Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Metriche di adattabilità per applicazioni basate su servizi.

Relatore: Prof. Raffaela Mirandola
Correlatore: Ing. Diego Perez-Palacin

Tesi di laurea di:
Micaela Zanotti Matr. 786972

Anno Accademico 2015–2016

Alla mia famiglia che mi ha sempre sostenuto in questo percorso,
A mia sorella perchè possa affrontare con determinazione ogni sfida che la vita le presenterà e sappia uscirne vincente,
Ai miei amici e amiche che hanno trovato la parola giusta nel momento giusto,
Al running che mi ha insegnato a non arrendermi mai e a cercare un continuo miglioramento,
A me stessa per aver realizzato il mio sogno.

Ringraziamenti

Vorrei ringraziare innanzitutto la Professoressa Mirandola Raffaela e l'Ingegnere Diego Perez - Palacin per aver creduto in me e avermi proposto questa tesi.

Mi hanno permesso di affrontare un campo di studio nuovo e mi hanno guidato nella formulazione di metriche innovative di analisi, per quelli che a mio avviso sono i sistemi di applicazioni del futuro. Li ringrazio per ogni spunto di riflessione che mi hanno fornito per poter scrivere questa tesi e per avermi messo a mio agio sempre. Mi auguro di poter lavorare ancora con loro.

Ringrazio il Politecnico perchè in questi anni di studio mi ha insegnato a dare sempre il massimo delle mie possibilità.

Infine, ma non meno importanti, ringrazio la mia famiglia e le mie amiche per aver affrontato con me gioie e dolori di questi anni, per i momenti in cui mi hanno fatto svagare e per i momenti in cui hanno rispettato i miei «no oggi devo studiare».

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

I sistemi auto - adattativi sono sistemi in grado di modificare il loro comportamento a tempo di esecuzione per rispondere ai cambiamenti che avvengono nell'ambiente di esecuzione del sistema, per rispondere ai requisiti dell'utente (es. nuove funzionalità introdotte) o per rispondere ad una propria esigenza di modifica (es. fallimento di una componente interna). Essi permettono alle applicazioni moderne di sopravvivere in un mondo che muta in maniera impredicibile. Tale campo di studio è però relativamente recente e quindi non esistono ancora metriche riconosciute universalmente per poterli analizzare. Lo scopo della tesi è mostrare un tentativo di formulazione di metriche adatte a studiare l'adattabilità di tali sistemi.

Quando si parla di sistemi auto - adattativi sorgono delle domande relative ad essi, quantomeno per capire come affrontarne lo studio. L'articolo «A survey on engineering approaches for self-adaptive systems» [21] analizza i sistemi auto - adattativi e si propone di rispondere ad alcune di queste domande per poter poi giungere ad una tassonomia di tali sistemi.

La prima domanda che ci si pone ed anche la più logica è « Chi deve attuare il cambiamento nel sistema? » Le domande la cui risposta non è così immediata, sono: quando è necessario adattare e perchè farlo; dove è necessario fare il cambiamento e che tipo di cambiamento fare e come effettuare questo cambiamento. E' possibile [21] definire delle dimensioni con cui affrontare la tematica. Per questa tesi è risultato importante concentrarsi sulla dimensione: *Controllo dell'adattamento.*

Per capire come controllare l'adattamento sono necessarie delle metriche di adattabilità. Lo scopo della tesi è sviluppare queste metriche, per questo motivo si è preso come punto di partenza l'articolo «On the relationships between QoS and software adaptability at the architectural level» [23].

In esso si affronta una prima analisi dei sistemi auto - adattativi studiando come varia l'adattabilità dell'intero sistema rispetto a due requisiti ritenuti importanti: availability e costo.

In [23] vengono proposte delle metriche di adattabilità fortemente ispirate al lavoro di Subramanian e Chung (2001) [22]. Tali metriche analizzano l'aspetto puramente statico del sistema senza considerare la sua evoluzione a tempo di esecuzione. La tesi parte da questo insieme di metriche di analisi di adattabilità e lo espande con-

siderando anche il comportamento delle applicazioni a tempo di esecuzione ponendo l'attenzione sui servizi forniti dai componenti software dell'applicazione stessa. Tali servizi vengono richiesti da altri componenti all'interno dell'applicazione e servono a fornire il servizio finale. Il motivo per cui si è concentrata l'attenzione su tali servizi è quello di fornire un'informazione aggiuntiva sui sistemi auto - adattativi ovvero capire quale componente all'interno dell'applicazione è quello che entra maggiormente in esecuzione (perchè richiamato un numero maggiore di volte) e quello che ha la maggiore probabilità di rottura. Tale studio è motivato dall'esigenza di definire metriche più accurate che permettano al sistema di adattarsi meglio e in maniera più accurata.

Per perseguire lo scopo dello studio è necessario analizzare uno spazio di architetture composte da componenti e per ognuna di esse calcolare le metriche di adattabilità al fine di individuare quale o quali di esse è la soluzione migliore verso cui migrare partendo dall'architettura attualmente in uso per l'applicazione.

Essendo un lavoro laborioso e che richiede uno sforzo computazionale non indifferente si è pensato di sviluppare un framework apposito che analizzi lo spazio delle architetture possibili e fornisca a video i risultati delle metriche in modo da permettere al progettista di sistema di scegliere l'architettura che sia maggiormente adattabile rispetto a quella in uso rispetto ai requisiti indicati dal proprietario del sistema.

La tesi è strutturata nel modo seguente:

nel capitolo uno viene descritto nel dettaglio cosa è un sistema auto - adattativo e si riporta l'analisi proposta in [21] circa le dimensioni con cui analizzarlo. Inoltre si presentano metriche definite in [23], il framework implementato e le limitazioni del lavoro esistente.

Nel capitolo due si presentano le motivazioni alla base di questa tesi, le metriche definite in questo lavoro riportando per miglior comprensione un esempio pratico per ogni metrica descritta.

Nel capitolo tre si affronta un caso di studio reale e più complesso su cui applicare le metriche proposte per illustrare in modo esplicito l'applicabilità delle metriche proposte.

Nel capitolo quattro si descrive il framework sviluppato (scaricabile al link [6]) illustrando dapprima le tecnologie utilizzate e poi la sua utilizzazione attraverso un insieme di screenshots.

Nelle conclusioni si riepilogano i limiti di questo lavoro fornendo spunti per gli sviluppi futuri.

Capitolo 1

Stato dell'arte

In questo capitolo si introducono le nozioni relative al campo di interesse dello studio ovvero i Sistemi auto - adattativi.

Subito dopo ci si concentrerà sulla fase iniziale dello studio ovvero la definizione formale delle metriche utili ai sistemi auto - adattativi per poter controllare la propria adattabilità ai cambiamenti.

In questa fase dello studio si svolge una accurata analisi statica del sistema, rimandando al capitolo successivo l'analisi dinamica del comportamento del sistema e relative metriche di adattabilità.

1.1 I Sistemi auto - adattativi

Oggi giorno la complessità dei moderni sistemi di informazioni «pervasivi» è in continuo aumento e, gli utenti ormai si aspettano di potervi accedere in qualunque posto, in qualunque momento e in qualsiasi condizione grazie alla notevole diffusione dei dispositivi mobili e dell'ormai onnipresente rete wireless ad alta velocità. Attualmente le case produttrici presenti sul mercato sono molteplici e producono una moltitudine di apparecchi. Per questa ragione, i sistemi devono avere come primo obiettivo quello di creare una piattaforma che permetta l'accesso a tutta la eterogeneità di dispositivi che spesso differiscono tra loro proprio nella configurazione dei parametri di accesso a sistemi o portali.

Tali sistemi inoltre sono altamente distribuiti e devono saper operare in un ambiente mutevole come sono di fatto i sistemi di grandi dimensioni quale è «Internet of things».

La complessità di questi sistemi rende la loro realizzazione, configurazione e manutenzione davvero ardua.

Per questo motivo e per l'importanza che rivestono sempre più tali sistemi, si sono studiati i metodi con cui diminuire lo sforzo di sviluppo. La soluzione giunge con i *Self Adaptive Systems (SAS)* [21].

Un *SAS* è un sistema in grado di rispondere ai cambiamenti dell'ambiente in cui opera modificandosi automaticamente e autonomamente agendo sui propri parametri di sistema. Tali sistemi stanno ottenendo sempre più considerazione in vari ambiti di ricerca.

Forniscono tutte le proprietà di auto-gestione quali: auto configurazione, auto-risanamento in caso di rottura, auto-ottimizzazione e auto-protezione. Esse vengono racchiuse nelle due proprietà base : self awareness e context awareness.

La proprietà Self awareness è intesa come la capacità del sistema di avere consapevolezza di se stesso e di saper monitorare le proprie risorse e comportamento.

La proprietà Context awareness è intesa invece come la capacità del sistema di conoscere l'ambiente in cui opera laddove il context è rappresentato da qualsiasi informazione utilizzata da un componente (sia esso persona, oggetto o altro).

Quando si trattano questi sistemi è bene identificare una tassonomia che incorpori i risultati dei vari studi circa le relative prospettive lungo cui studiare l'adattabilità del sistema.

Per giungere ad una tassonomia ci si pone delle domande: le *5W + 1H* che sono:

- Quando si adatta?
- Perchè farlo?
- Dove attuare il cambiamento?
- Che tipo di cambiamento è richiesto?
- Chi deve fare l'adattamento?
- Come fare l'adattamento?

in un contesto di auto - adattabilità, la domanda « Chi deve fare il cambiamento? » trova la risposta banale in: è il sistema stesso che si adatta; per tutte le altre domande sono state studiate negli anni diverse dimensioni: *Tempo, Ragione, Livello, Tecnica, Controllo dell'adattamento.*

La Figura [1.1] mostra la tassonomia relativa alle dimensioni identificate per i sistemi self adaptive.

1.1.1 Le dimensioni

La dimensione del *Tempo* è quella che permette di rispondere alla domanda « Quando si adatta? ». La prospettiva tradizionale è quella reattiva ovvero è necessario un adattamento solo a seguito di un evento scatenante. Sono state trovate altre due prospettive interessanti quali: predittiva e proattiva. Rispettivamente la prima studia il sistema e la sua necessità di adattabilità prima che un evento diminuisca drasticamente le sue performance mentre la seconda rappresenta un adattamento

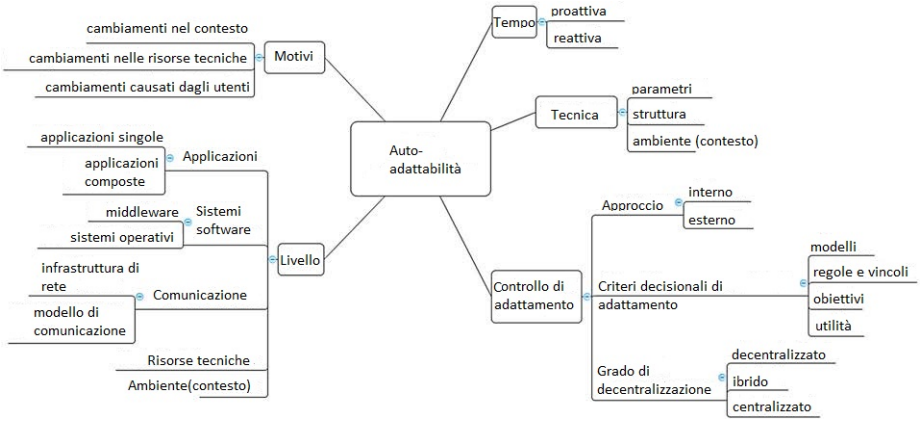

Figura 1.1: Tassonomia dei sistemi self adaptive [21].

che migliora le prestazioni del sistema a prescindere da eventi scatenanti che possano ridurne la capacità di funzionamento a regime. Dal punto di vista dell'utente la prospettiva proattiva è la miglior scelta in quanto non crea interruzioni nella fruizione del servizio. Purtroppo però questa prospettiva è difficile da sviluppare e fortemente dipendente dalle funzioni predittive utili per capire dove applicare il miglioramento. Altro aspetto interessante di questa dimensione è il monitoraggio. Esso può essere continuo o adattativo. Un monitoraggio continuo richiede un notevole dispendio di energie da profondere nel controllare costantemente le risorse e l'ambiente di lavoro mentre un monitoraggio adattativo si interessa ad aspetti scelti e selezionati che vengono analizzati periodicamente e solo in caso di anomalie si intensifica il lavoro.

La seconda dimensione considerata è la *Ragione*. Sappiamo che in generale l'adattamento è una reazione al cambiamento e, molto spesso è costoso. Sorge quindi spontaneo chiedersi « Perchè farlo? » e se veramente è necessario.

Le ragioni vanno ricercate nei cambiamenti di vari elementi del sistema:

- un cambiamento nelle risorse tecniche come un difetto hardware o un problema a livello software
- un cambiamento dell'ambiente
- un cambiamento relativo agli utenti come la costituzione di nuovi gruppi o variazioni delle preferenze.

La terza dimensione considerata è il *Livello*. In generale un SAS è costituito da due elementi: il managed element e la logica di adattamento. Quest'ultima tende a rimanere stabile mentre è il managed element che si adatta. Esso è composto da vari livelli. Al livello base si trovano le risorse tecniche che rappresentano l'hardware cioè computers, smartphones e altri apparecchi. Esse vengono controllate dal sistema operativo lato software o, nel caso di sistemi distribuiti, dal middleware. Al livello più alto si trovano infine le applicazioni. Per rispondere alla domanda « Dove attuare

il cambiamento? » quindi bisogna stabilire il giusto livello in cui operare. Non bisogna trascurare la gestione della comunicazione tra il *managed resource* (MR) e la *adaptation logic* (AL). Anch'essa può essere osservata in due modi differenti. La prima considera la costituzione fisica della rete con vari elementi da configurare come la scheda di rete o i routers e la seconda view è relativa al pattern di comunicazione ovvero la gestione della logica con cui avviene l'interazione tra elementi.

La quarta dimensione considerata è la Tecnica. Essa è fondamentale perchè non basta sapere a quale livello attuare il cambiamento ma bisogna anche rispondere alla domanda « Che tipo di cambiamento è richiesto? ». In questo caso o si agisce sui parametri del sistema o si considera il sistema come un aggregato di componenti. Questo modo di vedere il sistema prende il nome di visione composita, essa permette ai sistemi di collaborare favorendo l'interscambio di algoritmi e, cosa ancora più importante permette ai sistemi di scambiarsi i componenti necessari al loro funzionamento. Questo si traduce in un miglioramento delle performance in quanto supporta la sostituzione di componenti rotti o difettati.

Per quanto concerne la variazione dei parametri del sistema, essa permette sicuramente di avere pieno controllo del sistema in questione ma aumenta di molto la complessità ed impedisce l'integrazione dinamica con nuovi componenti ed algoritmi.

La quinta dimensione considerata è il *Controllo dell'adattamento.* Porremo maggiore attenzione, nel seguito, a questa dimensione in quanto prevede l'utilizzo di metriche.

1.1.1.1 La dimensione: Controllo dell'adattamento

Come si è detto un SAS [21] è costituito da due elementi: il *managed resource* (MR) e la *adaptation logic* (AL). La logica si interessa a « Come fare l'adattamento? » in altre parole controlla come avviene l'adattamento gestendo possibili anomalie e pianificando le fasi di adattamento.

Esistono due approcci di controllo: uno è interno e interconnette la logica di adattamento del sistema con le risorse ed uno invece è un approccio esterno in cui si divide la logica adattativa dalle risorse il che aumenta la facilità di manutenzione grazie alla modularità. L'approccio interno ha due principali problemi, uno relativo alla scalabilità e l'altro alla manutenzione.

Un ulteriore aspetto da considerare di questa dimensione è il grado di decentralizzazione. Infatti finchè si tratta di gestire un sistema con un ridotto numero di risorse allora è da preferire una logica adattativa centralizzata ma, per sistemi di grandi dimensioni e un elevato numero di componenti, è da preferire una soluzione decentralizzata separando le responsabilità ed incrementando le performance.

Un SAS può essere visto come una tupla SAS = (AL, MR) dove:

- MR= mr_1,... mr_n sono le risorse gestite ovvero l'insieme dei componenti hardware e software.
- AL= (al_1,... al_n, M, A, P, E) sono gli elementi della logica in cui:
  - (M) è il monitoraggio dell'ambiente
  - (A) è l'analisi del dato che cambia
  - (P) è la pianificazione dell'adattamento
  - (E) è il controllo dell'esecuzione dell'adattamento.

Le dimensioni finora analizzate possono essere mappate con le funzionalità MAPE come descritto in tabella [1.1].

Tabella 1.1: Relazione tra le attività MAPE e le dimensioni della tassonomia [21].

	tempo	***ragione***	***livello***	***tecnica***
monitoraggio	continuo	cosa monitorare	identificare il livello	-
analisi	l'algoritmo dipende dalla proattività o reattività della dimensione	dove analizzare	-	-
planning	-	cosa può influenzare il planning	adattare il planning ai vari livelli	planning delle tecniche
esecuzione	-	-	esecuzione dei cambiamenti	esecuzione delle tecniche

Sono presenti differenti approcci per l'analisi e l'organizzazione. Tali approcci hanno bisogno delle metriche ad hoc per guidare l'adattabilità del sistema. Le metriche sono basate su:

- modello: in questo caso si ha una rappresentazione sia della situazione attuale che di quella desiderata;
- regole e policies: che servono a determinare come il sistema dovrebbe reagire al cambiamento;
- goals: che vengono utilizzati per influenzare il sistema sul come dovrebbe comportarsi per raggiungere determinate performance;
- utility: che viene definita per valutare il sistema in funzione di un costo o un ricavo. L'obiettivo risulta in una minimizzazione o massimizzazione rispettivamente dell'utility.

Una volta definite le metriche di adattabilità è possibile utilizzarle per analizzare il sistema. Si può scegliere di studiare il comportamento del sistema a runtime ovvero come vengono forniti i servizi all'utente finale oppure considerare il sistema come un insieme di componenti (hardware o software) e, capire come collaborano tra loro per il funzionamento di tutta l'architettura.

1.2 Il framework SOLAR

Le applicazioni moderne operano in un ambiente dinamico che muta in maniera imprevedibile e, dove i requisiti di funzionamento cambiano di continuo. Il requisito principale di un software diventa quindi la capacità di adattare il proprio comportamento in maniera dinamica per continuare a mantenere la stessa qualità del servizio fornito. Senza questa capacità di adattamento l'applicazione avrà un degrado delle prestazioni a causa dei componenti difettosi presenti nel sistema mentre con l'adattamento dinamico, l'applicazione potrà cambiare alcuni dei componenti che utilizza per fornire il servizio finale.

Negli anni recenti il mondo accademico ha affrontato il problema dell'adattamento per rispondere a questa esigenza e, ha introdotto i sistemi auto-adattativi evidenziando come sempre più utenti richiedano che le applicazioni siano flessibili e in grado di adattarsi alle loro esigenze con prestazioni sempre più elevate.

Tuttavia, fornire la capacità di adattamento del software può influenzare altri attributi di qualità come le prestazioni, l'affidabilità o manutenibilità e, nel caso peggiore, può capire che mentre si sta migliorando l'adattabilità del sistema potrebbero diminuire altri attributi di qualità. Va trovato il miglior equilibrio tra i diversi requisiti di qualità che un sistema deve soddisfare e la sua adattabilità. Questo è l'obiettivo ambizioso che lo studio vuole perseguire.

Come prima cosa lo studio fornisce un approccio per valutare i compromessi tra la capacità di adattamento del sistema e altri attributi di qualità del sistema stesso, come la availability o il costo. L'approccio si basa sulla definizione di un insieme di parametri che consentono la valutazione della capacità di adattamento del sistema a livello di architettura.

Lo studio ha il proposito di fornire uno strumento efficace che mostri al progettista del software le possibili soluzioni al problema in modo da supportare il progettiva nella scelta dei componenti che possono soddisfare tutti i requisiti di qualità del sistema. Questi componenti costituiscono l'architettura software, che sarà valutata secondo le metriche di adattabilità proposte dallo studio. La valutazione consentirà l'analisi del compromesso di adattabilità tra diversi attributi di qualità del software.

Lo studio fornisce quindi una possibile soluzione per aiutare i progettisti nella scelta non "una soluzione per ogni situazione".

Tale strumento trova implementazione nel framework SOLAR (SOftware quaLities and Adaptability Relationships) [23].

Verranno ora esposte le metriche di adattabilità dello studio e poi come vengono utilizzate per l'adattamento del sistema.

1.2.1 Le metriche di SOLAR

Le metriche di adattabilità definite in questo studio sono fortemente ispirate al lavoro di In Subramanian e Chung (2001) [22]. Il set di metriche analizza se il sistema è adattabile o meno e quantifica anche il grado di adattabilità ovvero la potenziale capacità di adattamento di una architettura software.

Tutte le metriche sono definite a livello di architettura.

Per definire le metriche e per valutare la qualità degli attributi ci si basa su una visione componente-e-connettore (*C&C view*) dell'architettura software, dal momento che questo punto di vista è comunemente utilizzato per valutare gli attributi di qualità del sistema a runtime. In questo tipo di visione si hanno:

- i componenti: sono gli elementi base presenti in fase di esecuzione dell'architettura.
- i connettori: sono percorsi di interazione tra i componenti e hanno anche interfacce o ruoli.

La rappresentazione grafica prevede l'uso del diagramma UML.

In figura [1.2] viene mostrato un esempio di un insieme di componenti. Tale insieme di componenti verrà utilizzato per esemplificare le metriche di adattabilità proposte.

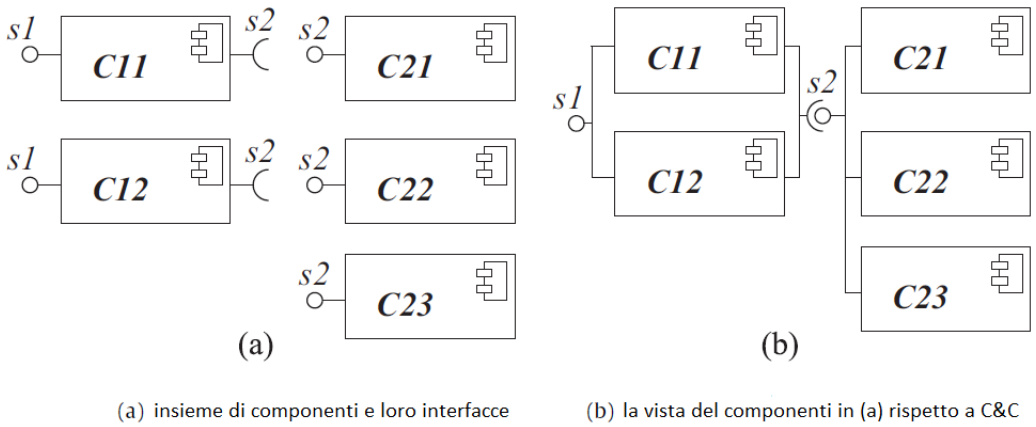

Figura 1.2: Esempio di un set di componenti considerati [23].

