

FP e UML: suggerimenti per sfruttare i diagrammi UML nella Function Point analisi

NB: Questo articolo contiene materiale estratto dal Manuale delle Regole di Conteggio IFPUG. Viene riprodotto in questo documento con permesso dell'IFPUG.

Introduzione: FP e UML come elementi comuni che prescindono dalla tecnologia.

I Function Point sono la metrica funzionale più diffusa al mondo (per una introduzione vi rimando ad un articolo sempre su questo sito che ne illustra i principi base), uno standard de facto nel dimensionamento funzionale del software che, ormai da alcuni anni, è stato anche riconosciuto dall'ISO (con rif. a ISO/IEC 20926). In Italia il suo utilizzo è ampio e consolidato soprattutto nel settore della Pubblica Amministrazione ma anche in diversi ambiti industriali (telecomunicazioni, automotive, ...).

Un aspetto saliente dei Function Point è la loro assoluta indipendenza dalla tecnologia: il processo di dimensionamento in FP (di un progetto, una applicazione, una manutenzione evolutiva) prescinde completamente dalla specifica tecnologia ed architettura realizzativa per concentrarsi invece sulle caratteristiche funzionali implementate, in termini di funzionalità realizzate (o da realizzare se si tratta di un progetto), e di dati trattati (letti e/o aggiornati).

I Function Point sono governati, a livello mondiale, dall'IFPUG – International Function Point User Group: un organismo internazionale con sede negli Stati Uniti che definisce le regole di conteggio dei Function Point, gestisce l'evoluzione della metrica e le linee guida per l'applicazione pratica.

Attualmente è in vigore la versione 4.2 del Manuale di Conteggio, ma è di prossima uscita (prima metà del 2009 ?) la versione 4.3.

UML è lo Unified Modeling Language: il linguaggio di modellazione che costituisce lo standard di diritto e di fatto per la modellazione di progetti realizzati con tecnologie object oriented.

UML è uno standard gestito dall'OMG – Object Management Group ed è il risultato della convergenza su un unico piano di tre “guru” dell'informatica mondiale: Grady Booch, Ivar Jacobson, Jim Rumbaugh.

UML ha una storia ormai più che decennale ed è centrale oggi per la realizzazione delle applicazioni software.

Come ha detto Andrew Watson, vice-direttore e Direttore Tecnico dell'Object Management Group, *“la storia della modellazione visuale nell'industria del software si divide in due ere: ‘Prima dell'UML’ e ‘Dopo l'UML’”*

UML viene utilizzato non solo per la progettazione del software ma anche in ambiti extra realizzazione del software, ad esempio per analisi di processi di business e progettazione di sistemi.

Sicuramente uno dei fattori che ha favorito il successo planetario di UML è, anche qui, il suo mantenersi avulso da qualsiasi specifica tecnologia e/o piattaforma software o hardware.

Tuttavia, mentre per i Function Point questa indipendenza è praticamente scontata e non potrebbe essere altrimenti, per UML la possibile “scivolata” verso uno specifico “lido tecnologico” è stato sicuramente un pericolo molto concreto che, tuttavia, fino ad oggi non è mai avvenuto.

Attualmente siamo arrivati alla versione 2.1 di UML, anche se, almeno in Italia, per quello che mi consta, si fa ancora sostanzialmente largo utilizzo di UML 1.x.

L'ambito più comune in cui Function Point e UML si incrociano è quello dei nuovi progetti software (nuove applicazioni, manutenzioni evolutive di un certo impegno su applicazioni esistenti).

Questo articolo si rivolge a a coloro che effettuano valutazioni in Function Point e che hanno una qualche conoscenza circa UML, senza la pretesa di esaurire l'argomento dell'incrocio FP-UML ma solo con l'obiettivo di fornire degli spunti per sfruttare al meglio nell'ambito della Function Point analisi le informazioni contenute in diagrammi UML realizzati nell'ambito di progetto.

