ELABORATO FASE B

Bertozzi Pietro

E-mail istituzionale: pietro.bertozzi3@studio.unibo.it (mailto:pietro.bertozzi3@studio.unibo.it)
GitHub organizzation gruppo progetto finale sito web: https://ingegneria-sistemi-software-m.github.io (https://ingegneria-sistemi-software-m.github.io)
GitHub organizzation gruppo progetto finale: https://github.com/ingegneria-sistemi-software-m (https://github.com/ingegneria-sistemi-software-m)
GitHub repository personale: https://github.com/ingegneria-sistemi-software-m/ingegneria-sistemi-software-m-pietro-bertozzi (https://github.com/ingegneria-sistemi-software-m/ingegneria-sistemi-software-m-pietro-bertozzi)
Diario principale: https://github.com/ingegneria-sistemi-software-m/ingegneria-sistemi-software-m-pietro-bertozzi/tree/main/riflessioni (https://github.com/ingegneria-sistemi-software-m/ingegneria-sistemi-software-m-pietro-bertozzi/tree/main/riflessioni)

Quello che lo scrivente ha percepito sia la finalità dei contenuti discussi in questa seconda fase, e quello che lo scrivente si aspetta sia lo scopo della prossima fase del corso

In questa seconda fase del corso ho colto come obiettivo principale l'introduzione e l'approfondimento del modello ad attori come paradigma di riferimento per progettare sistemi distribuiti. Rispetto a modelli più tradizionali, come quello a oggetti o event-driven, il modello ad attori mette in primo piano la nozione di entità autonome, in grado di gestire messaggi in modo indipendente e con logiche di comportamento legate al loro stato interno.
È un cambiamento di prospettiva che non riguarda solo la tecnica, ma anche il modo di pensare la progettazione.
Un altro aspetto centrale è stato quello legato alla relazione tra linguaggi e problemi: non basta saper risolvere un

problema, è importante anche con quale livello di espressività e chiarezza si riesce a descriverlo. Da qui la riflessione sull'abstraction gap e su come strumenti come DSL (in particolare QAK, nel nostro caso) possano aiutare ad avvicinare la descrizione del sistema al modo in cui lo immaginiamo.
Per quanto riguarda la prossima fase, al solito, lascio che il professore mi sorprenda.

Le nuove abilità / competenze che lo scrivente pensa di avere appreso (o che sia possibile apprendere) dopo questa seconda fase, sia in relazione alla parte 'pratica' del saper-fare (strumenti, tecnologie, etc.) sia con riferimento ad aspetti più 'concettuali' (metodologie, modelli, etc.)

Dal punto di vista pratico, in questa fase ho acquisito maggiore familiarità con il linguaggio QAK, che permette di descrivere un sistema distribuito in termini di attori, messaggi e contesti. Ho imparato a progettare e implementare sistemi composti da più attori, ognuno con il proprio comportamento reattivo, e a farli comunicare in modo ordinato e comprensibile.
Sul piano concettuale, ho consolidato una visione più matura della progettazione software. Ho capito che un modello non è solo un aiuto visivo o una documentazione, ma può (e deve) essere eseguibile: se un modello può essere compreso dalla macchina, allora è anche non ambiguo, preciso, e utile per il confronto. Ho riflettuto inoltre sull'importanza di scegliere il giusto metamodello in base al tipo di sistema da progettare: non sempre UML è la scelta migliore, e in alcuni contesti, come quelli trattati qui, un approccio basato su attori risulta decisamente più efficace.
Di fatto questo cambio di prospettiva apre moltisse possibilità e, appena avrò tempo di farlo, sudierò volentieri lo stato dell'arte a riguardo.
Sarò arrogante, ma ho l'impressione che sia un campo in cui è possibile innovare ulteriormente... devo/voglio studiare più approfonditamente la questione.

Gli aspetti di Ingegneria del software (key points) che il caso di studio ha permesso di richiamare o di introdurre nelle varie fasi ‘evolutive’ dei prototipi sviluppati durante la fase 2

Durante questa fase, un aspetto che mi ha colpito particolarmente è stata la differenza tra documentazione tradizionale e modello eseguibile.
In molti contesti, l’analisi del problema si ferma a descrizioni testuali o diagrammi, spesso ambigui e difficili da verificare. Con QAK, invece, abbiamo prodotto modelli che non sono solo rappresentazioni, ma sono già software, pronti per essere eseguiti e osservati.
Questo approccio ha reso subito evidente se il comportamento del sistema era coerente con le attese, eliminando i passaggi (inevitabilmente ambigui, siamo esseri umani) tra analisi, progettazione ed implementazione.
Il fatto che il modello sia interpretabile dalla macchina garantisce chiarezza e verificabilità, e consente di lavorare più vicino al dominio del problema, senza perdersi nei dettagli tecnici prematuri.
Personalmente, ho trovato questo modo di procedere molto più efficace e diretto: permette di progettare pensando al comportamento, non solo alla struttura, e riduce i fraintendimenti tipici delle fasi iniziali.

Opzionalmente: Il ruolo che lo scrivente ritiene abbia il concetto di linguaggio (di programmazione, ma non solo) più volte evocato dal docente come esempio di un salto evolutivo fondamentale dell’informatica rispetto al solo sviluppo di librerie

Il parallelismo tra linguaggi naturali e linguaggi di programmazione è perfettamente valido.
Un linguaggio, che sia di programmazione o naturale, non si limita a fornire una forma per esprimere idee, ma gioca un ruolo fondamentale nel processo stesso di creazione e formulazione delle stesse.
Persone che conoscono linguaggi naturali diversi tendono a pensare in modo diverso proprio a causa della struttura e dei vocaboli a loro disposizione.
Imparare nuovi vocaboli non significa solo migliorare l’espressione, ma anche affinare la granularità con cui

siamo in grado di cogliere le sfumature, e quindi di pensare criticamente.
Normalmente il linguaggio è il modo in cui pensiamo alle idee, ma per chiedersi in quale linguaggio è bene esprimere un'idee è necessaria un profonda familiarità con l'idea stessa.
Questo evidenzia quanto sia importante, ma anche complesso, sviluppare un linguaggio che soddisfi le esigenze specifiche di un determinato contesto.
Per quanto riguarda la differenza tra linguaggi e librerie invece; un linguaggio di programmazione è un sistema completo per descrivere algoritmi e strutture di controllo, mentre una libreria è un insieme di componenti già pronti, scritti in quel linguaggio, pensati per affrontare problemi specifici.
Le librerie estendono l'espressività del linguaggio in contesti concreti, ma non ne cambiano la natura.
Al contraria passare da un linguaggio ad un altro, specie se di derivazione diversa, può stravolgere completamente il paradigma, il modo di organizzare le idee e di procedere.
La scelta del linguaggio, in conclusione, è la scelta di mettere il primo piano alcuni aspetti, piuttosto che altri, rendendoli più semplici da cogliere ed esprimere.