La rappresentazione grafica dell'insieme di componenti prevede l'uso di interfacce, disegnate tramite cerchi, con cui si espone il servizio fornito dal componente e, da agganci per indicare che un componente richiede un servizio.

Quando lo stesso servizio è richiesto da più componenti, si uniscono gli agganci come in Figura [1.2] (b).

In figura [1.3] viene mostrato l'insieme di componenti usati nell'architettura. Per distinguerli dall'insieme dei componenti possibili sono stati selezionati e inseriti in un riquadro grigio scuro.

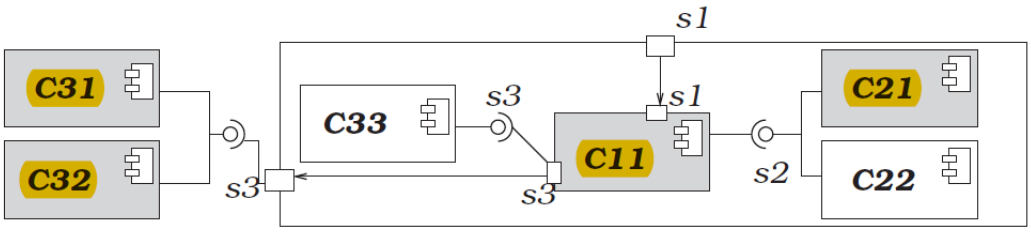

Figura 1.3: Componenti trovati. Componenti in uso sono quelli evidenziati e dentro riquadro grigio scuro. [23].

Le informazioni contenute in figura sono:

- il servizio s1 è fornito dal componente C11 che è in uso nell'architettura e anche l'unico presente tra i componenti utilizzabili.
- il componente C11 richiede il servizio s2 e il servizio s3.
- il servizio s2 viene fornito dai componenti C21 e C22. In uso si ha il componente C21.
- il servizio s3 viene fornito dai componenti C31, C32 e C33 ma si sceglie di avere in uso solo i componenti C31 e C32.

Basandosi sull'esempio fornito verranno ora presentate le metriche di adattabilità. Quattro di esse misurano le caratteristiche dei servizi forniti mentre una analizza l'architettura nella sua interezza. Si presenta prima in tabella [1.2] un riassunto dei termini utilizzati.

Tabella 1.2: Riassunto dei termini utilizzati nel capitolo 1 [23].

	significato
UC_i	numero dei componenti usati per il servizio i-esimo
C_i	numero totale dei componenti disponibili per il servizio i-esimo
n	numero dei servizi totali

1.2.1.1 Adattabilità dei servizi

- ***Assoluta adattabilità del servizio***. La prima metrica trattata misura il numero di componenti usati per fornire un dato servizio.

$$(AAS)\,\epsilon\mathbb{N}^n|AAS_i = \lfloor UC_i \rfloor$$

Questa metrica quantifica quanto un servizio sia adattabile semplicemente contando il numero di differenti alternative con cui esso può essere eseguito.

Per il caso di studio considerato si ha: AAS[1,1,2].

- ***Relativa adattabilità del servizio***. La seconda metrica trattata misura sempre il numero di componenti usati per fornire un servizio ma questa volta in relazione al numero di componenti totali disponibili che possano fruire il servizio designato.

$$(RAS)\,\epsilon\mathbb{Q}^n|RAS_i = \frac{\lfloor UC_i \rfloor}{\lfloor C_i \rfloor}$$

In questo modo si informa l'utente di quanto il servizio sia maggiormente adattabile.

Quando il rapporto tra componenti usati e componenti disponibili tende a uno implica che si sta usando quasi tutta la potenziale adattabilità raggiungibile.

Per il caso di studio considerato si ha: RAS[1,0.5,0.$\dot{6}$].

- ***Media di assoluta adattabilità del servizio***. Si può inoltre misurare la media dei componenti usati per servizio con la metrica

$$(MAAS)\,\epsilon\mathbb{Q}|MAAS = \frac{\sum_{i=1}^{n} AAS_i}{n}$$

Misura una media dello sforzo necessario per fornire ogni servizio. Architetture con servizi maggiormente adattabili avranno un alto valore di MAAS pertanto con MAAS >1 si afferma che il sistema include servizi adattabili e almeno un AAS ha valore maggiore di 1 mentre per MAAS$\leq$ 1 alcuni servizi potrebbero non essere adattabili.

Per il caso di studio considerato si ha: MAAS= 4/3.

- ***Media di relativa adattabilità del servizi***o. Infine per la parte dei servizi si misura una media della relativa adattabilità

$$(MRAS)\,\epsilon\mathbb{Q}\{0,....1\}|MRAS = \frac{\sum_{i=1}^{n} RAS_i}{n}$$

Misura utile per definire una media dell'utilizzo dei potenziali componenti.

Per il caso di studio considerato si ha: MRAS= (1+0.5+0.$\dot{6}$)/3.

1.2.1.2 Adattabilità della architettura

Vediamo ora la metrica studiata per valutare l'intera architettura

- ***Livello dell'adattabilità del sistema (LSA)***. Misura del numero di componenti usati per costruire il sistema rispetto al numero di componenti che l'architettura più adattabile userebbe.

$$(LSA)\,\epsilon\mathbb{Q}\{0,....1\}\mid LSA = \frac{\sum_{i=1}^{n} AAS_i}{\sum_{i=1}^{n} \lfloor C_i \rfloor}$$

Il valore di questa metrica rientra in un range tra zero ed uno. Il valore di uno indica che l'architettura sta già utilizzando tutti i componenti esistenti e disponibili e la sua adattabilità è al massimo, un valore vicino ad uno indica che esiste ancora ancora poca scelta sul mercato per poter aumentare il grado di adattamento.

Inoltre per ogni componente che viene aggiunto all'architettura, LSA cresce di $\frac{1}{\sum_{i=1}^{n} C_i}$.

Per il caso di studio considerato si ha: LSA= $0.\dot{6}$

1.2.2 L'adattamento del sistema

Come già spiegato precedentemente molto spesso il raggiungimento di un determinato livello di qualità di un attributo ha un effetto che può essere positivo o negativo, sul raggiungimento del livello di qualità di altri attributi. Di conseguenza un incremento dell'adattabilità dell'intera architettura può incrementare il livello di qualità di alcuni attributi come le prestazioni, affidabilità e manutenibilità e, diminuire il livello di altri. Lo studio affronta in questa parte come approcciarsi all'adattabilità del sistema cercando il miglior compromesso tra adattabilità dell'architettura e mantenimento della qualità del servizio. Si affronta l'analisi basandosi sulle metriche mostrate nel paragrafo precedente.

L'analisi del compromesso tra adattabilità del sistema e mantenimento del livello di qualità degli attributi porta a tre risultati differenti.

Si mostrano in tabella [1.3] gli effetti di una variazione dell'adattabilità rispetto alla misura del livello di qualità di un attributo.

Tabella 1.3: Effetti dell'adattabilità sulla misura della qualità di un requisito [23].

aumento di adattabilità	HIGHER THAN	LOWER THAN
valore di attributo cresce	aiuta	danneggia
valore di attributo diminuisce	danneggia	aiuta
valore di attributo non varia	non ha effetto	non ha effetto

La colonna HIGHER THAN e LOWER THAN indicano come è formulato un attributo.

Quando si parla di HIGHER THAN significa che l'attributo A deve essere maggiore di un certo valore, al contrario quando si parla di LOWER THAN significa che l'attributo A deve essere minore di un certo valore.

Questo significa che un aumento dell'adattabilità del sistema aiuta gli attributi il cui valore deve essere maggiore di un certo numero e danneggia invece gli attributi che devono avere un valore inferiore ad una certa soglia.

I casi migliori sono quelli in cui vi è una totale dipendenza dell'attributo dal sistema. Questi sono:

- caso 1 in cui all'aumentare dell'adattabilità il valore dell'attributo cresce
- caso 2 in cui all'aumentare dell'adattabilità il valore dell'attributo diminuisce.

Sono i casi migliori perchè si hanno a disposizione tutte le informazioni per effettuare l'analisi. In figura [1.4] sono mostrati tutti i casi intermedi rispetto a quelli mostrati in tabella [1.3].

Sull'asse delle X si denota l'incremento del valore dell'adattabilità mentre sull'asse delle Y si indica il valore degli attributi.

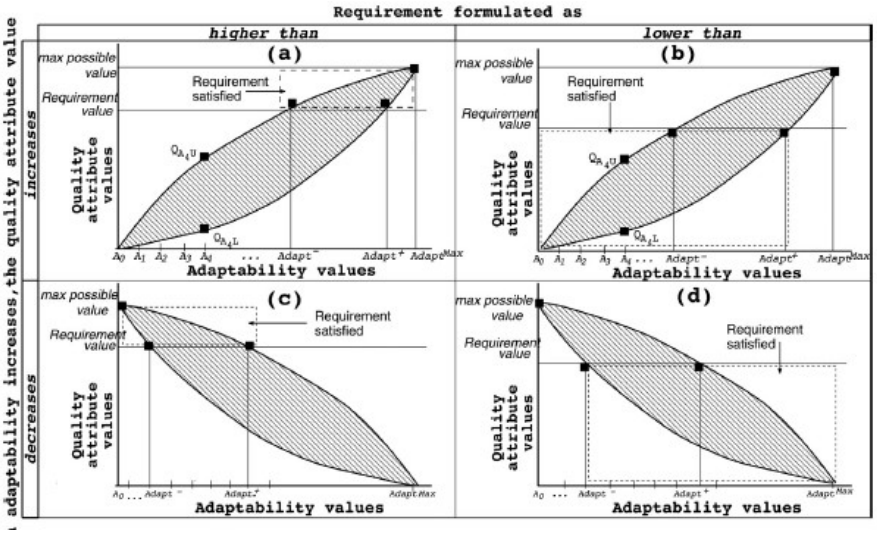

Figura 1.4: Casi intermedi di aumento dell'adattabilità rispetto al valore dell'atributo. [23].

Per ogni valore A_i dell'adattabilità ci si interessa a due valori in particolare:

- Q_{A_iU} ovvero il massimo valore che l'attributo può assumere rispetto al valore di A_i
- Q_{A_iL} ovvero il minimo valore che l'attributo può assumere rispetto al valore di A_i

Tra questi due valori si hanno tutte le architetture con lo stesso valore di adattabilità ma diversi valori per gli attributi.

Tra tutti i valori di Q_{A_iU} e Q_{A_iL} ci si interessa ai valori di:

- *Adapt*$^-$ rappresenta il più basso valore dell'adattabilità di una architettura che soddisfi ancora i requisiti
- *Adapt*$^+$rappresenta il valore più basso dell'adattabilità, compreso tra Q_{A_iU} e Q_{A_iL} che soddisfi i requisiti.

anch'essi presenti nel grafico.

I valori indicano che, per soddisfare i requisiti, una architettura deve assumere valori di adattabilità almeno pari almeno ad *Adapt*$^-$e qualsiasi architettura che abbia un valore di almeno *Adapt*$^+$automaticamente soddisfa i requisiti.

Verrà ora presentato un esempio di studio della relazione esistente tra l'adattabilità del sistema e la sua availability secondo il metodo appena esposto.

1.2.3 Caso di studio della relazione tra adattabilità e requisiti

L'esempio prende in considerazione una applicazione web usata dagli studenti per registrarsi all'anno accademico in una università.

Il sistema è composto da:

- uno livello di presentazione con una interfaccia grafica e meccanismi di interazione con lo studente
- uno livello per la logica applicativa che approva o respinge le richieste dello studente per i corsi da acquistare.

All'inizio lo studente inserisce le sue richieste nel sistema. Queste informazioni vengono mandate alla logica applicativa. Se la loro proposta soddisfa le regole dell'università allora la logica interagisce con i servizi web della banca per procedere al pagamento. Una volta terminato tutto il controllo si ritorna al livello di presentazione, che invia un messaggio e-mail allo studente con le informazioni utili.

Si assume che esistano due componenti per la logica applicativa, un componente per il livello di presentazione; un componente che implementi sia il livello di presentazione che la logica applicativa, due servizi per il pagamento e tre servizi per inviare le email: uno di loro è locale per l'Università e due sono forniti da terzi.

Si assume inoltre, come requisito dell'attributo di qualità, che l'availability del sistema deve essere superiore a 0,9.

Si riassume in tabella [1.4] i servizi utilizzati e componenti indicati nel caso di studio.

Tabella 1.4: Riassunto dei servizi e componenti usati nel caso di studio [23].

descrizione	servizio o componente
registrazione dello studente	s_1
soddisfacimento richiesta dello studente	s_2
inviare una mail	s_3
pagamento in banca	s_4
livello di presentazione	C11
livello di presentazione + logica applicativa	C12
logica applicativa 1	C21
logica applicativa 2	C22
fornitore del componente email di terze parti 1	C31
fornitore del componente email di terze parti 2	C32
fornitore del componente email locale	C33
servizio per il pagamento della banca 1	C41
servizio per il pagamento della banca 2	C42

Per poter proseguire nell'analisi sono necessari alcuni dati aggiuntivi che vengono mostrati in figura [1.5].

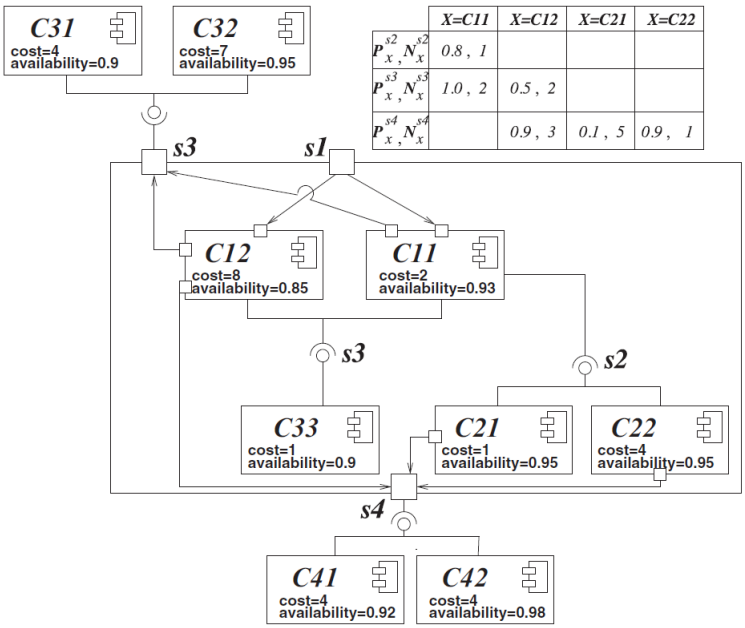

Figura 1.5: Informazioni aggiuntive per calcolare l'availability [23].

Per ogni componente vengono riportati costo e availability direttamente assegnati accanto al nome. In tabella invece si hanno:

- $P_{ij}^{s_k}$ che denota la probabilità con cui il componente C_{ij}richiede il servizio s_k.
- $N_{ij}^{s_k}$ che denota il numero di volte che viene richiesto il servizio s_k.
- L'availability di un componente viene misurata monitorandolo.

- Si assume che non fallisca mai la connessione tra componenti.

Per il calcolo dell'availability si usa un approccio basato sulle reti di petri che permetta di rappresentare l'esecuzione del sistema. Partendo dalla metrica LSA si procede con l'approccio descritto nel paragrafo 2.2.2 e si definisce il grafico presentato in figura [1.6]. Esso indica la relazione esistente tra aumento dell'adattabilità e il valore assunto dal'attributo availability.

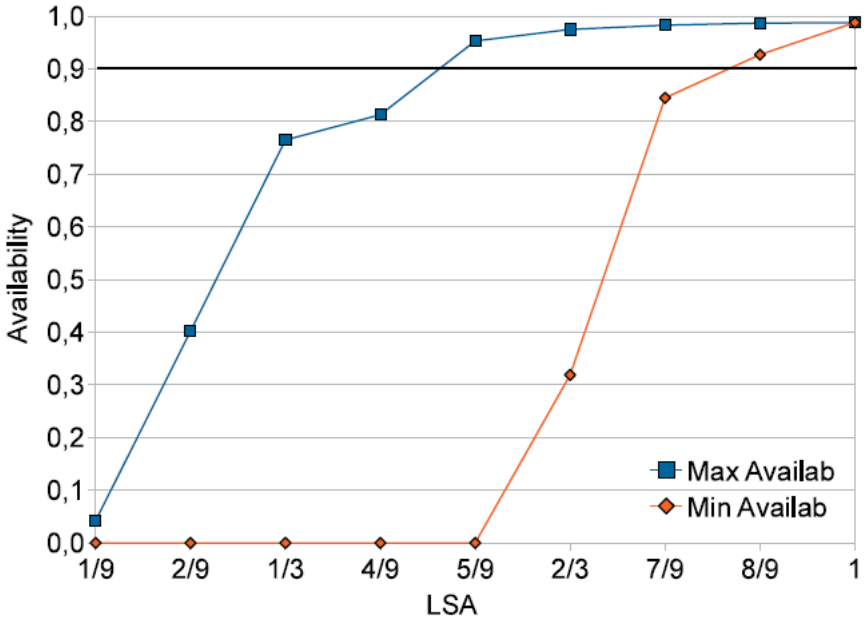

Figura 1.6: Relazione tra adattabilità e availability secondo l'approccio del paragrafo 2.2.2 [23].

Per ottenere questo risultato si parte considerando le architetture con il minimo valore possibile di ***LSA (Livello dell'adattabilità del sistema)***, ovvero quelle architetture composte da un solo componente. Per il componente selezionato si valuta il valore dell'availability. Esso sarà pari a zero se il componente considerato richiede servizi forniti da altri componenti ancora non considerati. Ogni valore trovato di ogni componente viene aggiunto al grafico e si procede iterativamente aggiunge un altro componente del sistema e ricalcolando il valore di LSA. Si itera fino a raggiungere il totale dei componenti disponibili.

Quello che si ottiene è il range per l'adattabilità rispetto all'availability:
Adattabilità w.r.t. availability = $\{\text{adapt}_{min}, \text{adapt}_{max}\}$
Nello specifico si ottiene che:

- con LSA equivalente a 5/9 si ha una $Adapt^{-}$ =5/9 e una availability =0,954 che soddisfa i requisiti. Rispetto al caso di studio LSA è costituito dai componenti UC_1 ={C11,C12}, UC_2 ={C22}, UC_3 ={C32} e UC_4 ={C42}

- per quanto riguarda $Adapt^+$ tutti i valori di LSA $> 7/9$ soddisfano i requisiti.

In questo modo si effettua una scrematura di tutte le possibili architetture adattabili. Tutte le architetture che ottengono un punteggio inferiore al minimo vengono scartate. In linea di massima questo si traduce nello scarto di uno dei componenti disponibili per la sostituzione del componente rotto perchè si assume di sostituire un solo componente rotto alla volta.

1.2.4 Studio dell'adattabilità rispetto a più attributi di qualità

Lo studio ora incrementa il numero di attributi da analizzare e la loro relazione rispetto all'incremento dell'adattabilità dell'architettura. In accordo con la tabella [1.3] si definiscono i valori dei requisiti indicati genericamente con R1 ed R2.

- il requisito R1 viene formulato come HIGHER THAN come «aiuta»
- mentre il requisito R2 viene formulato come LOWER THAN come «danneggia».

Questo significa che rispetto alla figura [1.4] R1 avrà un grafico come nel riquadro (a) mentre per il requisito R2 si avrà un grafico come nel riquadro (b).

L'obiettivo dello studio è trovare tutte le architetture che soddisfano i requisiti sia di R1 che di R2. Si definiscono quindi $Adapt^-$ e $Adapt^+$ sia per R1 che per R2 e si costruisce il grafico di intersezione tra i due requisiti.

In figura [1.7] si riportano tutte le intersezioni possibili tra l'adattabilità e i due requisiti considerati.

Nella figura vengono usati i simboli:

- $\forall$ per indicare la regione dove tutte le architetture soddisfano i requisiti R1 ed R2.
- $\exists$ per indicare la regione dove si può trovare almeno una architettura che, per ogni valore dell'adattabilità, soddisfa i requisiti R1 ed R2.
- $\nexists$ per indicare la regione dove nessuna architettura soddisfa i requisiti R1 ed R2.
- la lente di ingrandimento per indicare la regione dove non è possibile dimostrare l'esistenza di architetture che soddisfino i requisiti.

A questo punto è possibile riconsiderare il caso di studio dell'applicazione web ed analizzare l'adattabilità anche rispetto a un nuovo requisito, che è il costo.

Come fatto per l'availability si definisce il range di soddisfacimento del requisito. Esso infatto è definito soddisfatto quando è LOWER THAN 30 unità.

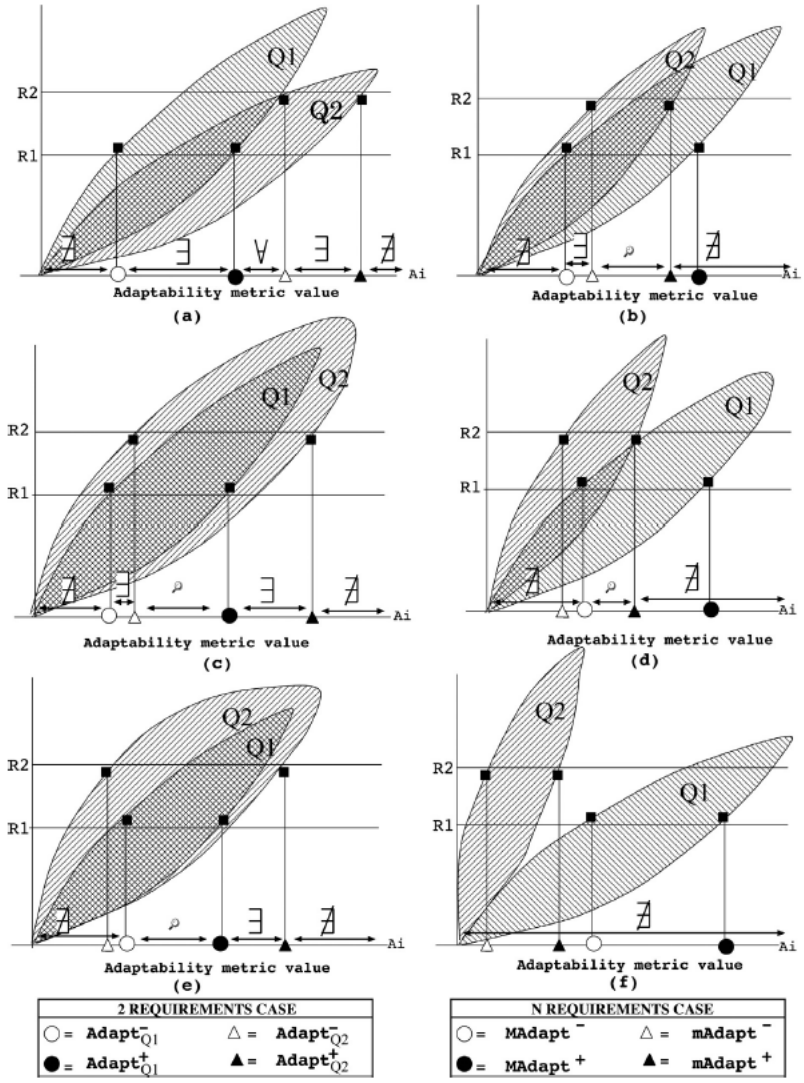

Figura 1.7: Relazione tra adattabilità e due differenti attributi. [23].