Occorre premettere che, pur non essendo questo un argomento nuovo (in rete si trovano diversi articoli a riguardo), non vi è nulla di sancito o di ufficiale a riguardo (almeno a mia conoscenza) e quindi quanto segue è frutto di informazioni tratte da materiali reperiti in rete e nozioni acquisite nell'esperienza quotidiana.

Elementi di UML utili per la valutazione dei FP

Una premessa: diversi diagrammi UML sono utilizzati in diversi momenti del ciclo di realizzazione del software: ad esempio il diagramma delle classi, forse il più importante diagramma dell'UML, viene utilizzato in Analisi, in primis per rappresentare il modello concettuale dei dati, ma viene anche utilizzato in Progettazione per definire classi che saranno direttamente tradotte in classi software, espresse nel linguaggio target (ad es. classi Java).

Ciò vale anche per gli altri diagrammi: potenzialmente tutti i diagrammi possono essere "visti" a diversi livelli di dettaglio, e in modo avulso o, invece, dipendente dalla tecnologia.

Noi, qui, ci concentriamo sui diagrammi visti a livello di Analisi, e quindi avulsi da qualunque specifica piattaforma tecnologica. Il motivo di ciò è che i diagrammi più "tecnici", frutto delle attività più avanzate del ciclo realizzativo (tipicamente la progettazione) sono normalmente derivati proprio a partire dai diagrammi di analisi. Questi vengono "calati" su una specifica piattaforma tecnologica e architeturale, tenendo conto di aspetti "non funzionali". Il risultato sono diagrammi a livello di Disegno, talvolta molto complessi da "leggere" (soprattutto per i non addetti ai lavori) e contengono molte informazioni che non interessano chi si appresta a effettuare una valutazione in FP.

Al contrario i diagrammi UML realizzati a livello di Analisi sono quelli effettivamente interessanti per i nostri scopi, in quanto si concentrano sulla rappresentazione e implementazione di caratteristiche funzionali della applicazione.

Altra cosa importante : non tutti i diagrammi UML sono interessanti ai fini del dimensionamento in FP di una applicazione.

Noi qui faremo direttamente riferimento ai tre elementi di UML che riteniamo più utili per i nostri scopi: casi d'uso e diagramma dei casi d'uso, diagramma delle classi, diagramma di sequenza/collaborazione (coerentemente con quanto esposto da Pace, Calavaro e Cantone in [PCC04]),

Tutti e tre questi diagrammi sono presenti fin dalla nascita di UML e, nelle sue varie versioni, non hanno subito sostanziali mutamenti. Daremo delle indicazioni su quali informazioni, in ottica Function Point analisi, si può ragionevolmente pensare di ricavare da ognuno di questi.

Casi d'uso e diagramma dei casi d'uso

"un caso d'uso è una sequenza di transazioni in un sistema il cui compito è di conseguire un risultato di valore misurabile per un singolo attore del sistema"

Ivar Jacobson

"Un caso d'uso è una collezione di possibili sequenze di interazioni tra il sistema in esame e i suoi attori esterni, relazionate a conseguire un determinato obiettivo"

Alistair Cockburn

Il "caso d'uso" rappresenta una funzionalità del sistema dal punto di vista di chi la utilizza.

Il concetto formale di “caso d’uso” fu introdotto da Ivar Jacobson nel suo testo [J92] in cui presenta il suo approccio all’Analisi e Disegno Object Oriented.

I casi d’uso sono, sostanzialmente, una tecnica per definire i requisiti funzionali: tramite i casi d’uso si esprimono le specifiche funzionali di una applicazione. Aiutano a mantenere gli analisti focalizzati sulla descrizione di “cosa deve fare” l’applicazione senza cadere anche nella descrizione (spesso prematura, sicuramente da trattare in documentazione a parte) di “come lo deve fare”: una tendenza cui gli informatici vanno facilmente incontro, col risultato di documenti di analisi che sono in realtà miscugli di analisi, progettazione, suggerimenti/vincoli per la realizzazione.

