. La prospettiva dei casi d'uso riportata di seguito è dunque molto scarna. E' fondamentale osservare che, come primo caso, d'uso si è preso in considerazione quello denominato "EseguiProgramma", che permetterà all'utente di poter eseguire un applicativo scritto secondo il linguaggio assembly supportato dal processore MIC-1.



4.1.2) Architettura logica

5) Design funzionale e non-funzionale

6) Prospettiva dell’implementazione

7) Testing