Per il calcolo del costo di una architettura si considerano i costi dei singoli componenti che ne fanno parte secondo la seguente formula:

$$Cost = \sum_{i} \sum_{\forall C_j \in UC_i} c_j * cost$$

L'approccio è identico a quello utilizzato per il calcolo dell'availability. Si seguono quindi le indicazioni mostrate nel paragrafo 2.2.2 e si ottiene il grafico presentato in figura [1.8]. Esso indica la relazione esistente tra aumento dell'adattabilità e il valore assunto dal'attributo costo.

Si ottiene un range di adattabilità dell'architettura rispetto al costo dei componenti:

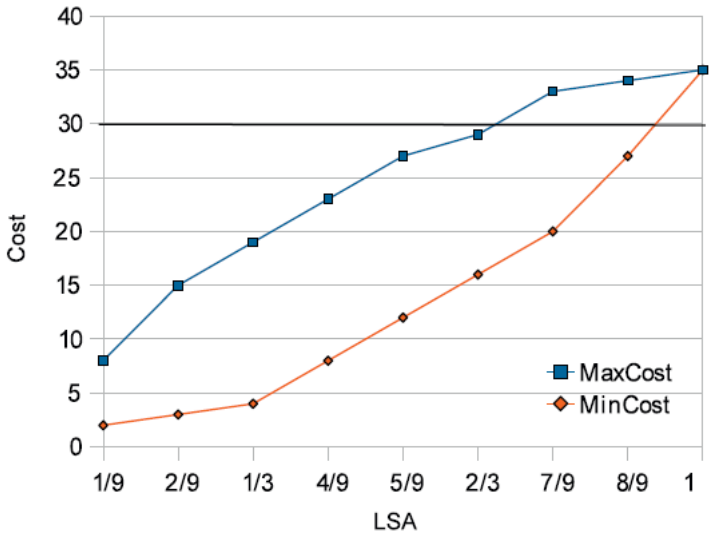

Figura 1.8: Relazione tra adattabilità e costo secondo l'approccio del paragrafo 2.2.2 [23].

Adattabilità w.r.t. costo = {adapt$_{min}$, adapt$_{max}$}.
Nello specifico si ottiene che:

- è possibile trovare soluzioni che soddisfino i requisiti per valori superiori a LSA = 8/9 mentre per tutti i valori di LSA minori di 7/9 i requisiti sono sempre soddisfatti.

A questo punto si applica il metodo appena descritto ovvero si cerca quella regione del grafico dove vengono soddisfatti sia il requisito dell'availability che del costo.
Quello che si ottiene è che:

- non esistono architetture adattabili con LSA inferiore a 5/9 o LSA=1 perchè in questi casi non si soddisfa nessun requisito.
- ci sono delle architetture adattabili per LSA = 5/9, LSA = 6/9 e LSA = 8/9
- possono esistere delle architetture adattabili con LSA = 7/9.

1.2.5 Analisi dell'approccio e i suoi limiti

Questo studio vuole fornire un supporto al progettista del software per la scelta dei componenti da inserire nell'architettura quando cambiano i requisiti o muta l'ambiente in cui è immersa l'applicazione. Lo scopo è mostrare l'insieme delle possibilità al progettista rispetto all'applicazione considerata. Lo studio fornisce tutte le possibili architetture adattabili che soddisfano i requisiti considerati rilevanti per l'applicazione ma la decisione finale viene lasciata al progettista.

Questo approccio richiede probabilmente un tempo di computazione superiore ad altri studi ma fornisce un valido strumento al progettista. Esso però ha dei limiti.

- Dal punto di vista dei requisiti: lo studio considera la soddisfazione dei requisiti come binaria ovvero {soddisfatto, non soddisfatto} quando invece bisognerebbe affrontare questa valutazione nel continuo ovvero con valori che variano da [0,...,1]. Lo studio rappresenta quindi il punto di partenza per ulteriori sviluppi e miglioramenti.

- Dal punto di vista delle metriche di adattabilità: lo studio infatti considera i componenti inseriti nelle architetture come tutti ugualmente importanti senza diversificare il servizio in base al tempo in cui esso è in esecuzione e l'importanza che assume all'interno del sistema. Lo studio infatti non tiene conto del comportamento a runtime dell'applicazione e si limita ad effettuare una analisi solo dal punto di vista statico. Esso osserva i componenti software dell'architettura elementi che forniscono un servizio e/o lo richiedono ma non considera le interazioni tra essi, il numero di volte che vengono utilizzati nel sistema e la loro rilevanza di utilizzo, come pure non considera la possibilità che il componente possa rompersi mentre è in uso o in maniera random.

- Dal punto di vista delle performance del tool: SOLAR permette di analizzare lo spazio delle possibili architetture ma richiede molto tempo di esecuzione.

Nel capitolo due verrà affrontato uno di questi aspetti: l'estensione dell'analisi delle metriche di adattabilità studiando il comportamento a runtime del sistema.

Capitolo 2

Studio del comportamento dinamico del sistema

Le applicazioni moderne nascono per soddisfare dei requisiti di funzionamento che sono soggetti a continui cambiamenti perchè l'ambiente in cui operano è dinamico e muta in maniera imprevedibile. Inoltre sempre più utenti richiedono che le applicazioni siano flessibili e in grado di adattarsi alle loro esigenze senza mai degradare le prestazioni.

Il mondo accademico come le industrie si sono interrogati su come fornire una soluzione a questa crescente richiesta e hanno introdotto i sistemi auto - adattativi.

Il requisito principale di questi sistemi è la capacità di adattare il proprio comportamento in maniera dinamica per continuare a mantenere la stessa qualità del servizio fornito a fronte di qualunque cambiamento dell'ambiente o dei requisiti richiesti. Risulta chiaro che senza questa capacità di adattamento l'applicazione avrà un degrado delle prestazioni perchè non potrà sostituire i componenti difettosi.

Tuttavia, fornire la capacità di adattamento del software può influenzare altri attributi di qualità come le prestazioni, l'affidabilità o manutenibilità. Come sottolineato nel capitolo uno va trovato il miglior equilibrio tra i diversi requisiti di qualità che un sistema deve soddisfare e la sua adattabilità.

Di per se questo è già un obiettivo ambizioso.

Va inoltre aggiunto che tali sistemi sono altamente distribuiti, il che aumenta la complessità di questi sistemi rendendo la loro realizzazione, configurazione e manutenzione davvero difficile.

Oggigiorno esistono molte metriche riconosciute da ISO/IEC 9126 per la valutazione della qualità del software e delle applicazioni anche per sistemi distribuiti. Le metriche vengono mostrate in figura [2.1].

Tuttavia nell'ambito di studio dei sistemi auto - adattativi, l'analisi delle metriche di adattabilità è un campo relativamente nuovo e non può far uso di queste metriche in quanto non tengono conto della caratteristica principale di questi sistemi

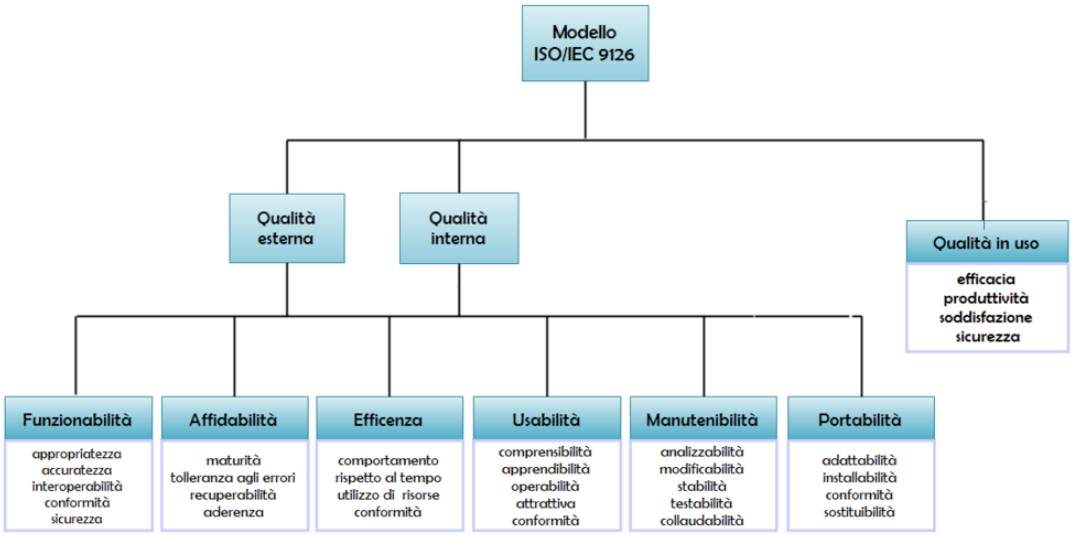

Figura 2.1: Metriche definite dall'ISO/IEC 9126 per l'analisi del software [10].

ovvero la loro capacità di auto - adattarsi. Per questo motivo e per i motivi sopra elencati, ancora non esistono delle metriche di adattabilità riconosciute e adottate universalmente ma solo degli studi di approcci a tali sistemi.

Nel capitolo uno si è presentato un approccio all'analisi di un sistema auto - adattativo limitandosi a considerarlo come statico ovvero senza valutare come si comporta l'applicazione quando è eseguita.

Il mio contributo in questo studio è quello di analizzare il comportamento dinamico del sistema. Ho ritenuto importante studiare la dinamica del sistema per aiutare il progettista di sistema ad effettuare una scelta più accurata dei componenti da inserire nell'architettura. Sono partita dall'assunzione che i servizi vengono forniti dai componenti software e che non si possa effettuare una scelta sui componenti prescindendo dall'analisi di tutte le sue caratteristiche. Nel capitolo uno si è studiato l'adattabilità del sistema rispetto all'availability e al costo dei componenti software ma non si è fatto menzione dei servizi forniti da tali componenti. Ho voluto porre l'attenzione sui limiti di SOLAR cercando di superarli, almeno per quanto riguarda le metriche di adattabilità. Ho pensato fosse poco accurato considerare tutti i servizi importanti allo stesso modo.

Per questo motivo il mio approccio è basato sullo studio dell'importanza che il funzionamento del servizio assume mentre l'applicazione è in esecuzione. A tale fine ho effettuato una diversificazione dei servizi forniti e richiesti nell'architettura valutando la possibilità di avere una rottura del componente software nella fruizione del servizio in due casi differenti ovvero alla chiamata del servizio e in maniera casuale.

L'obiettivo finale è quello di definire delle metriche di adattabilità sia per l'architettura che per i servizi. Nello specifico si vuole mostrare il grado di importanza di ogni servizio all'interno dell'architettura e di riflesso il grado di importanza del componente software che lo fornisce. Si vuole dare al progettista un'informazione aggiuntiva utile nella scelta del componente. Il mio studio effettua due analisi di adattabilità dell'architettura separate, una rispetto al requisito di availability e una rispetto al requisito di costo. Questo perchè parto da un'altra considerazione ovvero che: il costo maggiore di un componente in linea teorica rappresenta una migliore qualità del servizio fornito che si traduce in un migliore valore di availability. Sempre a livello teorico ho assunto che questo migliore valore di availability indichi che il componente può fornire il servizio per un tempo maggiore e con migliori prestazioni. In un discorso di adattabilità del sistema ho logicamente dedotto che un componente che fornisce il servizio più a lungo e con migliori prestazioni risulta più appetibile per essere aggiunto nell'architettura. Tutto questo viene fatto per permettere al progettista di conoscere quale o quali sono i servizi maggiormente importanti nel funzionamento dell'applicazione e, poter scegliere le architetture adattabili basandosi:

- sui requisiti di availability e costo di tutti i componenti forniti (secondo l'approccio mostrato nel capitolo uno)
- scegliendo i componenti che forniscono l'availability migliore soprattutto per quei servizi ritenuti importanti per il sistema prescindendo dal soddisfacimento del requisito del costo
- effettuando una scelta ibrida cercando il miglior equilibrio tra availability, costo e importanza del servizio fornito.

Come spiegato, il campo di studio dei sistemi auto - adattativi è relativamente nuovo e non ci sono ancora metriche di adattabilità collaudate e definite universalmente per valutare l'adattabilità per questo motivo tutto il mio studio è basato su assunzioni e considerazioni logiche di cosa possa essere importante analizzare per un sistema che si deve auto - adattare.

Come fatto per lo studio esposto nel capitolo uno, ho voluto fornire al progettista di sistema un framework che analizzi in maniera automatica le architetture adattabili rispetto al mio studio.

Il framework estende SOLAR e verrà presentato nel capitolo quattro con il nome di ADAPT_ANALYZER.

2.1 Introduzione

Questa parte dello studio introduce il mio contributo alla realizzazione di metriche di adattabilità per i sistemi auto - adattativi. L'approccio pone l'attenzione sul

comportamento dinamico dell'applicazione quando è in esecuzione.

Come esposto alla fine del capitolo precedente, l'analisi puramente statica di un sistema non tiene conto delle interazioni tra i vari componenti software e non si riesce pertanto a cogliere una significativa caratteristica di un componente ovvero il suo grado di rilevanza nel sistema.

Le metriche presentate di seguito effettuano una analisi iniziale dello spazio delle architetture che si possono costruire basandosi su di un insieme di componenti software. Poi si effettua l'analisi innovativa sui servizi forniti dai componenti software. Infine si studia l'adattabilità del sistema rispetto ai requisiti di availability e costo e si calcola la rilevanza del servizio rispetto ai risultati delle metriche che verranno presentate nell'analisi dei servizi forniti.

In questo capitolo vengono spiegate le metriche di adattabilità per lo studio del comportamento dinamico e si rimanda al capitolo tre per la rappresentazione del caso di studio.

2.2 Le metriche: analisi dinamica del sistema

Prima di elencare le metriche è bene fare una premessa: tutte le metriche sono valutate rispetto ai valori di due parametri target che il componente deve rispettare. Essi sono:

- $availability_{target}$ per l'availability
- $cost_{target}$ per il costo.

I valori di tali parametri sono lasciati alla scelta del progettista di sistema che li definirà in base al tipo di sistema reale che si sta modellando.

Come per il capitolo uno si presenta in tabella [2.1] un riassunto dei termini utilizzati.

2.2.1 Analisi iniziale dello spazio delle possibilità

Si effettua ora una analisi dei componenti software che possono essere utilizzati nell'architettura e poi si effettua l'analisi iniziale dello spazio delle architetture.

Per poter mostrare degli esempi di come vengono calcolate le metriche ci si basa sul seguente insieme di componenti che è un sotto insieme dei componenti che verranno poi mostrati nel caso di studio completo presentato nel capitolo tre.

- C_{11} con $availability_{C_{11}} = 0,93$, $cost_{C_{11}} = 2$, $execution\_time_{C_{11}} = 1$,
  - fornisce il servizio s_1
  - richiede il servizio s_2 con $P_{C_{11}-S_2}$= 0,8 e $N_{C_{11}-S_2}$= 1
  - richiede il servizio s_3 con $P_{C_{11}-S_3}$= 1 e $N_{C_{11}-S_3}$= 2

Tabella 2.1: Riassunto dei termini utilizzati nel capitolo 2.

	significato
$availability_{target}$	valore di riferimento per il requisito di availability
$cost_{target}$	valore di riferimento per il requisito di costo
av_{Ci}	availability del componente $C_{i-esimo}$
$nome-metrica_{Ci-s_j}$	indica che si prendono in considerazione tutti i servizi forniti dal componente $C_{i-esimo}$
C_i	componente i-esimo
c_{Ci}	costo del componente $C_{i-esimo}$
A_i	architettura i-esima dello spazio delle possibili architetture
P_{Ci-s_j}	probabilità del componente $C_{i-esimo}$ di richiamare il servizio s_j
N_{Ci-s_j}	numero di volte in cui il componente $C_{i-esimo}$ richiama il servizio s_j
N_{Ci}	numero di volte in cui il componente $C_{i-esimo}$ è in esecuzione
T_{Ci}	tempo di esecuzione del componente $C_{i-esimo}$
P_{path_i}	probabilità di esecuzione del $path_{i-esimo}$ alternativo
m_{path_i}	numero totale di volte con cui l' $attività_{i-esima}$ è eseguita nel $path_{i-esimo}$
u_{Si-act_i}	numero di volte con cui il servizio $s_{i-esimo}$ viene eseguito nell' $attività_{i-esima}$
A_a	architettura attualmente in uso
C_r	componente da sostituire
C_{nu-Sr}	componente non usato nell'architettura che fornisce il medesimo servizio di C_r
W_1	peso attribuito all'importanza del requisito di availability
W_2	peso attribuito all'importanza del requisito di costo
W_3	peso attribuito all'importanza della probabilità che il componente si rompa alla chiamata
W_4	peso attribuito all'importanza del reale utilizzo del componente nel sistema

- C_{12} con $availability_{C_{12}} = 0,85$, $cost_{C_{12}} = 8$, $execution\_time_{C_{12}} = 3$,
  - fornisce il servizio s_1
  - fornisce il servizio s_2
  - richiede il servizio s_3 con $P_{C_{12}-S_3}$= 0,5 e $N_{C_{12}-S_3}$= 2
  - richiede il servizio s_4 con $P_{C_{12}-S_4}$= 0,9 e $N_{C_{12}-S_4}$= 3
- C_{31} con $availability_{C_{31}} = 0,9$, $cost_{C_{31}} = 4$, $execution\_time_{C_{31}} = 1,5$,
  - fornisce il servizio s_3
- C_{32} con $availability_{C_{32}} = 0,95$, $cost_{C_{32}} = 7$, $execution\_time_{C_{32}} = 2$,
  - fornisce il servizio s_3
- C_{41} con $availability_{C_{41}} = 0,92$ $cost_{C_{41}} = 4$, $execution\_time_{C_{41}} = 2$,
  - fornisce il servizio s_4

2.2.1.1 Analisi iniziale dei componenti

Per quanto riguarda i componenti si hanno:

- ***Fitness ratio w.r.t. availability.*** Questa metrica definisce il rapporto esistente tra l'availability di un componente e l'$availability_{target}$ del sistema.

$$(FRA)\,\epsilon\mathbb{N}^{n}|\{FRA_{Ci} = \frac{1 - availability_{target}}{1 - av_{Ci}}\}$$

Questa metrica è il risultato di un'ipotesi che afferma che un componente software con availability intrinseca alta possa a primo acchito dare maggiori garanzie di funzionamento. In altre parole questo equivale a dire che, se il componente dovesse soddisfare il requisito target allora la probabilità di guasto nel funzionamento del servizio potrebbe diminuire o quantomeno allungarsi il tempo vita della fruizione del servizio tramite quel componente. Di riflesso questo si tradurrebbe in un miglioramento delle prestazioni del sistema.

Per poter apprezzare il significato si effettuano i calcoli sull'insieme dei componenti presi come esempio.

- $esempio_1$:
  - C_{11} con $availability_{C_{11}} = 0,93$ e $availability_{target} = 0{,}9$
  - C_{12} con $availability_{C_{12}} = 0,85$ e $availability_{target} = 0{,}9$
  - C_{31} con $availability_{C_{31}} = 0,9$ e $availability_{target} = 0{,}9$
  - C_{32} con $availability_{C_{32}} = 0,95$ e $availability_{target} = 0{,}9$
  - C_{41} con $availability_{C_{41}} = 0{,}92$ e $availability_{target} = 0{,}9$
- $esempio_2$:
  - C_{11} con $availability_{C_{11}} = 0,93$ e $availability_{target} = 0{,}99$
  - C_{12} con $availability_{C_{12}} = 0,85$ e $availability_{target} = 0{,}99$
  - C_{31} con $availability_{C_{31}} = 0,9$ e $availability_{target} = 0{,}99$
  - C_{32} con $availability_{C_{32}} = 0,95$ e $availability_{target} = 0{,}99$
  - C_{41} con $availability_{C_{41}} = 0{,}92$ e $availability_{target} = 0{,}99$

Calcoliamo ora i valori della metrica FRA per ogni esempio:

- $esempio_1$:
  - $FRA_{C_{11}} = 1{,}4285$
  - $FRA_{C_{12}} = 0{,}6666$
  - $FRA_{C_{31}} = 1{,}0$
  - $FRA_{C_{32}} = 2{,}0$
  - $FRA_{C_{41}} = 1{,}2500$
- $esempio_2$:

- $FRA_{C_{11}}$= 0,1428
- $FRA_{C_{12}}$= 0,0666
- $FRA_{C_{31}}$= 0,1
- $FRA_{C_{32}}$= 0,2
- $FRA_{C_{41}}$= 0,1250

La metrica quindi mostra il valore intrinseco del rapporto tra $availability_{target}$ e av_{C_i}.

Come è possibile notare, a parità di valore di av_{C_i}, all'aumentare del valore di $availability_{target}$ diminuisce il valore assunto dalla metrica FRA. Questo significa che maggiore è il divario tra il valore di $availability_{target}$ e il valore dell'availability del componente considerato e minore è la capacità del componente di soddisfare il requisito di availability.

A fronte dell'ipotesi fatta si cercano dei componenti che soddisfino il requisito di availability. Per questo motivo si introduce la seguente metrica:

- ***Boolean suitability w.r.t. availability.*** Questa metrica definisce l'idoneità di un componente a prendere parte al sistema solo rispetto all'$availability_{target}$.

$$(BSA)\,\epsilon\{0,1\}¦\{BSA_{Ci} = 1 \iff (FRA_{Ci} \geq 1) \vee BSA_{Ci} = 0 \iff (FRA_{Ci} < 1)\}$$

L'idoneità del componente rispetto all'availability viene decretata in base al valore della metrica BSA. Essa infatti parte dal rapporto tra il valore di $availability_{target}$ e il valore dell'availability del componente considerato, espresso con la metrica FRA e, stabilisce che un componente viene considerato idoneo (BSA = 1) solo quando il valore di FRA eguaglia o addirittura supera il valore 1.