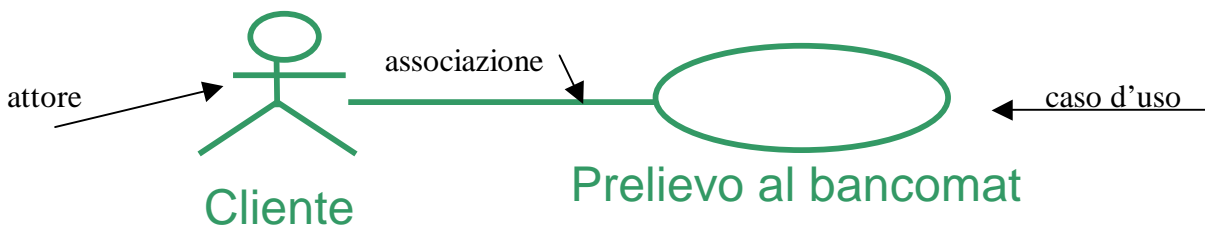
Al concetto di caso d’uso si accompagna il concetto di “**Attore**”:

gli Attori

- sono coloro che interagiscono con il sistema, ma sono **esterni** ad esso
- possono essere:
 - persone (dipendenti, clienti)
 - organizzazioni, enti, istituzioni
 - altre applicazioni o sistemi (hardware e software)

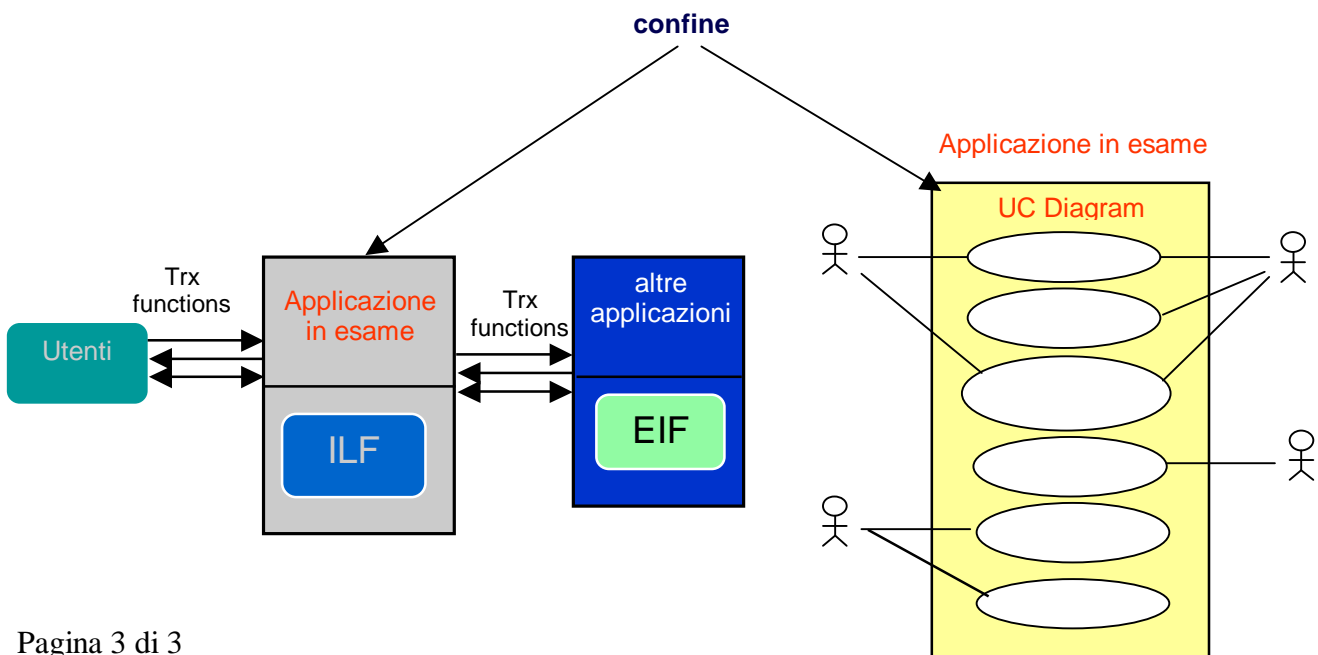
normalmente per ogni caso d’uso c’è un attore “iniziatore” che genera l’evento che innesca il caso d’uso (attore primario) ed eventualmente altri attori che vi partecipano (attori secondari), e che forniscono “assistenza” al sistema per consentire di raggiungere il risultato