La metrica FRA assume valore uguale o superiore a 1 solo quando il valore dell'availability del componente considerato supera il valore dell' $availability_{target}$.

Consideriamo nuovamente gli esempi appena mostrati:

- $esempio_1$:
  - $FRA_{C_{11}}$= 1,4285 ==> $BSA_{C_{11}} = 1$
  - $FRA_{C_{12}}$= 0,6666 ==> $BSA_{C_{12}} = 0$
  - $FRA_{C_{31}}$= 1,0 ==> $BSA_{C_{31}} = 1$
  - $FRA_{C_{32}}$= 2,0 ==> $BSA_{C_{32}} = 1$
  - $FRA_{C_{41}}$= 1,2500 ==> $BSA_{C_{41}} = 1$
- $esempio_2$:
  - $FRA_{C_{11}}$= 0,1428 ==> $BSA_{C_{11}} = 0$
  - $FRA_{C_{12}}$= 0,0666 ==> $BSA_{C_{12}} = 0$

- $FRA_{C_{31}}$= 0,1 ==> $BSA_{C_{31}} = 0$
- $FRA_{C_{32}}$= 0,2 ==> $BSA_{C_{32}} = 0$
- $FRA_{C_{41}}$= 0,1250 ==> $BSA_{C_{41}} = 0$

Seguendo l'ipotesi fatta, nessun componente soddisfa i requisiti di availability quando il valore è $availability_{target} = 0{,}99$.

Quando invece il valore dell' $availability_{target} = 0{,}9$ allora solo il componente C_{12} non risulta idoneo e quindi non fornisce garanzie di funzionamento adeguate. Mentre invece il componente C_{32} fornisce il valore più alto per la metrica FRA e, a prima vista dà maggiori garanzie di funzionamento. In una logica di miglioramento delle prestazioni del sistema nella fornitura del servizio, fa sperare che il servizio venga fornito con migliore qualità o quantomeno per una durata maggiore.

L'idoneità di un componente a fornire un servizio è rimandata a dopo in quanto vanno considerati i path con cui è possibile fornire il servizio.

Procedendo in modo logico, si valuta ora quanto un componente sia idoneo rispetto al costo.

- ***Fitness ratio w.r.t. cost.*** Questa metrica definisce il rapporto esistente tra il costo di un componente e il costo target $cost_{target}$ del sistema.

$$(FRC)\,\epsilon\mathbb{N}^n|\{FRC_{Ci} = \frac{cost_{target}}{c_{ci}}\}$$

In modo del tutto simile a come è stata trattata la metrica FRA si mostrano gli esempi di come viene calcolata tale metrica sempre basandosi sull'insieme dei componenti.

- $esempio_1$:
  - C_{11} con $cost_{C_{11}} = 2$ e $cost_{target} = 30$
  - C_{12} con $cost_{C_{12}} = 8$ e $cost_{target} = 30$
  - C_{31} con $cost_{C_{31}} = 4$ e $cost_{target} = 30$
  - C_{32} con $cost_{C_{32}} = 7$ e $cost_{target} = 30$
  - C_{41} con $cost_{C_{41}} = 4$ e $cost_{target} = 30$
- $esempio_2$:
  - C_{11} con $cost_{C_{11}} = 2$ e $cost_{target} = 38$
  - C_{12} con $cost_{C_{12}} = 8$ e $cost_{target} = 38$
  - C_{31} con $cost_{C_{31}} = 4$ e $cost_{target} = 38$
  - C_{32} con $cost_{C_{32}} = 7$ e $cost_{target} = 38$
  - C_{41} con $cost_{C_{41}} = 4$ e $cost_{target} = 38$

Calcoliamo quindi i valori di FRC:

- $esempio_1$:
  - $FRC_{C_{11}}$= 15
  - $FRC_{C_{12}}$= 3,75
  - $FRC_{C_{31}}$= 7,5
  - $FRC_{C_{32}}$= 4,2857
  - $FRC_{C_{41}}$= 7,5
- $esempio_2$:
  - $FRC_{C_{11}}$= 19
  - $FRC_{C_{12}}$= 4,75
  - $FRC_{C_{31}}$= 9,5
  - $FRC_{C_{32}}$= 5,43
  - $FRC_{C_{41}}$= 9,5

Questa metrica mostra il valore intrinseco del rapporto tra $cost_{target}$ e c_{C_i}.

Come è possibile notare, a parità di valore di c_{C_i}, all'aumentare del valore di $cost_{target}$ il valore della metrica FRC aumenta e non diminuisce come invece avveniva per la metrica FRA. Questo significa che maggiore è il divario tra il valore di $cost_{target}$ e il valore del costo del componente considerato e maggiore sarà il risparmio nell'acquisto di quel componente rispetto al budget massimo inserito per acquisire ogni componente.

Questa metrica mostra quindi il rapporto esistente tra budget massimo di acquisto di ogni componente e il costo dei componenti inseriti nell'architettura. Risulta chiaro che, se il progettista fosse portato a scegliere i componenti avendo come unico vincolo stringente il costo dei componenti stessi, dovrebbe selezionare solo quelli che abbiano un costo inferiore al $cost_{target}$. Per poter indicare questa idoneità si ha la metrica :

- ***Boolean suitability w.r.t. cost.*** Questa metrica definisce l'idoneità di un componente a prendere parte al sistema solo rispetto al $cost_{target}$.

$$(BSC)\,\epsilon\{0,1\}\text{¦}\{BSC_{Ci}=1 \iff (FRC_{Ci}\geq 1)\vee BSC_{Ci}=0 \iff (FRC_{Ci}<1)\}.$$

L'idoneità del componente rispetto al costo viene decretata in base al valore della metrica BSC.

Essa infatti parte dal rapporto tra il valore di $cost_{target}$ e il valore del costo del componente considerato, espresso con la metrica FRC e, stabilisce che un componente viene considerato idoneo (BSC = 1) solo quando il valore di FRC eguaglia o addirittura supera il valore 1.

La metrica FRC assume valore uguale o superiore a 1 solo quando il valore del costo del componente considerato è inferiore al valore del $cost_{target}$.

Consideriamo nuovamente gli esempi appena mostrati:

- $esempio_1$:
  - $FRC_{C_{11}}$= 15 ==> $BSC_{C_{11}} = 1$
  - $FRC_{C_{12}}$= 3,75 ==> $BSC_{C_{12}} = 1$
  - $FRC_{C_{31}}$= 7,5 ==> $BSC_{C_{31}} = 1$
  - $FRC_{C_{32}}$= 4,2857 ==> $BSC_{C_{32}} = 1$
  - $FRC_{C_{41}}$= 7,5 ==> $BSC_{C_{41}} = 1$
- $esempio_2$:
  - $FRC_{C_{11}}$= 19 ==> $BSC_{C_{11}} = 1$
  - $FRC_{C_{12}}$= 4,75 ==> $BSC_{C_{12}} = 1$
  - $FRC_{C_{31}}$= 9,5 ==> $BSC_{C_{31}} = 1$
  - $FRC_{C_{32}}$= 5,43 ==> $BSC_{C_{32}} = 1$
  - $FRC_{C_{41}}$= 9,5 ==> $BSC_{C_{41}} = 1$

Ora possiamo affermare che ogni componente è idoneo per il requisito di costo.

Per quanto riguarda lo studio il requisito di costo viene considerato meno stringente rispetto a quello dell'availability perchè si prosegue seguendo l'ipotesi fatta: un componente con miglior availability fornisce a prima vista migliori garanzie di funzionamento per il servizio e, a fronte di un miglior valore di availability solitamente corrisponde un costo maggiore del componente.

Non si escludono questi componenti dallo studio proprio perchè il mio approccio è relativo all'importanza rivestita dai servizi all'interno dell'architettura e si vuole mostrare al progettista di sistema anche i componenti più costosi che forniscono un valore di availability più alto così da lasciare al progettista la possibilità di scegliere se:

- acquistare un componente più costoso ma allo stesso tempo con miglior valore di availability per quei servizi maggiormente utilizzati nell'architettura
- acquistare un componente secondo l'approccio mostrato nel capitolo uno ovvero indicando availability e costo come unici valori stringenti.

2.2.1.2 Analisi iniziale delle architetture

Dopo aver analizzato il singolo componente rispetto al requisito di availability e rispetto al requisito di costo, si analizza l'intero sistema sempre suddividendo i due requisiti.

- ***Global availability of system.*** Questa metrica definisce l'availability congiunta dei componenti del sistema come la probabilità che siano tutti attivi in quell'istante.

$$(GAS)\,\epsilon\mathbb{N}\{GAS = \prod FRA_{Ci}\}$$

Per poter apprezzare il significato di questa metrica vengono ora presentati due esempi di architetture basandosi sul valore di $availability_{target} = 0{,}9$:

- $architettura_1$composta da:
  - C_{11} con $availability_{C_{11}} = 0,93$, $FRA_{C_{11}}$= 1,4285 ==> $BSA_{C_{11}} = 1$
  - C_{12} con $availability_{C_{12}} = 0,85$, $FRA_{C_{12}}$= 0,6666 ==> $BSA_{C_{12}} = 0$
  - C_{31} con $availability_{C_{31}} = 0,9$, $FRA_{C_{31}}$= 1,0 ==> $BSA_{C_{31}} = 1$
  - C_{41} con $availability_{C_{41}} = 0{,}92$, $FRA_{C_{41}}$= 1,2500 ==> $BSA_{C_{41}} = 1$
- $architettura_2$composta da:
  - C_{11} con $availability_{C_{11}} = 0,93$, $FRA_{C_{11}}$= 1,4285 ==> $BSA_{C_{11}} = 1$
  - C_{12} con $availability_{C_{12}} = 0,85$, $FRA_{C_{12}}$= 0,6666 ==> $BSA_{C_{12}} = 0$
  - C_{32} con $availability_{C_{32}} = 0,95$, $FRC_{C_{32}}$= 4,2857 ==> $BSC_{C_{32}} = 1$
  - C_{41} con $availability_{C_{41}} = 0{,}92$, $FRA_{C_{41}}$= 1,2500 ==> $BSA_{C_{41}} = 1$

Calcoliamo ora i valori di $GAS_{architettura_1}$ e di $GAS_{architettura_2}$.

- $GAS_{architettura_1}$ = 1,1905
- $GAS_{architettura_2}$ = 2,3810

La metrica GAS insiste sul rapporto tra $availability_{target}$ e il valore dell'availability dei componenti.

Come è possibile notare, la differenza tra il valore di GAS della prima architettura e il valore di GAS della seconda architettura dipende dalla sostituzione del componente C_{31} con il componente C_{32} rispettivamente.

Ancora una volta un miglior valore dell'availability del singolo componente implica un valore di FRA maggiore che si avverte in maniera più sensibile nella metrica GAS.

Tale metrica quindi permette di identificare le architetture con un valore di availability più elevato e, in base all'ipotesi iniziale, esse rappresentano quelle che a primo avviso danno maggiori garanzie di mantenere alte le prestazioni e la qualità del servizio fornito.

Allo stesso modo si tratta il requisito del costo

- ***Global cost of system.*** Questa metrica definisce il costo totale dei componenti del sistema come la somma dei costi dei singoli componenti rispetto al $cost_{target}$ assegnato.

$$(GCS)\,\epsilon\mathbb{N}\{GCS = \sum FRC_{Ci}\}$$

Per poter apprezzare il significato di questa metrica vengono ora presentati due esempi di architetture basandosi sul valore di $cost_{target} = 30$:

- $architettura_1$composta da:
  - C_{11} con $cost_{C_{11}} = 2$, $FRC_{C_{11}}$= 15 ==> $BSC_{C_{11}} = 1$
  - C_{12} con $cost_{C_{12}} = 8$, $FRC_{C_{12}}$= 3,75 ==> $BSC_{C_{12}} = 1$
  - C_{31} con $cost_{C_{31}} = 4$, $FRC_{C_{31}}$= 7,5 ==> $BSC_{C_{31}} = 1$
  - C_{41} con $cost_{C_{41}} = 4$, $FRC_{C_{41}}$= 7,5 ==> $BSC_{C_{41}} = 1$
- $architettura_2$composta da:
  - C_{11} con $cost_{C_{11}} = 2$, $FRC_{C_{11}}$= 15 ==> $BSC_{C_{11}} = 1$
  - C_{12} con $cost_{C_{12}} = 8$, $FRC_{C_{12}}$= 3,75 ==> $BSC_{C_{12}} = 1$
  - C_{32} con $cost_{C_{32}} = 7$, $FRA_{C_{32}}$= 2,0 ==> $BSA_{C_{32}} = 1$
  - C_{41} con $cost_{C_{41}} = 4$, $FRC_{C_{41}}$= 7,5 ==> $BSC_{C_{41}} = 1$

Calcoliamo ora i valori di $GCS_{architettura_1}$ e di $GCS_{architettura_2}$.

- $GCS_{architettura_1}$ = 33,75
- $GCS_{architettura_2}$ = 30,5357

Il valore della metrica GCS aumenta con l'aumentare del rapporto FRC tra $cost_{target}$ e costo dei componenti dell'architettura.

In questo esempio il valore finale di GCS per l'$architettura_1$ e per l'$architettura_2$ è diverso. Si riassumono i dati delle due architetture per poterli analizzare:

- $architettura_1$:$GAS_{architettura_1}$= 1,1905 e $GCS_{architettura_1}$= 33,75
- $architettura_2$:$GAS_{architettura_2}$= 2,3810 e $GCS_{architettura_2}$= 30,5357

Questo è il caso più comune in cui il progettista possa incorrere. Infatti spetta a lui scegliere quale architettura scegliere che abbia il miglior compromesso tra i due requisiti.

Presentiamo ora un altro caso di analisi generico, supponiamo di avere:

- $architettura_1$:$GAS_{architettura_1}$= 3,57 e $GCS_{architettura_1}$= 60,0
- $architettura_2$:$GAS_{architettura_2}$= 14,29 e $GCS_{architettura_2}$= 60,0

La differerenza con il caso precedente è che il valore di GCS di entrambe le architetture ora è identico.

Questo è sicuramente il caso più fortunato in cui il progettista possa incorrere in quanto le metriche esprimono che, a parità di guadagno nell'acquisto dei componenti che si traduce come lo stesso valore di GCS, si può scegliere l'architettura che dà maggiori garanzie di prestazioni e qualità di servizio fornito esprimibile con il valore di GAS più elevato.

Risulta chiaro che in condizioni come il caso appena esposto la scelta risulta ovvia, purtroppo però non è un caso così comune anzi, la maggior parte delle volte il progettista si trova davanti un caso come il primo esposto con valori di GAS e GCS differenti per ogni architettura. Lo studio vuole supportare il progettista nella scelta dei componenti propri in casi come il primo esposto evidenziando le metriche di valutazione.

A questo punto lo studio vuole mostrare tutte le architetture considerate idonee per l'adattabilità rispetto all'availability e rispetto al costo e lo fa con due metriche che riassumono in un valore scalare la cardinalità dello spazio delle possibili architetture adattabili rispetto ai due requisiti.

- ***Space of possibility of system availability.*** Questa metrica permette di ottenere il valore della cardinalità dell'insieme di tutte le architetture considerate idonee rispetto al requisito di availability.

$$(SPA)\,\epsilon\mathbb{N}|\{SPA = \sum A_i|GAS_{A_i} \geq 1\}$$

- ***Space of possibility of system cost.*** Questa metrica, in modo equivalente, permette di ottenere il valore della cardinalità dell'insieme di tutte le architetture considerate idonee rispetto al requisito di costo

$$(SPC)\,\epsilon\mathbb{N}|\{SPC = \sum A_i|GCS_{A_i} \geq 1\}$$

Riprendo l'esempio precedente in cui si hanno:

- $architettura_1$:$GAS_{architettura_1}$= 1,1905 e $GCS_{architettura_1}$= 33,75
- $architettura_2$:$GAS_{architettura_2}$= 2,3810 e $GCS_{architettura_2}$= 30,5357

I valori di SPA e SPC che si ottengono sono entrambi uguali a 2 in quanto entrambe le architetture presentano valori di GAS e GCS uguali o superiori a 1.

Questa metrica serve da riassunto dello studio fatto finora per poter indicare in maniera sintetica tramite un numero quante sono tutte le architetture che soddisfano rispettivamente il requisito di availability e di costo.

Lo spazio totale delle architetture risulterà utile più avanti per il calcolo di due indici:

- l'indice di adattabilità di qualsiasi architettura ovvero quante sono le architetture verso cui poter migrare che si differenziano da quella presa in considerazione solo per un componente. In questo indice verranno prese in considerazione solo le architetture che appartengono agli spazi SPA ed SPC.

- l'indice di adattabilità dei servizi rispetto alle alternative.

2.2.2 Analisi dei servizi

Dopo aver analizzato nel dettaglio tutto lo spazio delle architetture, in questa parte si prosegue lo studio concentrandosi sui servizi forniti da una architettura. Questa parte rappresenta il fulcro di tutto il mio studio in quanto mi concentro sull'analisi dei servizi. Quello che si vuole mostrare in questa sezione è il differente grado di importanza assunto dai servizi durante la loro esecuzione nell'applicazione. Con lo studio esposto nel capitolo uno si analizzano i componenti come tutti ugualmente importanti fra loro e di conseguenza anche il servizio che forniscono hanno la stessa importanza. In questa parte si vuole scendere nel dettaglio dell'analisi dei servizi studiandone il comportamento durante l'esecuzione del sistema.

Ciò che si vuole mettere in evidenza è il servizio che più si distingue dagli altri per:

- probabilità di essere in funzione

- peso della permanenza nel sistema inteso come quante volte il servizio entrerà in funzione nell'applicazione

Questi valori risultano utili per calcolare una metrica che verrà presentata nell'analisi finale e renderà noto al progettista quale servizio acquisisce maggiore importanza rispetto agli altri. Tutto lo studio segue la considerazione finale di voler permettere al progettista una scelta più accurata dei componenti da inserire nell'architettura e renderlo consapevole di quale servizio ha bisogno di un componente che dia maggiori garanzie di continuare a fruirlo. Questo perchè, se dovesse fallire un componente che fornisce un servizio usato poche volte, il degrado delle prestazioni dell'intero sistema non ne risentirebbero molto e si può pensare di sostituirlo con un qualsiasi altro componente che rispetti i requisiti di availability e costo mentre invece il degrado delle prestazioni del sistema diventa percepibile a fronte del fallimento di un servizio molto utilizzato dall'applicazione. Ecco il perchè di questa sezione, far sapere al progettista quale servizio può inficiare maggiormente sulle prestazioni del sistema e lasciare al progettista libero arbitrio sulla scelta del componente adeguato: se rispettare entrambi i vincoli di availability e costo o seguire l'ipotesi fatta, ovvero che un componente più costoso e con maggior valore di availability possa fornire maggiori garanzie di prestazioni per il servizio considerato e, quindi optare per la

scelta di un componente che non rispetti il vincolo di costo ma dia maggior garanzie di prestazione.

Sulla base di quanto detto sono state studiate le seguenti metriche:

- ***Number of executions.*** Questa metrica indica il numero di volte con cui un servizio viene richiamato all'interno dell'architettura considerata.

$$N_{exec} \epsilon \mathbb{N}^{n_i} \{ \sum N_{exec_{Sj}} = \sum P_{Ci-s_j} * N_{Ci-s_j} * N_{Ci} \}$$

in cui

$$N_{C_i} \epsilon \mathbb{N}^{i} \{ N_{C_i} = \sum N_{exec_{C_i si}} \}$$

Questo significa che il numero di volte con cui un generico servizio s_j va in esecuzione è rappresentato dalla somma delle volte in cui i componenti C_i hanno la necessità di richiamare il servizio s_j e lo fanno con una probabilità P_{Ci} per un numero di volte pari a N_{Ci-s_j} e per il numero di volte N_{Ci} che rappresenta le volte in cui ogni singolo componente C_i è in esecuzione.

Viene presentato un esempio per spiegare la metrica. Supponiamo di avere questa architettura:

- C_{11} con $availability_{C_{11}} = 0,93$, $cost_{C_{11}} = 2$, $execution\_time_{C_{11}} = 1$,
  - fornisce il servizio s_1
  - richiede il servizio s_2 con $P_{C_{11}-S_2}$= 0,8 e $N_{C_{11}-S_2}$= 1
  - richiede il servizio s_3 con $P_{C_{11}-S_3}$= 1 e $N_{C_{11}-S_3}$= 2
- C_{12} con $availability_{C_{12}} = 0,85$, $cost_{C_{12}} = 8$, $execution\_time_{C_{12}} = 3$,
  - fornisce il servizio s_1
  - fornisce il servizio s_2
  - richiede il servizio s_3 con $P_{C_{12}-S_3}$= 0,5 e $N_{C_{12}-S_3}$= 2
  - richiede il servizio s_4 con $P_{C_{12}-S_4}$= 0,9 e $N_{C_{12}-S_4}$= 3
- C_{31} con $availability_{C_{31}} = 0,9$, $cost_{C_{31}} = 4$, $execution\_time_{C_{31}} = 1,5$,
  - fornisce il servizio s_3
- C_{41} con $availability_{C_{41}} = 0{,}92$ $cost_{C_{41}} = 4$, $execution\_time_{C_{41}} = 2$,
  - fornisce il servizio s_4

Calcoliamo ora i valori di N_{exec}:

- si parte con il primo componente con cui si avvia l'applicazione che è C_{11}
  - in questo caso lui fornisce s_1 , nessuno lo richiama quindi si assume che $N_{exec_{s1}} = 1$

- fatta questa assunzione allora anche il componente C_{11} avrà un $N_{c_1} = 1$

si calcola ora il numero di volte che vanno in esecuzione i servizi richiamati dal componente

- $N_{exec_{s2}}$= $P_{C_{11}-s_2} * N_{C_{11}-s_2} * N_{C_{11}}$ = 0,8*1*1 = 0,8
- $N_{exec_{s3}}$= $P_{C_{11}-s_3} * N_{C_{11}-s_3} * N_{C_{11}}$ = 1*2*1 = 2

A questo punto si analizzano gli altri componenti dell'architettura:

- il componente C_{12} fornisce il servizio s_2 quindi in base a quanto stabilito esso andrà in esecuzione $N_{c_{12}} = 0,8$ perchè dipende dalle volte che va in esecuzione il servizio che fornisce.

  si calcola ora il numero di volte che vanno in esecuzione i servizi richiamati dal componente

  - $N_{exec_{s3}}$= $P_{C_{12}-s_3} * N_{C_{12}-s_3} * N_{C_{12}}$=0,5*2*0,8 = 0,8
  - $N_{exec_{s4}}$= $P_{C_{12}-s_4} * N_{C_{12}-s_4} * N_{C_{12}}$=0,9*3*0,8 = 2,16

- il componente C_{31} fornisce il servizio s_3 quindi in base a quanto stabilito esso andrà in esecuzione $N_{c_3} = 2 + 0,8$. Esso non richiede altri servizi.

- il componente C_{41} fornisce il servizio s_4 quindi in base a quanto stabilito esso andrà in esecuzione $N_{c_4} = 2,16$ essendo richiamato dal componente C_{12}. Esso non richiede altri servizi.

Riassumendo si ha: $N_{exec_{s1}} = 1$, $N_{exec_{s2}}$= 0,8 , $N_{exec_{s3}}$= 2,8 , $N_{exec_{s4}}$= 2,16.

La metrica mette in relazione il numero di volte che un servizio va in esecuzione con il numero di volte che il componente che lo fornisce vada in esecuzione.

Nell'esempio il componente C_{12} fornisce più servizi nello specifico fornisce il servizio s_1 e s_2 ma viene ritenuto responsabile solo per il servizio s_2 in quanto il servizio s_1 è fornito dal componente C_{11} che entra in esecuzione prima del componente C_{12} .

In generale la metrica si adatta a studiare il contributo di tutti i componenti C_j che forniscono più di un servizio ma solo se non già fornito da componenti entrati in esecuzione prima di C_j. In caso il componente C_j fornisse più di un servizio ancora non richiamato allora si avrebbe $N_{c_j} = \sum N_{exec_{C_j sj}}$ ovvero la somma di tutti i valori $N_{exec_{C_j sj}}$ dei servizi che fornisce il componente C_j e non forniti da altri componenti già in esecuzione.

La metrica appena descritta risulta importante per lo studio per capire con quale probabilità il componente che eroga il servizio può fallire. Questo viene studiato in due modi:

- si suppone che il componente fallisca alla chiamata del servizio che deve erogare (caso 1)

- si suppone che il componente fallisca casualmente durante l'esecuzione dell'applicazione (caso 2)

I due casi vengono ora trattati separatamente.

2.2.2.1 Caso uno: il componente fallisce alla chiamata del servizio

Per questo caso è stata definita la metrica:

- ***Probability to be running.*** Questa metrica calcola la probabilità di trovare un servizio in esecuzione a seguito di una chiamata e di riflesso la probabilità di trovare in esecuzione il componente che lo fornisce sempre all'interno dell'architettura considerata.

$$PTR_S \epsilon \mathbb{N}^{n_|} \{PTR_{S_i} = \frac{N_{exec_{S_i}}}{\sum N_{exec}}\}$$

$$PTR_C \epsilon \mathbb{N}^{n_|} \{PTR_{C_i} = \sum PTR_{C_i S_i}\}$$

Riprendendo l'esempio precedente si ha:

- $N_{exec_{s1}} = 1$, $N_{exec_{s2}} = 0{,}8$, $N_{exec_{s3}} = 2{,}8$, $N_{exec_{s4}} = 2{,}16$.
- $\sum N_{exec} = 1{+}0{,}8{+}2{,}8{+}2{,}16 = 6{,}76$

Si ottiene che:

- $PTR_{S_1} = 1/6,76 = 0{,}1479$
- $PTR_{S_2} = 0,8/6,76 = 0{,}1183$
- $PTR_{S_3} = 2,8/6,76 = 0{,}4142$
- $PTR_{S_4} = 2,16/6,76 = 0{,}3195$

La metrica fornisce una informazione utile ai fini dello studio ovvero la probabilità che il componente che fornisce il servizio sia in esecuzione. Secondo il caso uno questo indica anche la probabilità che il componente si rompa nel momento stesso in cui il servizio che fornisce viene invocato. È un interessante metro di valutazione per decidere se sostituire il componente a fronte di una probabilità troppo alta di rottura causata dalla chiamata del suo servizio.

Nell'esempio fatto si è spiegato come vengono calcolati i valori della metrica PTR per i servizi facendo attenzione al caso del componente C_{12} che fornisce due servizi ma risulta resposabile solo per il secondo servizio che fornisce. Gli altri componenti forniscono un solo servizio quindi in questo caso il calcolo del valore della metrica PTR relativa ai componenti risulta essere uguale al valore PTR del servizio che forniscono:

- $PTR_{C_{11}}$ = il componente fornisce solo il servizio s_1 quindi sarà uguale a PTR_{S_1}= 0,1479
- $PTR_{C_{12}}$ = il componente fornisce sia il servizio s_1 che s_2 ma è responsabile solo per s_2 quindi $PTR_{C_{12}} = PTR_{S_2} = 0,1183$
- $PTR_{C_{31}}$ = il componente fornisce il servizio s_3 quindi $PTR_{C_{31}} = PTR_{S_3} = 0,4142$
- $PTR_{C_{41}}$ = il componente fornisce il servizio s_4 quindi $PTR_{C_{41}} = PTR_{S_4} = 0,3195$

Supponiamo ora invece che il componente C_{12} faccia parte di un'altra architettura in cui è responsabile di entrambi i servizi che fornisce, in questo caso si avrà:

- $PTR_{C_{12}} = PTR_{S_1} + PTR_{S_2} = 0,2662$.

2.2.2.2 Caso due: il componente si rompe in maniera casuale

In questo secondo caso non si può prescindere dalla durata di esecuzione dei componenti che forniscono i servizi. Diventa necessario calcolare la percentuale di tempo totale di permanenza in esecuzione del componente all'interno dell'architettura. Esso dipende da quanti servizi fornisce il componente e da quante volte vengono richiamati. Diventa necessario introdurre la durata di esecuzione di un componente perchè stiamo studiando il caso in cui esso possa rompersi in qualsiasi momento: prima durante o dopo una chiamata ad un servizio. Per questo motivo si introduce la metrica:

- ***Weight of residence time in system.*** Questa metrica calcola il tempo totale di esecuzione di un componente all'interno dell'architettura considerata e permette di conoscere quale componente permane in esecuzione per un tempo maggiore quindi il peso che assume rispetto al tempo totale di esecuzione dell'architettura. In questo modo si evidenzia quale componente ha maggiori probabilità di essere in esecuzione in caso di rottura.

$$T_{Ci} \epsilon \mathbb{N} \{ T_{Ci} = \sum N_{exec_{C_i - S_j} Ci - s_j} * T_{Ci} \}$$

$$WRT \epsilon \mathbb{N}^{n} \{ WRT_{C_i} = \frac{T_{Ci}}{\sum T} \}$$

in cui T_{Ci} è il tempo totale con cui il componente $C_i$rimane in esecuzione ed equivale al numero di volte che vengono richiamati i servizi che fornisce per il tempo intrinseco del componente per espletare la richiesta (tempo indicato con T_{Ci}).

La metrica WRT invece calcola il peso del tempo di esecuzione totale di ogni componente rispetto al tempo totale di esecuzione dell'applicazione considerata.

Supponiamo di avere questa architettura:

- C_{11} con $availability_{C_{11}} = 0,93$, $cost_{C_{11}} = 2$, $execution\_time_{C_{11}} = 1$,
  - fornisce il servizio s_1
  - richiede il servizio s_2 con $P_{C_{11}-S_2}$= 0,8 e $N_{C_{11}-S_2}$= 1
  - richiede il servizio s_3 con $P_{C_{11}-S_3}$= 1 e $N_{C_{11}-S_3}$= 2
- C_{12} con $availability_{C_{12}} = 0,85$, $cost_{C_{12}} = 8$, $execution\_time_{C_{12}} = 3$,
  - fornisce il servizio s_1
  - fornisce il servizio s_2
  - richiede il servizio s_3 con $P_{C_{12}-S_3}$= 0,5 e $N_{C_{12}-S_3}$= 2
  - richiede il servizio s_4 con $P_{C_{12}-S_4}$= 0,9 e $N_{C_{12}-S_4}$= 3
- C_{31} con $availability_{C_{31}} = 0,9$, $cost_{C_{31}} = 4$, $execution\_time_{C_{31}} = 1,5$,
  - fornisce il servizio s_3
- C_{41} con $availability_{C_{41}} = 0{,}92$ $cost_{C_{41}} = 4$, $execution\_time_{C_{41}} = 2$,
  - fornisce il servizio s_4

e supponiamo di aver già calcolato i valori di:
$N_{exec_{C_{11}}} = 1$, $N_{exec_{C_{12}}}$= 0,8 , $N_{exec_{C_{31}}}$= 2,8 , $N_{exec_{C_{41}}}$= 2,16.
Calcoliamo ora i valori di WRT:

- $T_{C_{11}}$ =$N_{exec_{C_{11}}} * execution\_time_{C_{11}}$= 1*1 = 1
- $T_{C_{12}}$ =$N_{exec_{C_{12}}} * execution\_time_{C_{12}}$= 0,8*3 = 2,4
- $T_{C_{31}}$ =$N_{exec_{C_{31}}} * execution\_time_{C_{31}}$= 2,8*1,5 = 4,2
- $T_{C_{41}}$ =$N_{exec_{C_{41}}} * execution\_time_{C_{41}}$= 2,16*2 = 4,32

Si ottiene che: $\sum T$ = 11,92

- $WRT_{C_{11}} = 1/11,92 = 0{,}0839$
- $WRT_{C_{12}} = 2,4/11,92 = 0{,}2013$
- $WRT_{C_{31}} = 4,2/11,92 = 0{,}3523$
- $WRT_{C_{41}} = 4,32/11,92 = 0{,}3624$

In questo modo si identifica nel componente C_{41} il componente maggiormente in esecuzione nell'architettura.

Finora si è studiata la probabilità che un servizio andasse in esecuzione e la relativa probabilità che il componente che lo fornisce si rompesse all'atto della chiamata del servizio o in maniera casuale in base al suo tempo totale di permanenza

in esecuzione nell'architettura. Metriche utili a mio avviso, dal punto di vista del progettista per poter decidere se investire in un componente con maggior garanzie di funzionamento o meno.

L'assunzione che si è fatta finora è che ogni componente dell'architettura venisse usato ogni volta che l'applicazione entra in esecuzione. Di fatto questo non è del tutto vero. Consideriamo il caso di studio trattato nel capitolo uno: l'applicazione web che registra il piano di studio di uno studente presso la sua università.

Se l'utente rispetta i vincoli dell'università allora gli viene permesso il pagamento e l'applicazione viene utilizzata nella sua interezza ma, se la logica applicativa stabilisce che qualche vincolo nel regolamento dell'università non viene rispettato dal piano degli esami inseriti, rimanda all'utente una notifica di errore e non viene permesso il pagamento.

In questa fase l'applicazione non attiva il servizio di pagamento online presso la banca.

Per questo motivo è importante considerare un altro aspetto ovvero la vera dinamica con cui si svolge l'applicazione. Di fatto si passa ad analizzare il workflow di funzionamento dell'architettura. Grazie ad esso è possibile identificare quali attività vengono messe in atto.

Le attività sono le interazioni tra il componente che entra in esecuzione e richiede un servizio e il componente che fornisce tale servizio.

Supponiamo che l'utente sia ostinato nel voler inserire un esame che viola le regole dell'università. Tenterà n volte e verrà respinto n volte dalla logica applicativa. Si avranno:

- l'attività «aggiunta dell'esame» che richiede la presenza della logica di presentazione che richiama la logica applicativa
- l'attività seguente è «analisi dei vincoli universitari» che viene svolta dalla logica applicativa che richiama i propri metodi
- infine avremo «notifica di errore allo studente» con cui la logica applicativa richiama la logica di presentazione.

Questo mini estratto di workflow verrà svolto per tutte le n volte con cui l'utente tenterà di aggiungere l'esame al suo piano di studi.

Da qui l'idea di studiare una metrica che tenga conto di quante volte realmente un compoente entra in esecuzione.

- ***In action.*** Questa metrica calcola quante volte il servizio entra realmente in esecuzione all'interno di tutti i path alternativi di esecuzione dell'applicazione. In questo modo si riesce a determinare quante volte il componente che fruisce i servizi entra realmente in esecuzione.

$$IA_S \epsilon \mathbb{N}^n | \{IA = 1 - (1 - P_{path_i} * u_{Si-act_i})^{m_{path_i}}\}$$

$$IA_C \epsilon \mathbb{N}^n | \{IA_{C_i} = \prod IA_{C_i S_i}\}$$

- Path di esecuzione 1:
  - l'attività A_1 = «aggiunta dell'esame» che richiede la presenza della logica di presentazione che richiama la logica applicativa
  - l'attività seguente A_2 = è «analisi dei vincoli universitari» che viene svolta dalla logica applicativa che richiama i propri metodi
  - infine avremo A_3 = «notifica di errore allo studente» con cui la logica applicativa richiama la logica di presentazione.
- Path di esecuzione 2:
  - l'attività A_1 = «aggiunta dell'esame» che richiede la presenza della logica di presentazione che richiama la logica applicativa
  - l'attività seguente A_2 = è «analisi dei vincoli universitari» che viene svolta dalla logica applicativa che richiama i propri metodi
  - A_4 = la logica applicativa manda una «notifica di procedi con il pagamento all'utente» e richiama la logica di presentazione
  - l'attività successiva è A_5 = «l'utente effettua il pagamento», la logica di presentazione richiama il servizio di pagamento.
  - A_6 = «pagamento andato a buon fine» il servizio di pagamento invia una notifica alla logica di presentazione per avvisare l'utente.
  - A_7 = «notifica via email del pagamento» il servizio di pagamento invia una mail all'utente.

Innanzitutto si hanno due path alternativi di esecuzione, si suppone che abbiano la stessa probabilità di esecuzione quindi si calcola:

- $P_{path_1} = P_{path_2} = 1/2$=0,5

Assumiamo ora che:

- la logica di presentazione è il componente C_{11} che fornisce il servizio s_1 con cui l'utente si collega all'applicazione
- la logica applicativa è il componente C_{12} che fornisce il servizio $s_2$con cui la logica applicativa analizza i vincoli
- la banca è il componente C_{31} che fornisce il servizio di pagamento s_3

- il gestore di posta è il componente C_{41} che fornisce il servizio di email s_4

Facciamo la supposizione che ogni componente fornisca un solo servizio e per tanto si passa a scomporre direttamente le attività in base al componente che fornisce il servizio non essendoci ambiguità nella fruizione del servizio stesso.

Quindi scomponiamo le attività in base ai componenti utilizzati:

- l'attività A_1 richiede C_{11} e C_{12} con $u_{C_{11}-act_1} = 1$ e $u_{C_{12}-act_1} = 1$
- l'attività A_2 richiede C_{12} con $u_{C_{12}-act_2} = 2$. si suppone infatti che la logica invochi i propri metodi per valutare il vincolo di inserimento degli esami e poi il risultato venga rimandato alla logica per un totale di 2 utilizzi.
- l'attività A_3 richiede C_{11} e C_{12} con $u_{C_{11}-act_3} = 1$ e $u_{C_{12}-act_3} = 1$
- l'attività A_4 richiede C_{11} e C_{12} con $u_{C_{11}-act_4} = 1$ e $u_{C_{12}-act_4} = 1$
- l'attività A_5 richiede C_{11} e C_{31} con $u_{C_{11}-act_5} = 1$ e $u_{C_{31}-act_5} = 1$
- l'attività A_6 richiede C_{11} e C_{31} con $u_{C_{11}-act_6} = 1$ e $u_{C_{31}-act_6} = 1$
- l'attività A_7 richiede C_{41} e C_{31} con $u_{C_{41}-act_7} = 1$ e $u_{C_{31}-act_7} = 1$

Ora calcoliamo il valore della metrica IA per ogni componente:

- Per il componente C_{11} si ha: $IA_{C_{11}} = 1 - [(1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1] = 0,9921$
- Per il componente C_{12} si ha: $IA_{C_{12}} = 1 - [(1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1] = 0,9921$
- Per il componente C_{31} si ha: $IA_{C_{31}} = 1 - [(1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1 * (1 - 0,5 * 1)^1] = 0,875$
- Per il componente C_{41} si ha: $IA_{C_{41}} = 1 - [(1 - 0,5 * 1)^1] = 0,5$

In questo modo si ottiene che il componente maggiormente in esecuzione, dopo aver analizzato i path alternativi, è il componente C_i .

Questo dovrebbe guidare il progettista di sistema alla scelta di un componente di tipo C_i che abbia una elevata availability, sempre rispettando l'ipotesi fatta ovvero che una maggiore availability dia maggiori garanzie di funzionamento, in quanto questo componente è quello che realmente viene messo in esecuzione più volte nell'applicazione e senza il quale si avrebbe un degrado sensibile delle prestazioni del sistema o addirittura una interruzione del servizio.

La sezione relativa allo studio dei servizi forniti nell'architettura si è conclusa. L'intento di queste metriche è fornire un altro punto di vista al progettista per effettuare una scelta dei componenti in maniera più accurata. Come detto a inizio

capitolo tutto lo studio si basa su delle ipotesi fatte perchè di fatto il campo dei sistemi auto - adattativi non forniscono ancora metriche certe di analisi. Questa sezione tenta un primo approccio all'analisi dinamica di tali sistemi sperando di fornire un valido supporto alla scelta dei componenti.

2.2.3 Analisi di adattabilità

Dopo aver analizzato lo spazio delle possibili architetture e dopo aver analizzato in dettaglio una generica architettura rispetto ai servizi forniti in essa, in questa sezione verranno presentate le metriche di adattabilità dell'architettura e dei servizi attualmente in uso.

Nello specifico si suppone di avere a disposizione una generica architettura A_i e si nota un degrado delle prestazioni tutte le volte che viene invocato il servizio s_3. A questo punto il sistema essendo auto -adattativo cercherà di mantenere il proprio livello di prestazioni e dovrà selezionare i componenti più adeguati a tale scopo.

Di seguito sono quindi riportare quattro metriche che seguono il filo logico di questo discorso. Viene effettuata una distinzione tra adattabilità dell'architettura e adattabilità del servizio considerato.

- ***Architecture dynamic adaptation w.r.t. availability.*** Questa metrica identifica la cardinalità dell'insieme di architetture che soddisfano il requisito di availability e che si differenziano dall'architettura attualmente usata solo per il componente rotto.

$$ADAA \epsilon \mathbb{N} \{ ADAA = \sum A_i | GAS_{A_i} \geq 1 \wedge \forall C_i | \{ \{ (C_i \neq C_r \wedge C_i \in A_a) \implies (C_i \in A_i) \} \wedge \{ (C_i = C_r \wedge C_i \in A_a) \implies (C_i \notin A_i) \} \} \}$$

- ***Architecture dynamic adaptation w.r.t. cost.*** Questa metrica identifica la cardinalità dell'insieme di architetture che soddisfano il requisito di costo e che si differenziano dall'architettura attualmente usata solo per il componente rotto.

$$ADAC \epsilon \mathbb{N} \{ ADAC = \sum A_i | GCS_{A_i} \geq 1 \wedge \forall C_i | \{ \{ (C_i \neq C_r \wedge C_i \in A_a) \implies (C_i \in A_i) \} \wedge \{ (C_i = C_r \wedge C_i \in A_a) \implies (C_i \notin A_i) \} \} \}$$

Consideriamo l'esempio di queste due architetture:

- architettura attualmente in uso A_a formata dai componenti:
  - C_{11} con $availability_{C_{11}} = 0,93$, $cost_{C_{11}} = 2$, $execution\_time_{C_{11}} = 1$,
    - fornisce il servizio s_1
    - richiede il servizio s_2 con $P_{C_{11}-S_2}$= 0,8 e $N_{C_{11}-S_2}$= 1
    - richiede il servizio s_3 con $P_{C_{11}-S_3}$= 1 e $N_{C_{11}-S_3}$= 2
  - C_{12} con $availability_{C_{12}} = 0,85$, $cost_{C_{12}} = 8$, $execution\_time_{C_{12}} = 3$,
    - fornisce il servizio s_1

    - fornisce il servizio s_2
    - richiede il servizio s_3 con $P_{C_{12}-S_3}$= 0,5 e $N_{C_{12}-S_3}$= 2
    - richiede il servizio s_4 con $P_{C_{12}-S_4}$= 0,9 e $N_{C_{12}-S_4}$= 3
  - C_{31} con $availability_{C_{31}} = 0,9$, $cost_{C_{31}} = 4$, $execution\_time_{C_{31}} = 1,5$,
    - fornisce il servizio s_3
  - C_{41} con $availability_{C_{41}} = 0{,}92$ $cost_{C_{41}} = 4$, $execution\_time_{C_{41}} = 2$,
    - fornisce il servizio s_4

- architettura potenzialmente adattabile A_1 formata dai componenti:
  - C_{11} con $availability_{C_{11}} = 0,93$, $cost_{C_{11}} = 2$, $execution\_time_{C_{11}} = 1$,
    - fornisce il servizio s_1
    - richiede il servizio s_2 con $P_{C_{11}-S_2}$= 0,8 e $N_{C_{11}-S_2}$= 1
    - richiede il servizio s_3 con $P_{C_{11}-S_3}$= 1 e $N_{C_{11}-S_3}$= 2
  - C_{12} con $availability_{C_{12}} = 0,85$, $cost_{C_{12}} = 8$, $execution\_time_{C_{12}} = 3$,
    - fornisce il servizio s_1
    - fornisce il servizio s_2
    - richiede il servizio s_3 con $P_{C_{12}-S_3}$= 0,5 e $N_{C_{12}-S_3}$= 2
    - richiede il servizio s_4 con $P_{C_{12}-S_4}$= 0,9 e $N_{C_{12}-S_4}$= 3
  - C_{32} con $availability_{C_{32}} = 0,95$, $cost_{C_{32}} = 7$, $execution\_time_{C_{32}} = 2$,
    - fornisce il servizio s_3
  - C_{41} con $availability_{C_{41}} = 0{,}92$ $cost_{C_{41}} = 4$, $execution\_time_{C_{41}} = 2$,
    - fornisce il servizio s_4

Le architetture hanno i seguenti valori:

- $architettura_1$: $GAS_{architettura_1}$= 1,1905 e $GCS_{architettura_1}$= 33,75
- $architettura_2$: $GAS_{architettura_2}$= 2,3810 e $GCS_{architettura_2}$= 30,5357

Calcoliamo i valori assunti dalle metriche ADAA e ADAC:

- ADAA considera solo le architetture che si differenziano da quella attualmente in uso solo per il componente da sostituire. In questo caso vogliamo sostituire il componente che fornisce il servizio s_3. Le architetture A_a ed A_1 sono differenti solo per il componente, rispettivamente, C_{31} e C_{32} che forniscono lo stesso servizio. Questo permette alla metrica ADAA di selezionare l'architettura A_1. La metrica seleziona tutte le architetture possibili che si differenziano dall'architettura in uso solo per il componente da sostituire. L'esempio è semplificato ma nel capitolo tre verrà presentato un caso di studio più complesso e si potrà

apprezzare meglio questa metrica. Di fondo resta che questa metrica applica il vincolo di differenziazione tra architetture solo per un componente per effettuare una scrematura dello spazio delle possibili architetture. La metrica ADAA prevede che si rispetti anche un secondo vincolo ovvero che il valore di GAS dell'architettura sia maggiore o uguale a 1 il che significa che ogni suo componente fornisce delle garanzie di funzionamento elevate. L'architettura A_1 risulta idonea a tutti gli effetti perchè rispetta anche questo vincolo e quindi e entra a far parte di diritto dell'insieme di tutte e sole le architetture potenzialmente adattabili rispetto a quella in uso secondo il vincolo di availability. In questo esempio è l'unica architettura adattabile.