Da un punto di vista grafico gli elementi base del diagramma UML dei casi d’uso sono:



L’associazione tra attori e casi d’uso ha il significato di “comunicazione”. Può essere orientata per evidenziare la direzione delle comunicazioni.

Il diagramma dei casi d’uso a livello di sistema è utilissimo per individuare il confine della applicazione: i concetti di Attore in UML e Utente nella Function Point Analysis sono quantomeno molto simili mentre i casi d’uso rappresentano le funzionalità interne del sistema ([FAN97], [PCC04]).



Per un approfondimento su questi temi, e anche su quelli esposti nel seguito, consiglio un validissimo articolo di Fetcke, Abran, Nguyen (Università del Quebec, Canada) reperibile in rete [FAN97]

Invece, il diagramma dei casi d'uso è di scarsa (se non nulla) utilità per la definizione degli elementi funzione di tipo transazionale (EI, EO, EQ).

La descrizione dei casi d'uso è invece fondamentale per identificare EI, EO, EQ: se effettuata ad un livello di dettaglio approfondito può consentire anche un conteggio, altrimenti quantomeno una buona stima.

In termini assolutamente generali si può dire che

ad un caso d'uso tipicamente corrispondono da 1 a N processi elementari

Ma quanto può "essere grande" un caso d'uso? David Longstreet in un suo articolo [LONG03] fornisce una indicazione di massima sul limite superiore di dimensione in FP di un caso d'uso: 50 FP.

Quindi nella analisi Function Point occorre esaminare la descrizione fornita del caso d'uso e, applicando le regole definite nel CPM, identificare e classificare i processi elementari in essa contenuti. Una descrizione sufficientemente approfondita di un caso d'uso deve consentire non solo di individuare i processi elementari ma, assieme all'esame del modello concettuale dei dati (trattato in seguito), di definirne anche la complessità.

Qualora questo si riveli impossibile è lecito porsi dei dubbi circa la completezza delle specifiche formali definite per il sistema in realizzazione: è probabile che i problemi che incontra lo specialista dei Function Point nel svolgere la sua analisi saranno incontrati anche dai progettisti e sviluppatori che prenderanno in carico il caso d'uso, in quanto è probabile che anch'essi non troveranno nella specifica delle informazioni essenziali per il loro lavoro.

Più in particolare, per la mia esperienza, spesso la definizione dei casi d'uso è carente per quanto riguarda il consentire l'individuazione delle entità coinvolte in un certo passo o algoritmo o interfaccia: ciò rende difficile, se non impossibile, definire l'esatta complessità di un processo elementare individuato, limitando di fatto la precisione della valutazione ad una stima invece che al conteggio.

Analogo discorso vale per la definizione delle interfacce utente: se queste non sono specificate ad un livello di dettaglio adeguato non è possibile definire esattamente la complessità del processo elementare cui l'interfaccia è associata.

Diagramma delle classi (Class diagram)

Per determinare le funzioni di tipo dati (ILF, EIF) lo strumento ideale è il **modello concettuale** dei dati completo (quindi con entità, attributi, relazioni, cardinalità tutti completamente definiti).

Il modello concettuale dei dati viene tradizionalmente rappresentato mediante un diagramma entità/relazioni, che per evitare le ridondanze di informazioni, viene solitamente realizzato in forma normale. La terza forma normale è, di solito, quella più utilizzata.

In UML per avere qualcosa di analogo bisogna guardare al diagramma delle classi entità dell'applicazione.