- con lo stesso approccio si calcola ADAC considerando solo le architetture che si differenziano da quella attualmente in uso solo per il componente da sostituire. In questo caso vogliamo sostituire il componente che fornisce il servizio s_3. Le architetture A_a ed A_1 come già esposto si differenziano solo per il componente che fornisce il servizio richiesto quindi anche ADAC seleziona l'architettura A_1 . A questo punto si applica il secondo vincolo ovvero che il valore di GCS dell'architettura sia maggiore o uguale a 1 il che significa che ogni suo componente avrà un costo inferiore al budget massimo possibile. Ora sia affermare che l'architettura A_1 risulta idonea anche rispetto al vincolo di costo.

La cardinalità dell'insieme ADAA = 1, la cardinalità dell'insieme ADAC = 1.

Si mantengono i due requisiti separati sempre in virtù dell'ipotesi iniziale di aver componenti con maggiore availability a fronte di costi elevati.

Dopo aver selezionato le architetture che sono potenzialmente adattabili a quella in uso si procede analizzando quali componenti singoli sono considerati idonei a fruire il servizio.

- ***Service adaptation w.r.t. availability.*** Questa metrica calcola la somma dei componenti non usati nell'architettura ma attualmente disponibili che soddisfano il requisito di availability e possono pertanto essere usati per fornire il servizio.

$$SAA \epsilon \mathbb{N} \{ SAA = \sum C_{nu-s_r} \nmid \{ (C_{nu-s_r} \neq C_r) \wedge (C_{nu-s_r} \notin A_a) \wedge (C_r \in A_a) \wedge (BSA_{C_{nu-s_r}} = 1) \} \}$$

- ***Service adaptation w.r.t. cost.*** Questa metrica calcola la somma dei componenti non usati che soddisfano il requisito di costo e possono pertanto essere usati per fornire il servizio.

$$SAC \epsilon \mathbb{N} \{ SAC = \sum C_{nu-s_r} \nmid \{ (C_{nu-s_r} \neq C_r) \wedge (C_{nu-s_r} \notin A_a) \wedge (C_r \in A_a) \wedge (BSC_{C_{nu-s_r}} = 1) \} \}$$

Prendiamo in considerazione i seguenti componenti:

- componente C_{31} fornisce il servizio s_3 con $FRA_{C_{31}}$= 1,0 ==> $BSA_{C_{31}} = 1$ e $FRC_{C_{31}}$= 7,5 ==> $BSC_{C_{31}} = 1$

- componente C_{32} fornisce il servizio s_3 con $FRA_{C_{32}}$= 2,0 ==> $BSA_{C_{32}} = 1$ e $FRC_{C_{32}}$= 4,2857 ==> $BSC_{C_{32}} = 1$

Calcoliamo i valori assunti dalle metriche SAA e SAC:

supponiamo di avere attualmente in esecuzione il componente C_{31} per il servizio s_3. Si ottiene:

- Studiando la metrica SAA
  - il componente C_{32} non è in uso nell'architettura e fornisce lo stesso servizio richiesto quindi risulta idoneo inoltre ha un valore di BSA = 1 quindi viene selezionato
  - valore finale di SAA = 1
- Studiando la metrica SAC
  - il componente C_{32} non è in uso nell'architettura e fornisce lo stesso servizio richiesto quindi risulta idoneo inoltre ha un valore di BSC = 1 quindi viene selezionato
  - valore finale di SAC = 1

Come si evince le metriche SAA e SAC permettono di effettuare una scrematura dei componenti rispetto al servizio che forniscono. Sono metriche utili per evitare di effettuare una analisi a tutto campo con notevole risparmio computazionale.

2.2.4 Analisi finale sintetica

Dopo aver trattato in modo separato i componenti, le architetture e i servizi ora si presenta una analisi riepilogativa che raggruppa i concetti esposti.

Per le architetture si definisce una funzione da massimizzare ovvero essa permette al progettista di individuare quali architetture tra quelle risultate idonee all'adattabilità danno un miglior valore di adattamento legato al raggiungimento di un buon compromesso tra buone garanzie di funzionamento (requisito di availability soddisfatto) e risparmio sul budget (requisito di costo soddisfatto). Per il calcolo di questa metrica si lascia al progettista la possibilità di inserire i pesi con cui valutare i propri obiettivi. Verrà assegnato un peso maggiore al requisito di availability se si è interessati a fornire l'architettura con le migliori garanzie di prestazioni e funzionamento viceversa, si darà un peso maggiore al vincolo sul costo se si ha un budget limitato.

- ***Objective function.*** Questa funzione va massimizzata e definisce una media pesata degli interessi che si vogliono raggiungere.

$$max\,(OF)\,\epsilon\mathbb{N}\{OF = W_1 * GAS + W_2 * GCS\}$$

Per i componenti che forniscono i servizi si definisce:

- ***Relevance constraint.*** Questa funzione indica la rilevanza di un componente all'interno dell'architettura. Questa metrica è il riassunto di tutto il mio studio e vuole evidenziare nuovamente il componente su cui porre l'attenzione per due aspetti importanti a mio avviso:
  - il primo aspetto è riassunto nella metrica PTR ovvero la probabilità che il componente si rompa quando viene invocato il servizio che fornisce.
  - il secondo aspetto riguarda il reale utilizzo che si fa del componente ovvero il valore di RA nonchè le volte che il componente viene impiegato nel sistema durante tutta la sua esecuzione.

Con questa metrica si identifica quale componente può maggiormente inficiare le prestazioni di tutto il sistema. Infatti se esso ha maggiori probabilità di fallire alla chiamata causera una perdita di prestazioni con maggior probabilità rispetto ad altri componenti degradando le prestazioni totali. Motivo che dovrebbe consigliare al progettista di scegliere un componenti con maggiori garanzie di funzionamento. L'altro aspetto riguarda le volte con cui questo componente viene eseguito nel sistema. Un componente eseguito molte volte necessità di una availability elevata o le prestazioni degraderanno ad ogni sua invocazione. Per questo motivo si ha:

$$RC \epsilon \mathbb{N}^n | \{RC_{Ci} = W_3 * IA_{Ci} + W_4 * PTR_{Ci}\}$$

Anche in questa metrica si lascia al progettista la possibilità di scegliere quale peso attribuire ad entrambi gli aspetti presi in considerazione.

Prendiamo in considerazione la seguente architettura:

- $architettura_1$
  - costituita dai componenti C_{11}, C_{12}, C_{31} e C_{41}
  - che forniscono rispettivamente i servizi s_1 , s_2 , s_3 e s_4.
- $architettura_1$: $GAS_{architettura_1}$= 1,1905 e $GCS_{architettura_1}$= 33,75
- $PTR_{C_{11}}$ =PTR_{S_1}= 0,1479
- $PTR_{C_{12}}$ =PTR_{S_2} = 0,1183
- $PTR_{C_{31}}$ =PTR_{S_3} = 0,4142
- $PTR_{C_{41}}$ =PTR_{S_4} = 0,3195
- Per il componente C_{11} si ha: $IA_{C_{11}}$ = 0,9921
- Per il componente C_{12} si ha: $IA_{C_{12}}$ = 0,9921
- Per il componente C_{31} si ha: $IA_{C_{31}}$ = 0,875

- Per il componente C_{41} si ha: $IA_{C_{41}} = 0{,}5$

Supponiamo che i pesi attribuiti siano tutti pari a 0,5.

I valori delle due metriche di analisi riassuntive sono:

- OF = 0,5*1,1905 + 0,5*33,75 = 17,4702
- $RC_{C_{11}}$ =0,5*0,1479 + 0,5*0,9921 = 0,57
- $RC_{C_{12}}$ =0,5*0,1183 + 0,5*0,9844 = 0,5514
- $RC_{C_{31}}$ =0,5*0,4142 + 0,5*0,875 = 0,6446
- $RC_{C_{41}}$ =0,5*0,3195 + 0,5*0,5 = 0,4098

Da questa metrica riassuntiva si evidenzia che il componente C_{31} è il componente che potrebbe inficiare maggiormente sulle prestazioni del sistema.

2.3 Conclusioni

Lo studio vuole presentare lo spazio di alternative plausibili da cui estrarre quella che si ritiene migliore. La parte della selezione finale viene lasciata al progettista del software in stretta collaborazione con il proprietario del sistema, lo scopo dello studio rimane puramente informativo e non decisionale.

Lo studio verte a superare i limiti di SOLAR rispetto all'analisi di adattabilità dei sistemi auto - adattativi, si basa su ipotesi e pertanto richiede alcune assunzioni. Tali assunzioni possono essere analizzate e prese come spunto per lo sviluppo di ulteriori miglioramenti. Esse sono:

- si assume che per i componenti che forniscono più di un servizio venga preso in considerazione il loro contributo solo per servizi ancora non in esecuzione e non già forniti da componenti entrati in esecuzione prima. Uno sviluppo ulteriore potrebbe andare in questo senso.
- si assume che i componenti sostituibili possano essere solo uno alla volta per non aumentare di molto lo sforzo computazionale. Ulteriori sviluppi potrebbero approfondire questo aspetto.
- si assume che i servizi forniti dai componenti siano parte integrante di essi e ci si è limitati a studiare il comportamenti dei servizi richiesti ma si potrebbe analizzare anche questo aspetto.
- si è effettuato lo studio rispetto ai requisiti di availability e costo ma si potrebbero cercare altri requisiti interessanti. Un primo inizio di analisi delle tempistiche è stato presentato con la metrica WRT.

Ognuna di queste assunzioni può essere analizzata più nel dettaglio per cercare di rendere SOLAR uno strumento sempre più accurato.

Nel capitolo tre verrà presentato un caso di studio più articolato e completo così da mostrare tutte le metriche e poter apprezzare meglio soprattutto la parte delle metriche di adattabilità delle architetture in quanto su uno spazio più esteso di architetture si può notare meglio il significato delle metrice ADAA e ADAC.

Nel capitolo quattro verrà presentato il framework con cui si vuole implementare lo studio appena trattato.

Capitolo 3

Caso di studio: un'applicazione web

In questo capitolo si affronta lo studio del caso già trattato nel capitolo due relativo ad una applicazione web alla luce delle metriche di adattabilità di architetture e servizi esposte nel capitolo tre. Lo scopo è mostrare come l'analisi effettuata nel capitolo tre voglia fornire maggiori informazioni al progettista di sistema per una scelta più accurata dei componenti da inserire nell'architettura. Lo studio trova implementazione nel framework che estende SOLAR e da cui vengono estratte le schermate qui di seguito presentate. In questa sezione ci si concentra sull'analisi dei risultati e si rimanda al capitolo cinque per spiegare come il framework è stato implementato e per descrivere nel dettaglio ogni schermata proposta.

3.1 Introduzione

Lo studio delle metriche di adattabilità proposte nel capitolo tre è un tentativo di estensione delle metriche implementate in SOLAR presentate nel capitolo due. Per questo motivo si sceglie di trattare il medesimo caso di studio affrontato nel capitolo due e di estenderlo con le metriche ad hoc pensate per l'analisi dei servizi. Per semplicità di lettura verranno ora riproposte tutte le informazioni salienti del caso di studio per poi proseguire con l'analisi.

3.2 Presentazione dell'architettura e dei componenti

Il caso di studio analizza una applicazione web usata dagli studenti per registrarsi all'anno accademico in una università. Il sistema è composto da:

- uno livello di presentazione con una interfaccia grafica e meccanismi di interazione con lo studente

- uno livello per la logica applicativa che approva o respinge le richieste dello studente per i corsi da acquistare.

All'inizio lo studente inserisce le sue richieste nel sistema. Queste informazioni vengono mandate alla logica applicativa. Se la loro proposta soddisfa le regole dell'università allora la logica interagisce con i servizi web della banca per procedere al pagamento. Una volta terminato tutto il controllo si ritorna al livello di presentazione, che invia un messaggio e-mail allo studente con le informazioni utili.

Si assume che esistano due componenti per la logica applicativa, un componente per il livello di presentazione; un componente che implementi sia il livello di presentazione che la logica applicativa, due servizi per il pagamento e tre servizi per inviare le email: uno di loro è locale per l'Università e due sono forniti da terzi.

Si riassume in tabella [3.1] i servizi utilizzati e componenti indicati nel caso di studio.

Tabella 3.1: Riassunto dei servizi e componenti usati nel caso di studio. [23].

descrizione	servizio o componente
registrazione dello studente	s_1
soddisfacimento richiesta dello studente	s_2
inviare una mail	s_3
pagamento in banca	s_4
livello di presentazione	C11
livello di presentazione + logica applicativa	C12
logica applicativa 1	C21
logica applicativa 2	C22
fornitore del componente email di terze parti 1	C31
fornitore del componente email di terze parti 2	C32
fornitore del componente email locale	C33
servizio per il pagamento della banca 1	C41
servizio per il pagamento della banca 2	C42

Per poter proseguire nell'analisi sono necessari alcuni dati aggiuntivi che vengono mostrati in figura [3.1].

Per ogni componente vengono riportati costo e availability direttamente assegnati accanto al nome. In tabella invece si hanno:

- $P_{ij}^{s_k}$ che denota la probabilità con cui il componente C_{ij}richiede il servizio s_k.
- $N_{ij}^{s_k}$ che denota il numero di volte che viene richiesto il servizio s_k.
- L'availability di un componente viene misurata monitorandolo.
- Si assume che non fallisca mai la connessione tra componenti.

Per l'analisi introdotta nel capitolo tre si è deciso di introdurre un'altra informazione relativa ai componenti, essa è:

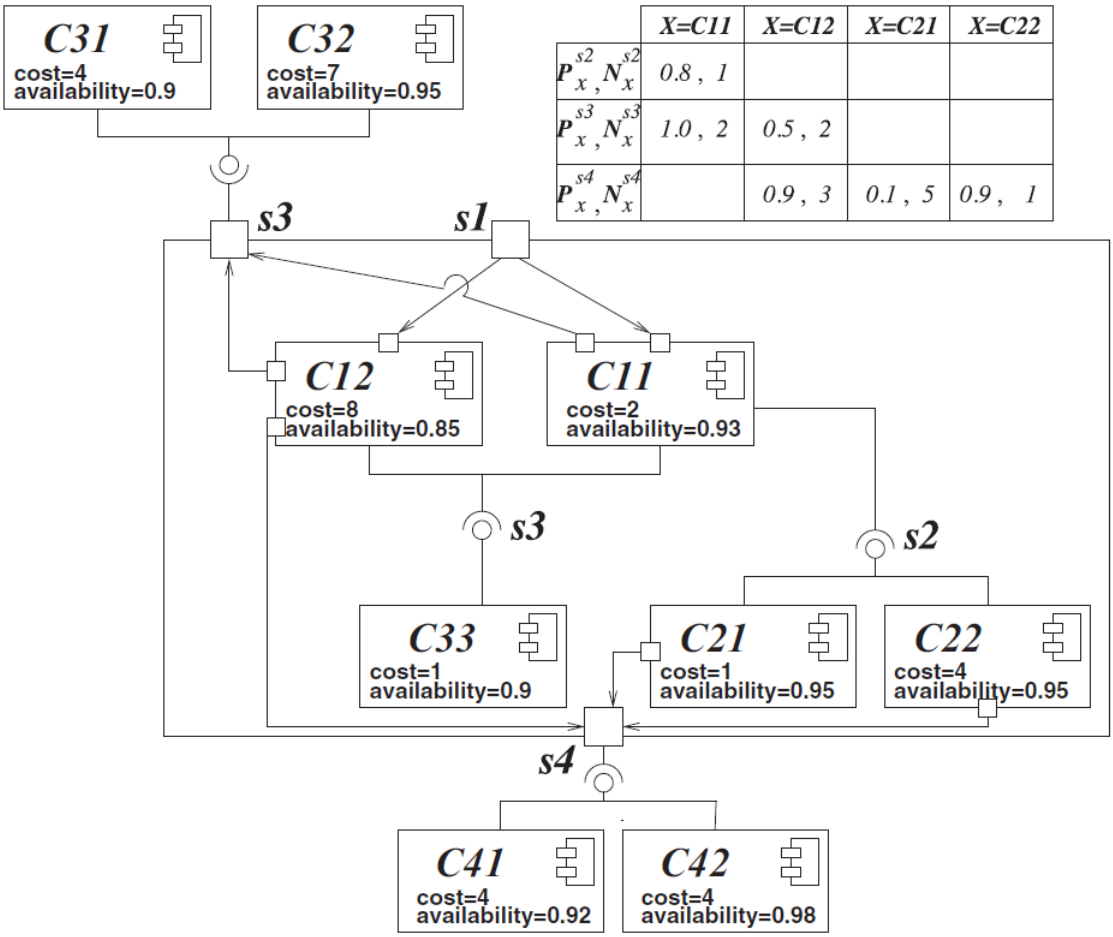

Figura 3.1: Informazioni aggiuntive per calcolare l'availability [23].

- $T_i$intesa come il tempo di esecuzione del componente $C_{i-esimo}$.

A questo punto si procede con l'analisi seguendo l'approccio presentato nel capitolo tre. Per semplicità di lettura dell'utente si riportano le formule relative ad ogni metrica introdotta rimandando al capitolo tre per la spiegazione di ognuna.

3.3 Calcolo delle metriche dello spazio iniziale dei componenti

Come prima cosa si analizzano tutti i componenti disponibili studiando:

- $(FRA)\, \epsilon \mathbb{N}^n | \{FRA_{Ci} = \frac{1-availability_{target}}{1-av_{Ci}}\}$
- $(BSA)\, \epsilon \{0,1\} | \{BSA_{Ci} = 1 \iff (FRA_{Ci} \geq 1) \vee BSA_{Ci} = 0 \iff (FRA_{Ci} < 1)\}$
- $(FRC)\, \epsilon \mathbb{N}^n | \{FRC_{Ci} = \frac{cost_{target}}{c_{ci}}\}$
- $(BSC)\, \epsilon \{0,1\} | \{BSC_{Ci} = 1 \iff (FRC_{Ci} \geq 1) \vee BSC_{Ci} = 0 \iff (FRC_{Ci} < 1)\}$.

In figura [3.2] (parte in alto) vengono mostrati i risultati per tutti i componenti disponibili. Tutte le immagini riportate in questa sezione sono state realizzate tramite l'utilizzo del framework AdaptAnalyzer.

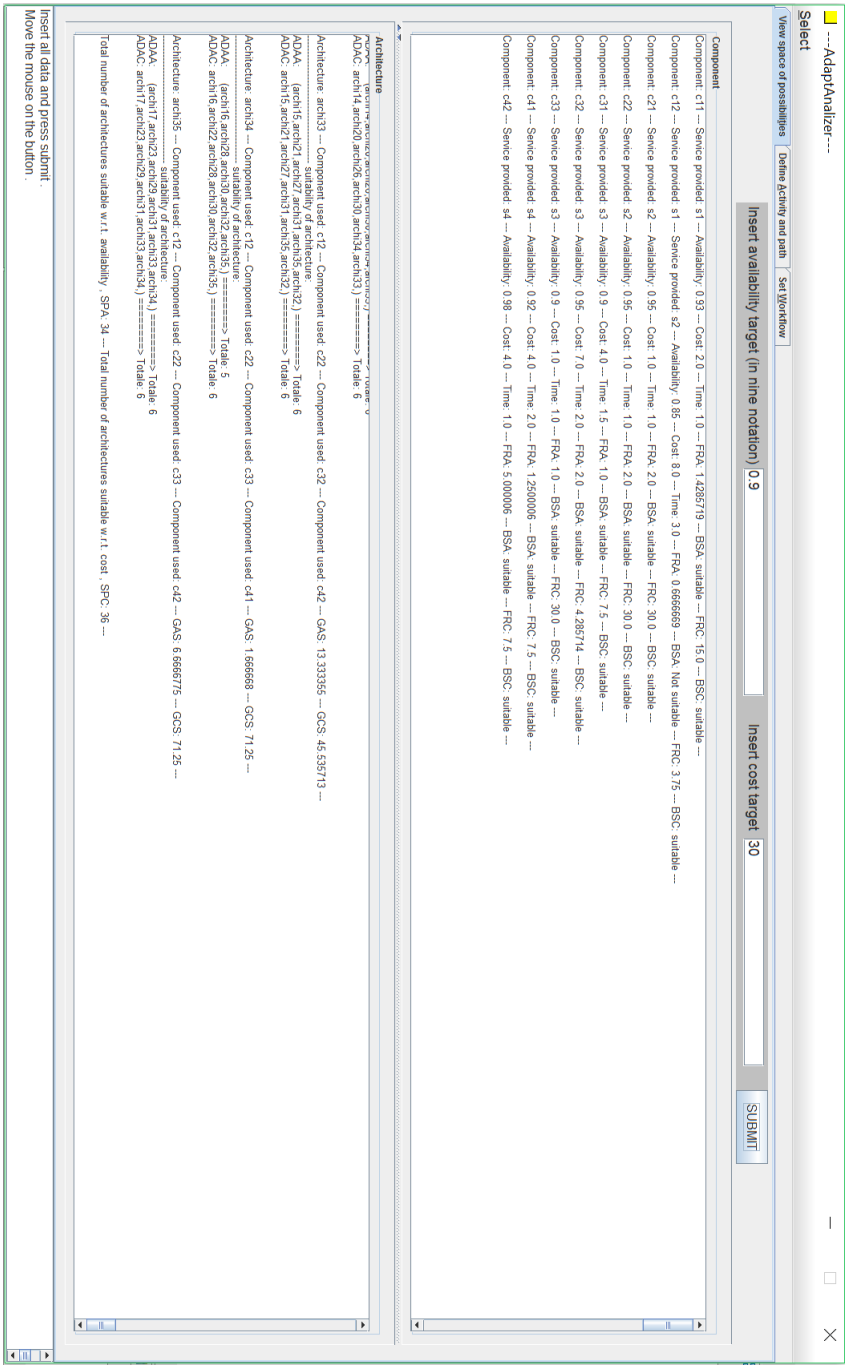

Figura 3.2:
Analisi dei componenti disponibili.

In figura si può notare come il framework permetta al progettista di inserire valori per l'$availability_{target}$ e il $cost_{target}$. In base a questi valori vengono calcolate le metriche sopra riportate. Per continuità con il caso di studio trattato nel capitolo due, anche qui vengono usati i valori di:

- $availability_{target}$ = 0,9
- $cost_{target}$ = 30

Questi valori verranno mantenuti identici per tutto lo studio. Da questa prima analisi si evince che il componente C_{12} fornisce sia il servizio s_1 che il servizio s_2 ma ha un valore di availability intrinseca troppo basso rispetto al target richiesto e diventa l'unico componente a non essere idoneo rispetto al requisito di availability. Infatti accanto al nome del componente C_{12} si può notare un «Not suitable» per il valore di BSA.

Tutti i componenti invece hanno un costo inferiore al target definito e risultano quindi idonei rispetto al requisito di costo (BSC = «suitable»).

Nonostante il componente C_{12} non risulti idoneo per il requisito di availability non viene escluso dallo studio infatti è possibile che il valore di adattabilità dell'intera architettura a cui il componente C_{12} prende parte sia comunque superiore a uno. (GAS $\geq$1) e l'architettura sia da considerarsi per lo studio delle architetture adattabili.

3.4 Calcolo delle metriche dello spazio iniziale delle architetture

Si prosegue lo studio analizzando tutto lo spazio di possibili architetture. Lo spazio è stato creato prendendo in considerazione tutti i componenti disponibili secondo ogni combinazione possibile. Per ogni servizio è stato selezionato un componente ed è stata creata l'architettura. Per l'insieme dei componenti definito in questo caso di studio lo spazio di architetture possibili è 36.

Per ognuna di queste architetture sono state studiate le metriche:

- $(GAS)\, \epsilon \mathbb{N} \{GAS = \prod FRA_{Ci}\}$
- $(GCS)\, \epsilon \mathbb{N} \{GCS = \sum FRC_{Ci}\}$

I risultati sono riportati sempre in figura [3.2] (parte in basso).

Definiti i valori delle metriche GAS e GCS di ogni architettura si è calcolato il valore di:

- $(SPA)\, \epsilon \mathbb{N} \{SPA = \sum A_i | GAS_{A_i} \geq 1\}$
- $(SPC)\, \epsilon \mathbb{N} \{SPC = \sum A_i | GCS_{A_i} \geq 1\}$.

I risultati di queste due metriche sono indicati in figura [3.2] (parte in basso) ultima riga. E sono il riassunto di questa prima analisi delle architetture.

Come si nota il valore di SPA equivale a 34 mentre SPC equivale a 36 questo significa che due architetture non soddisfano il requisito di availaibility mentre tutte le architetture risultano idonee rispetto al requisito di costo. A differenza dello studio sui singoli componenti ora il valore di GAS inferiore a 1 è discriminante e questo significa che tali architetture non verranno prese in considerazione per identificare tutte le architetture adattabili rispetto a quella attualmente in uso (almeno per quanto riguarda il requisito di availability).

Per fornire al progettista una visione di insieme di tutte le architetture sono stati calcolati, per ognuna di esse, anche i valori della cardinalità dell'insieme di architetture che si differenziano da essa solo per un componente e che rispettano il vincolo di availability e/o di costo.

Nello specifico nella figura [3.2] (parte in basso) per ogni architettura vengono indicati anche i valori di:

- $ADAA \epsilon \mathbb{N} \{ADAA = \sum A_i | GAS_{A_i} \geq 1 \wedge \forall C_i | \{\{(C_i \neq C_r \wedge C_i \in A_a) \implies (C_i \in A_i)\} \wedge \{(C_i = C_r \wedge C_i \in A_a) \implies (C_i \notin A_i)\}\}\}$
- $ADAC \epsilon \mathbb{N} \{ADAC = \sum A_i | GCS_{A_i} \geq 1 \wedge \forall C_i | \{\{(C_i \neq C_r \wedge C_i \in A_a) \implies (C_i \in A_i)\} \wedge \{(C_i = C_r \wedge C_i \in A_a) \implies (C_i \notin A_i)\}\}\}$

aggiungendo anche i nomi delle architetture che fanno parte dell'insieme delle architetture adattabili rispetto all'availability e i nomi di quelle quelle che fanno parte dell'insieme delle architetture adattabili rispetto al costo. In questo modo il progettista di sistema ha tutte le informazioni preliminari di analisi.

3.5 Calcolo delle metriche relative all'architettura attuale

Dopo aver analizzato lo spazio delle possibilità si seleziona l'architettura considerata in uso nel sistema. Si effettua l'analisi a livello di componenti, servizi forniti e richiesti e una analisi riassuntiva del sistema. Questa parte serve ad evidenziare non solo il livello di performance dell'architettura in uso ma anche quale componente risulta rilevante ai fini dello studio trattato nel capitolo tre.

Supponiamo di avere come architettura in uso l'architettura denominata *archi*0 in figura [3.2] (parte in basso).

Essa è composta dai componenti:

- C_{11} con $availability_{C_{11}} = 0,93$, $cost_{C_{11}} = 2$, $execution\_time_{C_{11}} = 1$,
  - fornisce il servizio s_1

  - richiede il servizio s_2 con $P_{C_{11}-S_2}$= 0,8 e $N_{C_{11}-S_2}$= 1
  - richiede il servizio s_3 con $P_{C_{11}-S_3}$= 1 e $N_{C_{11}-S_3}$= 2

- C_{12} con $availability_{C_{12}} = 0,85$, $cost_{C_{12}} = 8$, $execution\_time_{C_{12}} = 3$,
  - fornisce il servizio s_1
  - fornisce il servizio s_2
  - richiede il servizio s_3 con $P_{C_{12}-S_3}$= 0,5 e $N_{C_{12}-S_3}$= 2
  - richiede il servizio s_4 con $P_{C_{12}-S_4}$= 0,9 e $N_{C_{12}-S_4}$= 3

- C_{31} con $availability_{C_{31}} = 0,9$, $cost_{C_{31}} = 4$, $execution\_time_{C_{31}} = 1,5$,
  - fornisce il servizio s_3

- C_{41} con $availability_{C_{41}} = 0{,}92$ $cost_{C_{41}} = 4$, $execution\_time_{C_{41}} = 2$,
  - fornisce il servizio s_4

A questo punto verranno effettuate le analisi dettagliate dei componenti, dei servizi e dell'architettura concludendo con l'analisi riassuntiva.

Il framework che è stato sviluppato appositamente per questo studio riepiloga tutti i risultati di queste metriche in un'unica schermata, per questo motivo in questa sezione verrà presentata la stessa immagine evidenziando solo i risultati relativi ad ogni parte dell'analisi per favorire la comprensione dello studio.

3.5.1 Analisi dei componenti in uso

Si analizzano i componenti in uso secondo le metriche esposte a inizio capitolo ovvero FRA, BSA, FRC e BSC.

In figura [3.3] (parte selezionata) vengono mostrati i risultati di tali metriche per i componenti attualmente in uso nell'architettura *archi*0 .

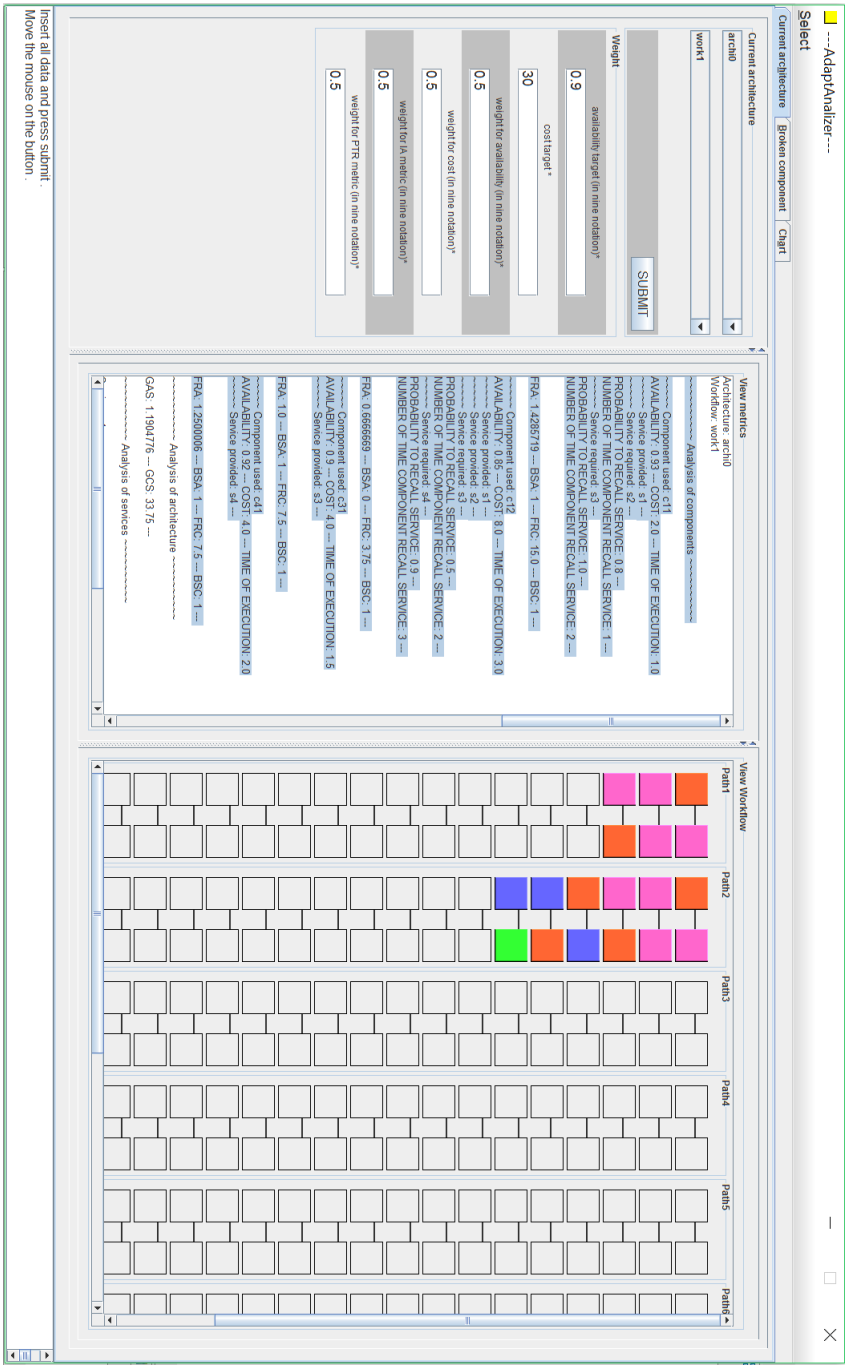

Figura 3.3: Analisi delle metriche relative ai componenti in uso nell'architettura.

Dal risultato delle metriche si evince che il componente C_{12} non soddisfa il requisito di availability mentre tutti i componenti risultano idonei per il requisito di costo. Questa analisi iniziale è semplicemente informativa per mostrare al progettista di sistema che eventualmente potrebbe intervenire sul componente C_{12} sostituendolo con uno con migliore availability che dia maggiori garanzie di funzionamento.

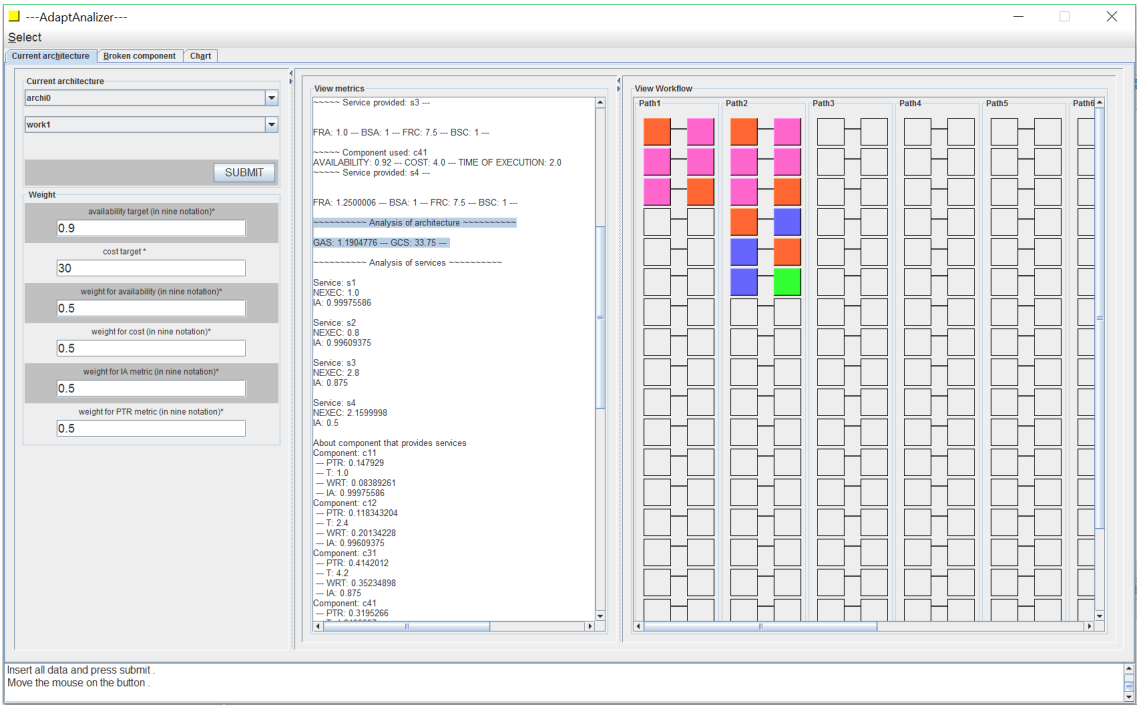

Figura 3.4: Analisi delle metriche relative all'architettura in uso. Analisi delle metriche relative all'architettura in uso.

3.5.2 Analisi dell'architettura attuale

In questa sezione si mostrano i valori per le metrice GAS e GCS.

In figura [3.4] (parte selezionata) vengono mostrati i risultati di tali metriche per l'architettura *archi*0.

Dal risultato delle metriche si evince che l'architettura offre un valore di GAS di poco superiore al minimo consentito (GAS = 1) mentre offre un buon livello di guadagno rispetto al budget massimo di spesa per l'acquisto di ogni componente.

Dal punto di vista del progettista queste ulteriori informazioni possono servire per decidere se sostituire qualche componente per aumentare il valore di GAS e quindi delle prestazioni dell'intero sistema. Scegliere di sostituire un componente per aumentare il valore di GAS significa seguire l'ipotesi secondo cui un valore più alto di GAS implichi un livello prestazionale più elevato.

3.5.3 Analisi dei servizi e dei componenti che li forniscono

In questa sezione si mostrano i valori delle metriche:

- $N_{exec} \epsilon \mathbb{N}^n | \{ \sum N_{exec_{Sj}} = \sum P_{Ci-s_j} * N_{Ci-s_j} * N_{Ci} \}$ e $N_{C_i} \epsilon \mathbb{N} | \{ N_{C_i} = \sum N_{exec_{C_i si}} \}$
- $PTR_S \epsilon \mathbb{N}^n | \{ PTR_{S_i} = \frac{N_{exec_{S_i}}}{\sum N_{exec}} \}$ e $PTR_C \epsilon \mathbb{N}^n | \{ PTR_{C_i} = \sum PTR_{C_i S_i} \}$
- $T_{Ci} \epsilon \mathbb{N} | \{ T_{Ci} = \sum N_{exec_{C_i - S_j} Ci-s_j} * T_{Ci} \}$ e $WRT \epsilon \mathbb{N}^n | \{ WRT_{C_i} = \frac{T_{Ci}}{\sum T} \}$

In figura [3.5] (parte selezionata) vengono mostrati i risultati di tali metriche per i servizi forniti all'interno dell'architettura *archi*0 .

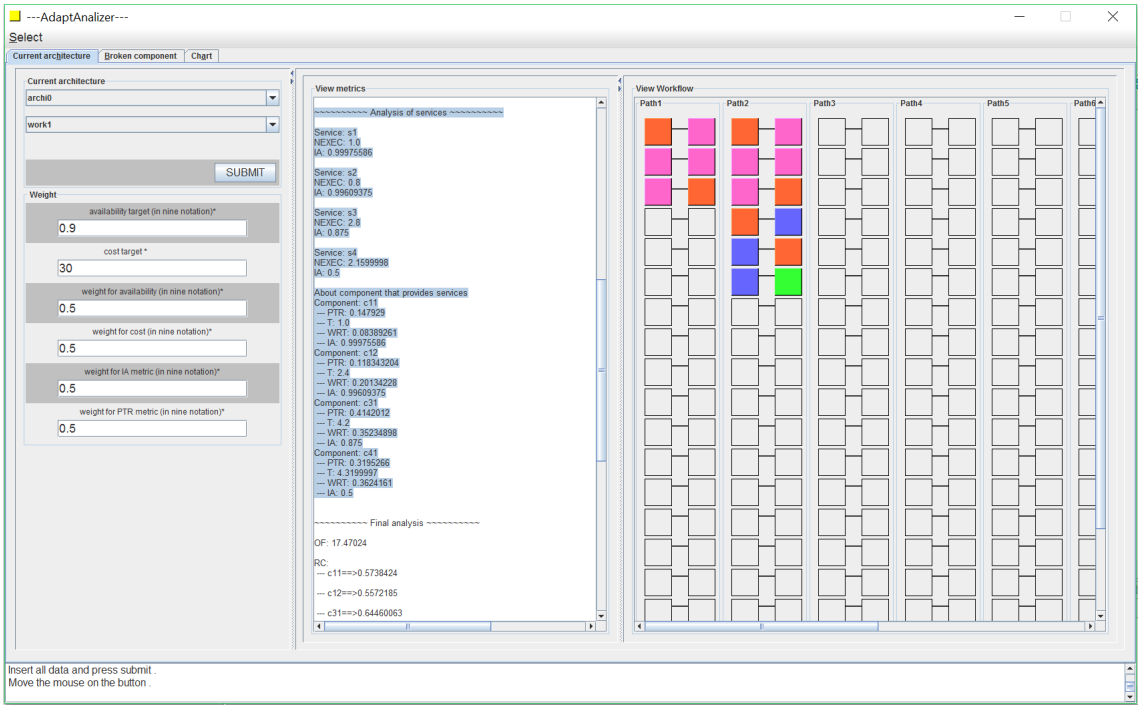

Figura 3.5: Analisi delle metriche relative ai servizi in funzione nell'architettura in uso.

Dal risultato delle metriche si evince che il servizio maggiormente in esecuzione è il servizio s_3. In questa architettura il servizio s_3 viene fornito dal componente C_{31} .

Il componente C_{31} quindi avrà il maggior valore di PTR ovvero sarà il componente con maggior probabilità di rompersi alla chiamata del servizio chiamato.

Per quanto riguarda la metrica WRT, ovvero il componente che permane in esecuzione per un tempo maggiore rispetto al tempo totale di esecuzione del sistema, si ottiene che il valore più alto è associato al componente C_{41}. Questo vuol dire che il componente C_{31} ha maggiori probabilità di rompersi ma il componente C_{41}è quello che rimane attivo per più tempo.

In un'ottica di individuazione dei componenti più adatti al sistema, questi risultati aiutano il progettista a concentrare l'attenzione su due componenti in particolare: C_{31} e C_{41}.

Per quanto riguarda la metrica IA la formula con cui si calcola è

$$IA_S \epsilon \mathbb{N}^{n_i} \{IA = 1 - (1 - P_{path_i} * u_{Si-act_i})^{m_{path_i}}\}$$

$$IA_C \epsilon \mathbb{N}^{n_i} \{IA_{C_i} = \prod IA_{C_i S_i}\}$$

ed il risultato è mostrato nella figura [3.5].

Per questa metrica è bene mostrare il workflow con cui l'applicazione va in esecuzione

In figura [3.6] è mostrato il workflow dell'applicazione presa come esempio nel capitolo tre.

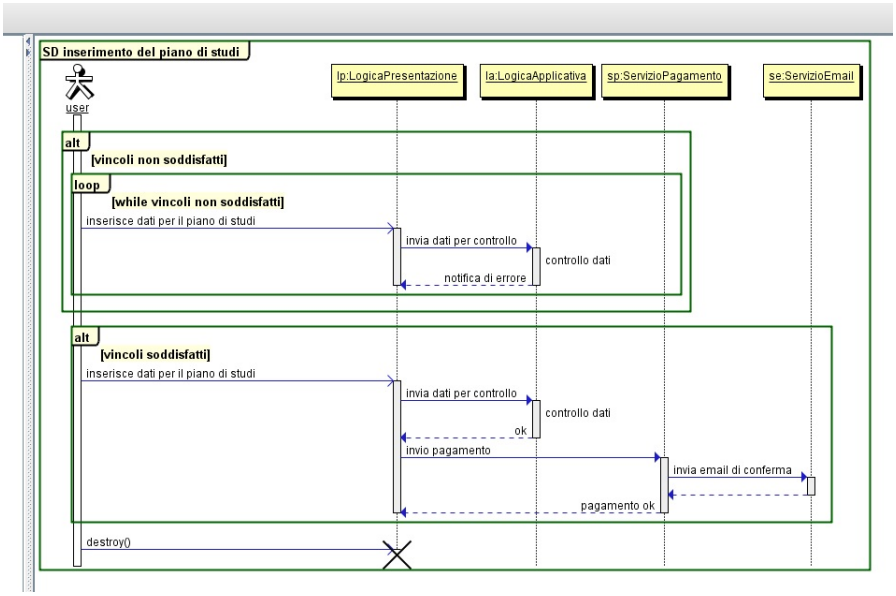

Figura 3.6: Workflow dell'applicazione mostrato secondo la rappresentazione comune.