Si ricorda che in UML, e più in particolare nell'approccio dettato da Jacobson in [J97] le classi sono specializzate fin dalla analisi (secondo un approccio tipo MVC – Model-View-Controller) in:

classi entità (ENTITY)

stereotipo UML



raccogliono le informazioni tipiche del dominio del problema (dati applicativi) e le operazioni naturalmente accoppiate a queste informazioni.

classi interfaccia (BOUNDARY)



si occupano di gestire il colloquio tra gli attori e il sistema. Possono essere interfacce ad altri sistemi o interfacce ad utenti umani, quali mappe, tabulati, ecc...

classi di controllo o di coordinamento (CONTROL)



contengono la logica necessaria per coordinare un servizio fornito dal sistema; gestiscono l'interazione transazionale tra oggetti interfaccia e oggetti entità; rappresentano il "collante" che fa comunicare questi oggetti. Normalmente modellano comportamenti che richiedono la collaborazione di più oggetti.

Le prime, le classi entità, sono di sicuro interesse ai fini del calcolo delle funzioni di tipo dati. Occorre specificare che anche il diagramma delle classi entità realizzato con UML ha tutti gli elementi espressivi per poter rappresentare un modello concettuale dei dati : si può dire che le classi entità, considerando solo le classi come aventi attributi e non operazioni, sono equivalenti alle entità del modello entità-relazioni (da qui il loro nome di "classi entità").

In termini assolutamente generali, si può dire che ogni (classe) entità indipendente corrisponde a un file logico ed ogni attributo della classe è un DET, mentre ogni entità dipendente corrisponde a un RET del file logico cui appartiene.

Una (classe) entità A è indipendente quando è significativa di per se stessa per il business, senza richiedere la presenza di altre entità. Una entità A è invece dipendente quando non è significativa di per se stessa per il business, ma richiede necessariamente la presenza di un'altra entità B da cui A dipende

Per una interpretazione del modello concettuale dei dati in ottica di dimensionamento in FP si rimanda a [CPM4.2] in particolare alla "Parte 2 – Prassi di conteggio" che illustra in modo, a mio parere, molto chiaro e ben fatto le regole da applicare.

Qui, in estrema sintesi, riportiamo la tabellina di corrispondenza che prende in considerazione le relazioni tra entità dipendenti e indipendenti, con l'avvertenza che la prima regola sempre da rispettare per il valutatore FP è **essere coerenti con la visione utente dei dati.**

Tipo di relazione tra le due entità, A e B	Se vale la condizione:	Allora conta come LF:
(1) : (N)	(A e B sono indipendenti)	2 LF
1 : N	B è un'entità dipendente da A	1 LF
	B è un'entità indipendente da A	2 LF
1 : (N)	B è un'entità dipendente da A	1 LF
	B è un'entità indipendente da A	2 LF
(1) : N	A è un'entità dipendente da B	1 LF
	A è un'entità indipendente da B	2 LF
(1) : (1)	(A e B sono indipendenti)	2 LF
1 : 1	(A e B sono dipendenti)	1 LF
1 : (1)	B è un'entità dipendente da A	1 LF
	B è un'entità indipendente da A	2 LF
(N) : (M)	(A e B sono indipendenti)	2 LF
N : M	B è un'entità dipendente da A	1 LF
	B è un'entità indipendente da A	2 LF
N : (M)	B è un'entità dipendente da A	1 LF
	B è un'entità indipendente da A	2 LF

A quanto reperibile dal Manuale di Conteggio [CPM4.2] credo sia solo aggiungere una considerazione che riguarda **la relazione di aggregazione** in quanto specifica di UML.

Nel diagramma delle classi entità si fa spesso uso della Aggregazione, ed in particolare della aggregazione per composizione. Si tratta di un tipo speciale di associazione tra classi in cui si rende esplicito il fatto che tra le due classi associate esiste una relazione del tipo “IS-PART-OF”

Vediamo un esempio:

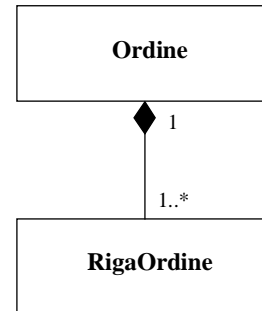
RigaOrdine è parte di Ordine.

Tra le due entità c'è un legame molto forte, in pratica

RigaOrdine non può esistere senza l'Ordine cui appartiene.

Quindi, se cancello Ordine devo cancellare anche tutte le

Righe a lui associate.



In ottica Function Point, è chiaro che sia Ordine che RigaOrdine, cioè entrambe le classi unite dalla aggregazione, appartengono allo stesso gruppo logico di dati, ILF o EIF, di cui saranno RET distinti.

Nota bene: analogamente a quanto succede per le relazioni di tipo IS-A in cui si possono costruire gerarchie di specializzazioni, anche la relazione IS-PART-OF può costruire delle “gerarchie di composizioni”: in questo caso si può dire che una intera gerarchia deve essere considerata, dal punto di vista dei FP, come appartenente ad un unico gruppo logico di dati, con differenti RET uno per ogni elemento della composizione.

Diagrammi di sequenza (o collaborazione)

I diagrammi di sequenza e collaborazione vengono, in UML, raggruppati sotto un unico cappello dei diagrammi di interazione. Sono diagrammi isomorfi, nel senso che, seppure in forma diversa, hanno lo stesso contenuto informativo: ad esempio in Rational Rose è possibile, disegnando uno dei due, ottenere automaticamente l'altro.

Noi qui ci riferiremo al diagramma di sequenza.

Contrariamente al diagramma delle classi, che rappresenta delle caratteristiche “statiche” del sistema in progettazione, il diagramma di sequenza rappresenta caratteristiche “dinamiche” cioè di comportamento “attivo” della applicazione.

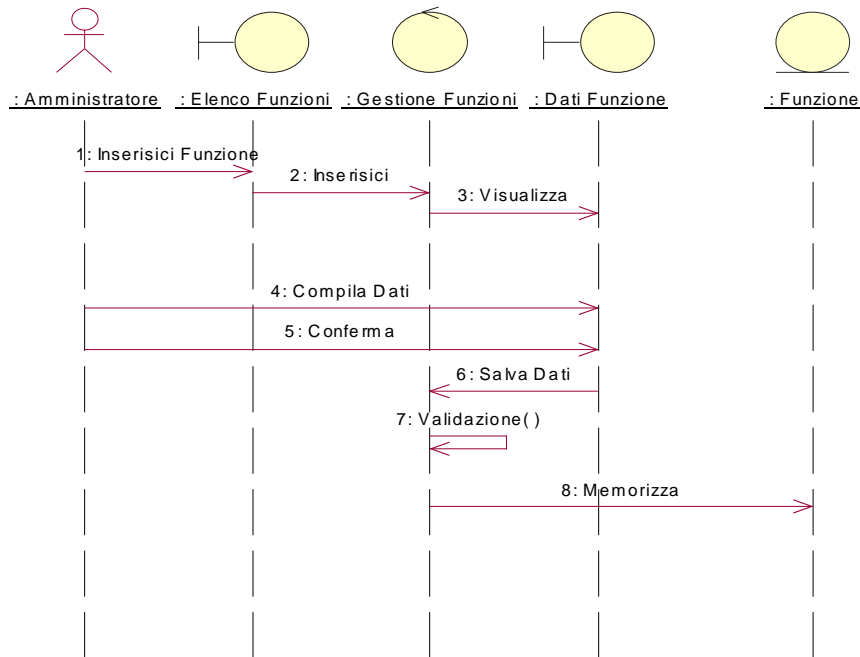
Il diagramma di sequenza viene infatti utilizzato per progettare le interazioni tra le classi che dovranno realizzare i comportamenti definiti nei casi d'uso.

In termini generali si può dire che occorre realizzare da 1 a N diagrammi di sequenza per ogni caso d'uso, in funzione degli scenari che il caso d'uso presenta.

Il diagramma di sequenza, ed anche il diagramma delle classi, possono essere utilizzati sia in analisi che in progettazione. Noi qui ci soffermiamo sui diagrammi a livello di analisi in quanto sono quelli che ci interessano: focalizzati sugli aspetti funzionali, mentre i diagrammi di sequenza a livello di progettazione devono necessariamente introdurre elementi legati alla architettura e piattaforma tecnologica di riferimento.

In analisi la realizzazione dei diagrammi di sequenza segue ancora l'indicazione della specializzazione delle classi in classi entità, classi interfaccia e classi di controllo già precedentemente introdotta.

Vediamo un semplice esempio di diagramma di sequenza a livello di analisi



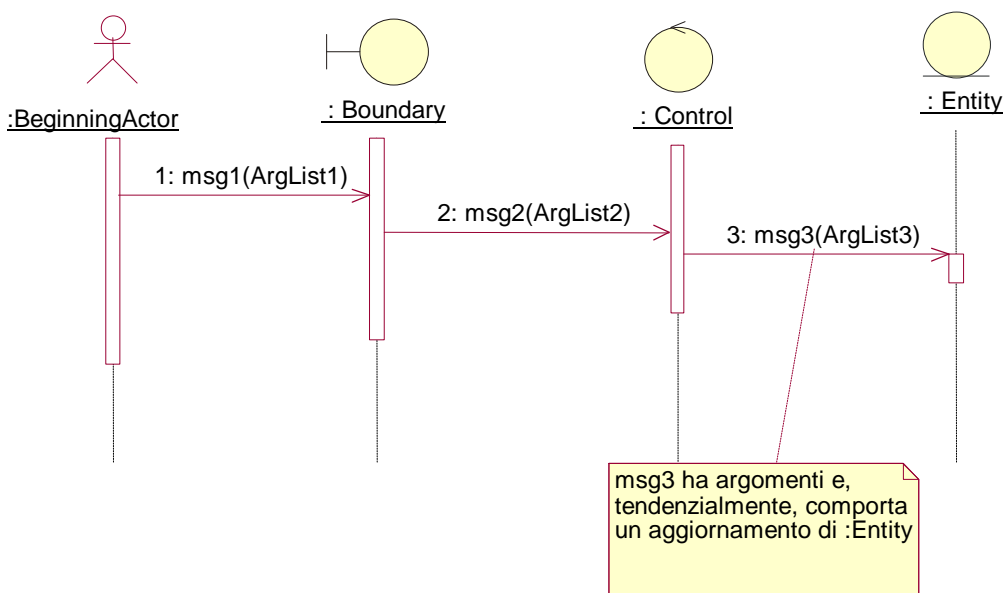
E' possibile dai diagrammi di sequenza ricavare delle utili indicazioni per individuare i processi elementari (EI, EO, EQ) e anche per ottenere informazioni di dettaglio circa le entità accedute da un determinato processo, utili per definirne l'esatta complessità.

Per quanto riguarda l'individuazione dei processi elementari possono essere utili due "pattern", che si rifanno ad articoli già presenti in rete ([UKI01], [PCC04]).

Nota bene: l'individuare in un diagramma di sequenza uno dei pattern sotto riportati non è garanzia di aver individuato un processo elementare, la eventuale conferma dovrà essere ricercata applicando le consuete regole definite in [CPM4.2].

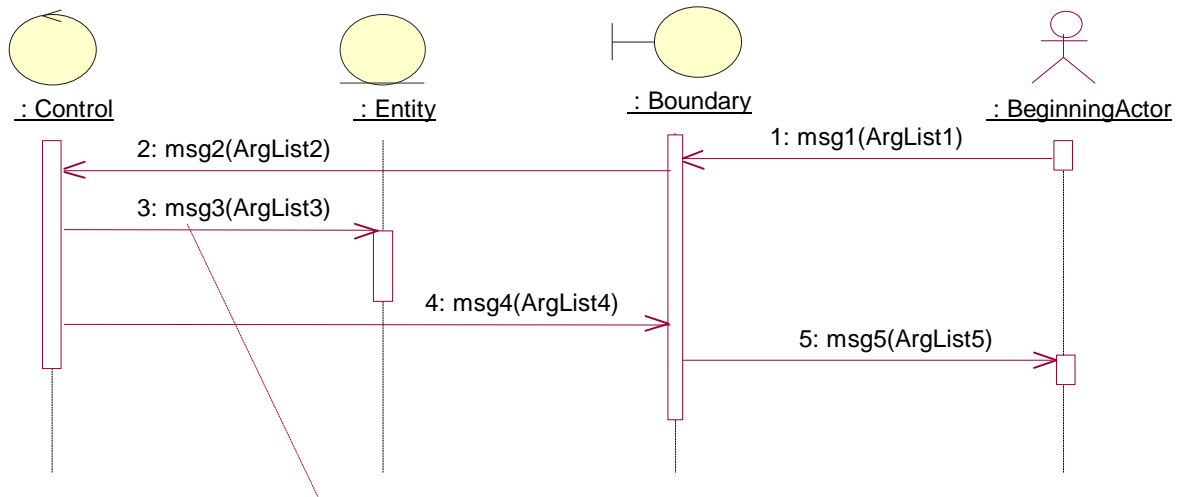
External input pattern

Obiettivo del Beginning Actor è, tramite msg1, inserire/modificare dati nel sistema. Con msg3 la transazione di inserimento/aggiornamento si chiude).



External Output/External Query pattern

Obiettivo del Beginning Actor è recuperare dei dati dal sistema, ArgList1 rappresenta l'insieme dei parametri di ricerca mentre ArgList5 sono i valori recuperati, coerenti con i parametri impostati



Se msg3 legge da Entity dei valori che sono tutti già presenti sul DB, senza effettuare calcoli e senza aggiornare Entity si ha un EQ, altrimenti se ci sono valori calcolati e/o aggiornamenti si ha un EO

Conclusione

Ovviamente molto altro ci sarebbe ancora da dire, e anche gli stessi argomenti che sono stati qui trattati lo potrebbero essere ad un livello di approfondimento ben maggiore.

Tuttavia, come già detto inizialmente, l'obiettivo di questo articolo è stimolare la curiosità e la ricerca degli interessati al dimensionamento in Function Point: i punti di contatto con UML sono molti e il monte informativo espresso nei diagrammi spesso non viene adeguatamente sfruttato da chi effettua la valutazione funzionale, preferendo il solo colloquio informale con analisti funzionali.

Sebbene sia chiaro che questo colloquio tra specialista di metriche e analista funzionale è fondamentale, tuttavia alcuni elementi importanti che possono aiutare a dirimere dubbi o a stabilire con maggiore precisione aspetti legati al conteggio possono essere ricavati dai diagrammi UML.

Ciò senza necessariamente entrare in aspetti tecnici o troppo al di fuori della competenza dello specialista metrico.

Riferimenti

[J92] Jacobson, I. et al. “*Object-Oriented Software Engineering: A Use-Case Driven Approach*”, Addison-Wesley, 1992

[PCC04] Pace, Calavaro, Cantone “*Function Point and UML: State of the Art and Evaluation Model*” SMEF-Software Measurement Forum 2004

[FAN97] Fetcke, Abran, Nguyen “*Mapping the OO-Jacobson Approach into Function Point Analysis*” 1997

[CPM4.2] IFPUG – The International Function Point User Group “*Function Point: Manuale delle Regole di Conteggio - Versione 4.2*” 2004

[UKI01] Uemura, Kusumoto, Inoue “*Function-Point analysis using design specification based on the Unified Modelling Language*” 2001

[LONG03] David Longstreet “*Use Cases and Function Points*” Copyright Longstreet Consulting Inc. 1995 -2003 <http://www.softwaremetrics.com/> David@SoftwareMetrics.Com
Longstreet Consulting Inc.
2207 S. West Walnut St.
Blue Springs, MO 64015
(816) 739-4058

This document was created with Win2PDF available at <http://www.win2pdf.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.