In questo workflow si hanno i seguenti path:

- Path di esecuzione 1:
  - l'attività A_1 = «aggiunta dell'esame» che richiede la presenza della logica di presentazione che richiama la logica applicativa
  - l'attività seguente A_2 = è «analisi dei vincoli universitari» che viene svolta dalla logica applicativa che richiama i propri metodi
  - infine avremo A_3 = «notifica di errore allo studente» con cui la logica applicativa richiama la logica di presentazione.
- Path di esecuzione 2:
  - l'attività A_1 = «aggiunta dell'esame» che richiede la presenza della logica di presentazione che richiama la logica applicativa
  - l'attività seguente A_2 = è «analisi dei vincoli universitari» che viene svolta dalla logica applicativa che richiama i propri metodi
  - A_4 = la logica applicativa manda una «notifica di procedi con il pagamento all'utente» e richiama la logica di presentazione

- l'attività successiva è A_5 = «l'utente effettua il pagamento», la logica di presentazione richiama il servizio di pagamento.
- A_6 = «pagamento andato a buon fine» il servizio di pagamento invia una notifica alla logica di presentazione per avvisare l'utente.
- A_7 = «notifica via email del pagamento» il servizio di pagamento invia una mail all'utente.

Esattamente come nell'esempio trattato nel capitolo tre si ha:

- la logica di presentazione è il componente C_{11} che fornisce il servizio s_1 con cui l'utente si collega all'applicazione
- la logica applicativa è il componente C_{12} che fornisce il servizio s_2 con cui la logica applicativa analizza i vincoli
- la banca è il componente C_{31} che fornisce il servizio di pagamento s_3
- il gestore di posta è il componente C_{41} che fornisce il servizio di email s_4

Facciamo la supposizione che ogni componente fornisca un solo servizio e per tanto si passa a scomporre direttamente le attività in base al componente che fornisce il servizio non essendoci ambiguità nella fruizione del servizio stesso.

Quindi scomponiamo le attività in base ai componenti utilizzati:

- l'attività A_1 richiede C_{11} e C_{12} con $u_{C_{11}-act_1} = 1$ e $u_{C_{12}-act_1} = 1$
- l'attività A_2 richiede C_{12} con $u_{C_{12}-act_2} = 2$. si suppone infatti che la logica invochi i propri metodi per valutare il vincolo di inserimento degli esami e poi il risultato venga rimandato alla logica per un totale di 2 utilizzi.
- l'attività A_3 richiede C_{11} e C_{12} con $u_{C_{11}-act_3} = 1$ e $u_{C_{12}-act_3} = 1$
- l'attività A_4 richiede C_{11} e C_{12} con $u_{C_{11}-act_4} = 1$ e $u_{C_{12}-act_4} = 1$
- l'attività A_5 richiede C_{11} e C_{31} con $u_{C_{11}-act_5} = 1$ e $u_{C_{31}-act_5} = 1$
- l'attività A_6 richiede C_{11} e C_{31} con $u_{C_{11}-act_6} = 1$ e $u_{C_{31}-act_6} = 1$
- l'attività A_7 richiede C_{41} e C_{31} con $u_{C_{41}-act_7} = 1$ e $u_{C_{31}-act_7} = 1$

La metrica viene calcolata secondo la formula:

$$IA_S \epsilon \mathbb{N}^{n_i} \{IA = 1 - (1 - P_{path_i} * u_{Si-act_i})^{m_{path_i}}\}$$

$$IA_C \epsilon \mathbb{N}^{n_i} \{IA_{C_i} = \prod IA_{C_i S_i}\}$$

In figura [3.7] (parte selezionata) vengono mostrati i path eseguiti nell'architettura *archi*0 e a lato si hanno gli schemi grafici dei path stessi. Ogni quadratino corrisponde al componente in uso nell'attività del path. Il colore di ogni quadratino è il colore che viene selezionato nel framework al momento dell'inserimento del componente.

Per l'utilizzo del framework si rimanda al capitolo cinque.

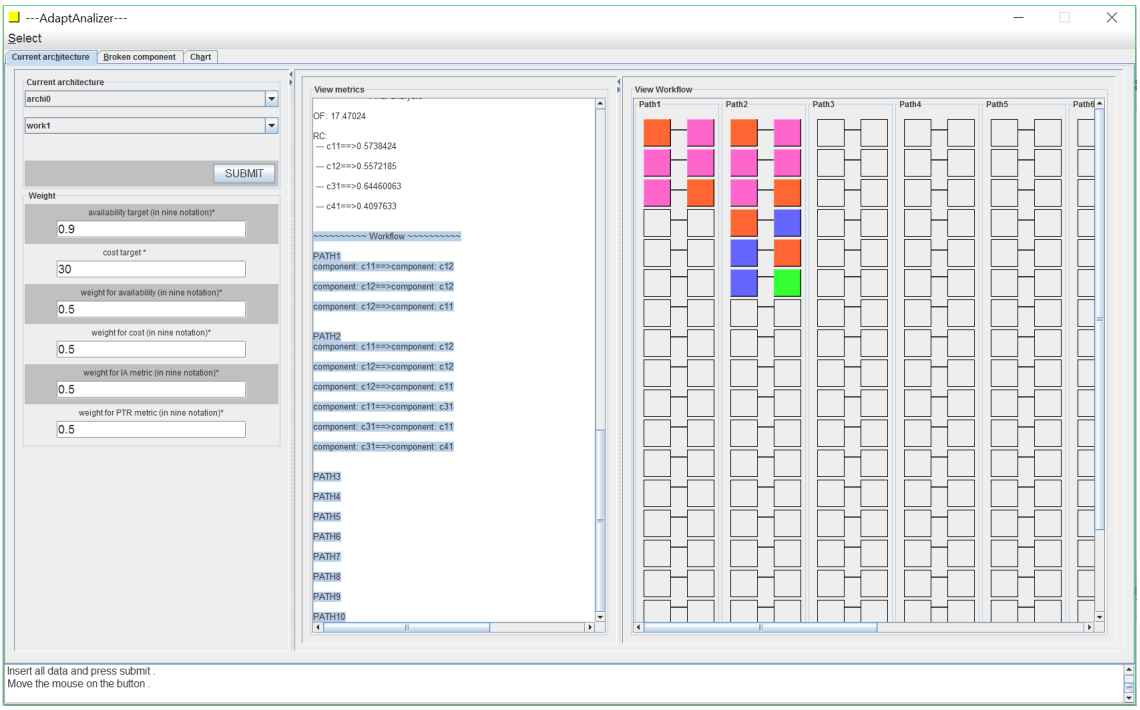

Figura 3.7: Riassunto del workflow dell'architettura in uso.

Per quanto riguarda i risultati della metrica si rimanda alla figura [3.5] (parte selezionata). Dal risultato si evince che il componente realmente in esecuzione per un numero di volte maggiore rispetto agli altri è il componente C_{11} seguito dal componente C_{12}. Alla luce di queste nuove informazioni ora il progettista di sistema dovrà decidere se dare maggior peso al componente C_{31} perchè ha maggiori probabilità di rompersi alla chiamata del servizio e quindi degradare le prestazioni del sistema o decidere di dare maggior rilevanza al servizio fornito dal componente C_{11} (o a pari merito a C_{12}) in quanto è il componente maggiormente richiamato durante tutto lo svolgimento dell'applicazione. Scegliere quale di questi componenti sostituire spetta al progettista, lo studio intanto indica in quale direzione guardare.

3.5.4 Analisi riassuntiva finale dell'architettura

In questa sezione si presenta l'analisi riassuntiva:

- per l'architettura in uso secondo la metrica $max\,(OF)\,\epsilon\mathbb{N}\!\upharpoonright\!\{OF = W_1 * GAS + W_2 * GCS\}$

- per i componenti che forniscono i servizi $RC \epsilon \mathbb{N}^{n} | \{RC_{Ci} = W_3 * IA_{Ci} + W_4 * PTR_{Ci}\}$

Per comprendere il risultato delle metriche OF e RC si riportano i valori dei risultati delle metriche usate per calcolarli

- $architettura_1$: $GAS_{architettura_1}$= 1,1905 e $GCS_{architettura_1}$= 33,75
- $PTR_{C_{11}}$ =PTR_{S_1}= 0,1479
- $PTR_{C_{12}}$ =$PTR_{S_2} = 0,1183$
- $PTR_{C_{31}}$ =$PTR_{S_3} = 0,4142$
- $PTR_{C_{41}}$ =$PTR_{S_4} = 0,3195$
- Per il componente C_{11} si ha: $IA_{C_{11}}$ = 0,9921 (in figura è leggermente differente per questione di arrotondamento)
- Per il componente C_{12} si ha: $IA_{C_{12}}$ = 0,9921 (in figura è leggermente differente per questione di arrotondamento)
- Per il componente C_{31} si ha: $IA_{C_{31}}$ = 0,875
- Per il componente C_{41} si ha: $IA_{C_{41}}$ = 0,5

Inoltre si suppone che il peso attribuito ad ogni parte delle metriche sia uguale a 0,5.

In figura [3.8] (parte selezionata) vengono mostrati i risultati di tali metriche riassuntive dell'architettura *archi*0 .

Il valore della metrica OF permette di confrontare l'architettura in atto con le architetture adattabili e si apprezzerà meglio più avanti nella trattazione del caso di studio.

I valori della metrica RC associati ad ogni componente invece riassumono quanto detto poc'anzi ovvero che lo studio ha evidenziato che il componente C_{31} ha maggiori probabilità di rompersi alla chiamata del servizio e quindi di degradare le prestazioni del sistema. Inoltre lo studio ha evidenziato che il componente maggiormente in esecuzione è C_{11} (o a pari merito a C_{12}) . In questo caso di studio si è supposto che sia PTR che IA abbiano lo stesso peso nell'analisi quindi il componente con maggior valore di RC è C_{31} ovvero è il componente con maggiori probabilità di rompersi alla chiamata ma è anche uno dei componenti maggiormente in esecuzione.

A mio avviso questa metrica riassume tutto lo studio. Si consideri l'ipotesi iniziale fatta ovvero che il sistema deve mantenere il livello di prestazione nella fornitura del servizio finale il più possibile costante. Partendo da questa assunzione si era passati a dire che lo studio voleva informare il progettista sul componente di maggior rilevanza nell'architettura e questo si ottiene grazie alla metrica riassuntiva RC. A questo punto

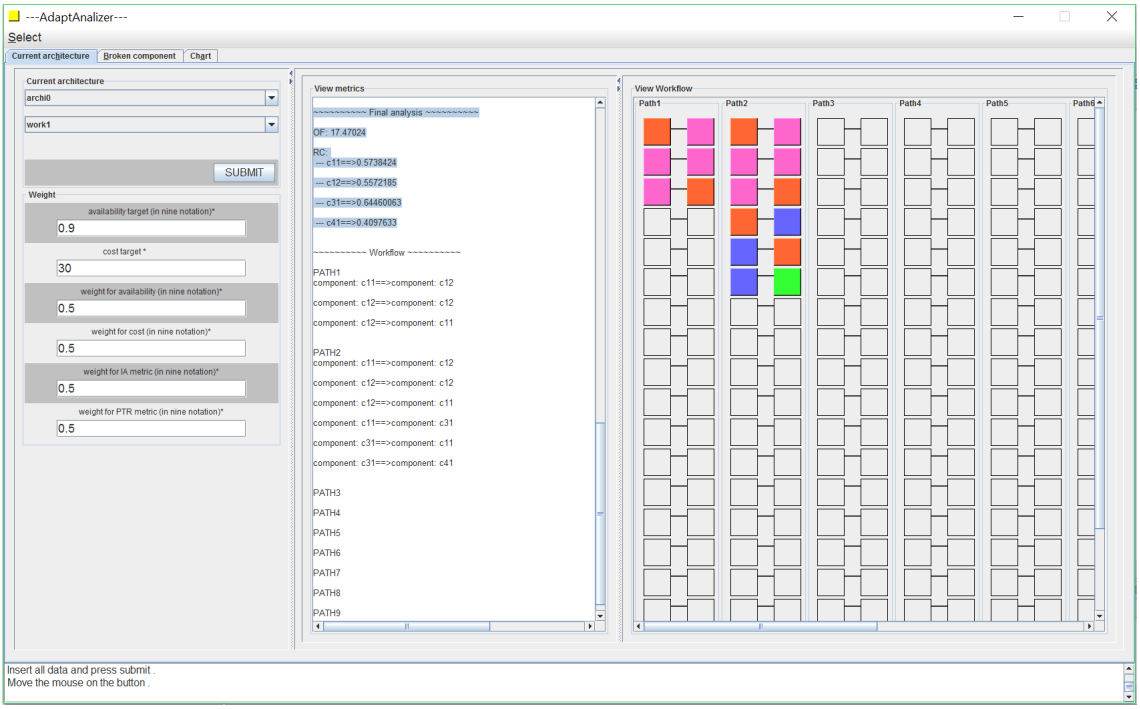

Figura 3.8: Analisi riassuntiva dell'architettura in uso.

si lascia al progettista la possibilità di scegliere se acquistare un componente per sostituire C_{31} con una availability molto più alta del componente in uso. Seguendo questa logica e rispettando la seconda ipotesi fatta (che un componente con maggiore availabililty abbia anche un costo di molto maggiore) si è volutamente mantenuta separata l'analisi di adattabilità delle architetture rispetto al requisito di availability e rispetto al costo. Questo viene fatto per dare al progettista un grado di libertà di scelta lungo cui muoversi.

Nelle sezioni seguenti si affronta proprio il caso in cui si vuole sostituire il componente C_{31} .

3.6 Calcolo delle metriche relative alle architetture adattabili

Dopo aver analizzato l'architettura in uso si giunge alla conclusione di voler migliorare le sue performance puntando proprio sulla sostituzione del componente che si è rilevato essere il maggiormente importante: C_{31}.

A questo punto è necessario cercare tutte le architetture adattabili rispetto a quella in uso. In figura [3.2] (parte in basso) sono mostrate, per ogni architettura, tutte le architetture che si differenziano da essa per un solo componente indicando quali di queste sono idonee per il requisito di availability e/o di costo. Da questo insieme

di architetture vengono selezionate tutte quelle che si differenziano dall'architettura attuale ($archi0$) solo per il componente C_{31}.

I requisiti da rispettare sono i medesimi dell'architettura in uso come i pesi da associare alle metriche di analisi finali.

Ci si avvale delle metriche:

- $ADAA\epsilon\mathbb{N}\text{¦}\{ADAA = \sum A_i\text{¦}GAS_{A_i} \geq 1 \wedge \forall C_i\text{¦}\{\{(C_i \neq C_r \wedge C_i \in A_a) \implies (C_i \in A_i)\} \wedge \{(C_i = C_r \wedge C_i \in A_a) \implies (C_i \notin A_i)\}\}\}$
- $ADAC\epsilon\mathbb{N}\text{¦}\{ADAC = \sum A_i\text{¦}GCS_{A_i} \geq 1 \wedge \forall C_i\text{¦}\{\{(C_i \neq C_r \wedge C_i \in A_a) \implies (C_i \in A_i)\} \wedge \{(C_i = C_r \wedge C_i \in A_a) \implies (C_i \notin A_i)\}\}\}$

per estrarre le architetture adattabili.

3.6.1 Analisi dell'architettura adattabile selezionata

In figura [3.9] si può notare come il framework lavori per estrarre le architetture idonee all'adattabilità.

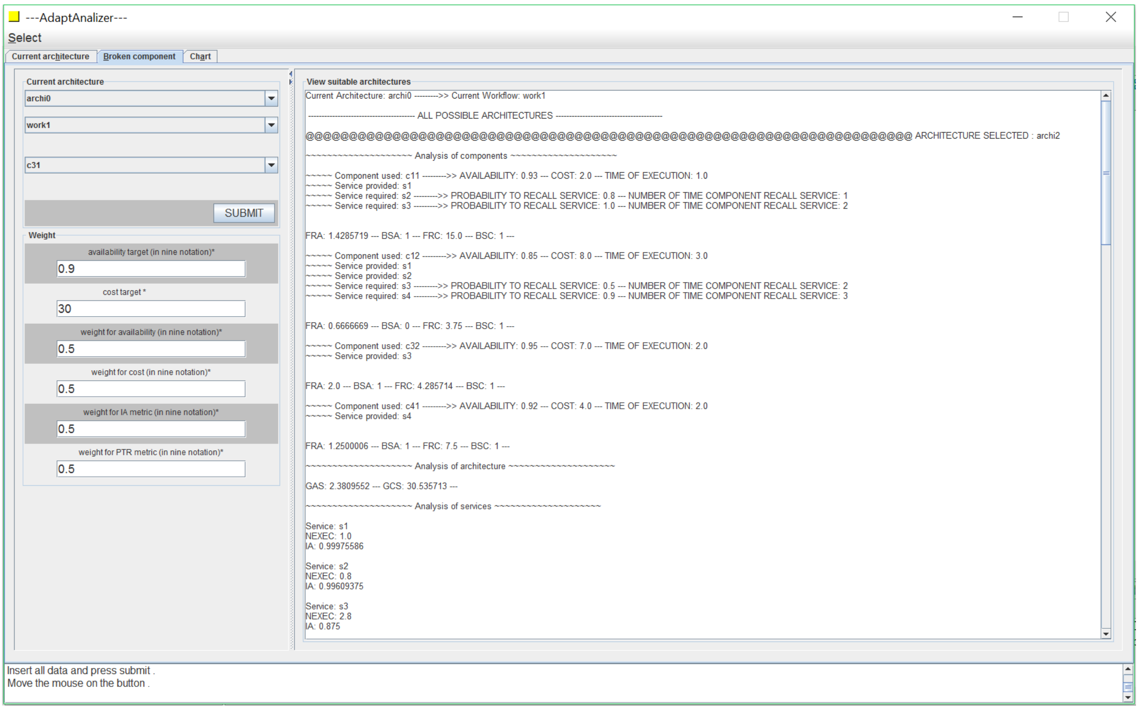

Figura 3.9: Analisi delle architetture adattabili parte 1.

In questa figura si nota come il framework abbia estratto l'architettura $archi2$ e per essa abbia effettuato il medesimo studio mostrato in precedenza per l'architettura in uso.

Di nuovo vengono studiati i componenti con le metriche FRA, BSA, FRC e BSC.

Poi lo studio analizza il valore dell'architettura rispetto alle metriche GAS e GCS. In seguito lo studio affronta l'analisi dei servizi in funzione nell'architettura $archi2$.

Infine lo studio presenta le metriche riassuntive dell'architettura *archi*2 : OF e RC di ogni suo componente.

In figura [3.10] si può notare la restante parte dell'analisi effettuata sull'architettura *archi*2 .

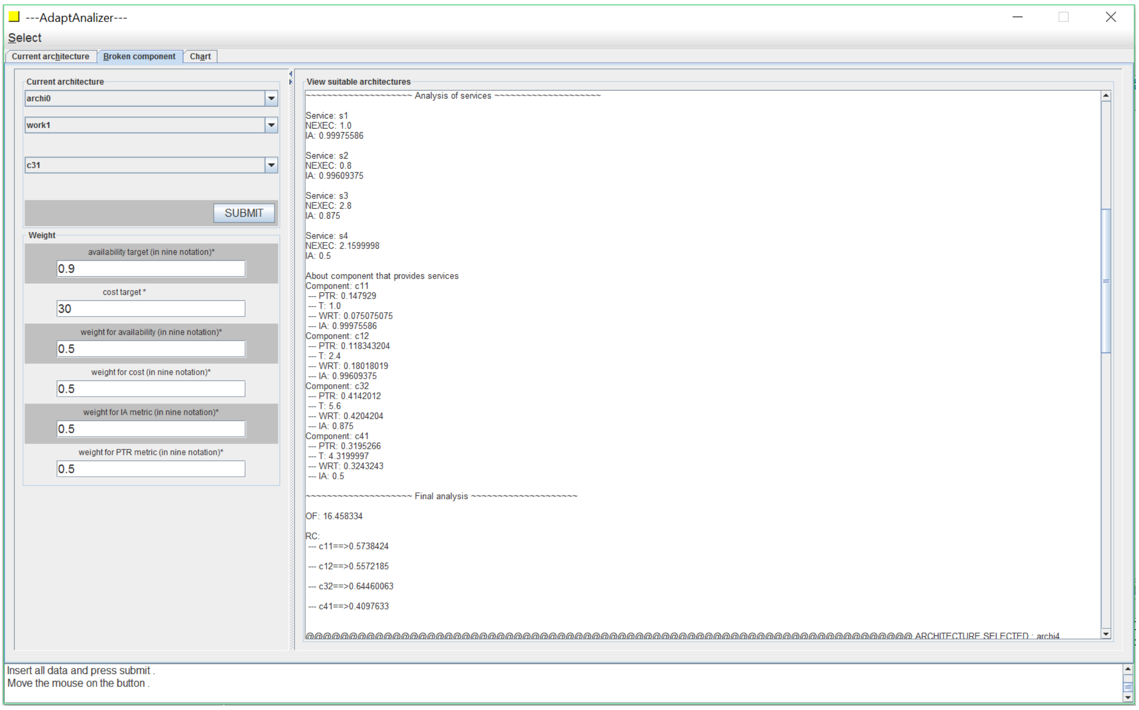

Figura 3.10: Analisi delle architetture adattabili parte 2.

Il framework estrae due architetture: *archi*2 e *archi*4. L'analisi dell'architettura *archi*4 non viene riportata in quanto segue lo stesso approccio dell'architettura in uso e di *archi*2.

Effettuando un confronto solamente tra i dati delle architetture *archi*0 e *archi*2 si evince che:

- $archi_0$:GAS_{archi_0}= 1,1905 e GCS_{archi_0}= 33,75
- $architettura_2$:GAS_{archi_2}= 2,3810 e GCS_{archi_2}= 30,5357

Questo può essere tradotto come: *archi*0 da minori garanzie di funzionamento ma permette di risparmiare maggiormente nell'acquisto dei componenti, *archi*2 invece fornisce migliori garanzie di funzionamento ma richiede un esborso maggiore. Il risultato è perfettamente in linea con le ipotesi fatte finora.

Dato che i pesi associati a GAS e GCS sono i medesimi allora *archi*2 ottiene un valore finale di OF inferiore a quello dell'architettura in uso. Qualora il progettista avesse scelto di puntare sul requisito di availability avremmo avuto un valore di OF maggiore per *archi*2.

Analizziamo ora i valori di RC: anche nell'architettura *archi*2 il componente maggiormente rilevante è il componente che fornisce il servizio s_3.

A questo punto il progettista deve effettuare una scelta e si basa sui valori della metrica OF. Il framework è stato studiato per mostrare un grafico riassuntivo di tale metrica così da fornire immediamente l'informazione al progettista.

3.7 Grafici di confronto delle architetture adattabili

In figura [3.11] si mostrano i grafici riassuntivi dello studio sulle architetture adattabili. Il framework ha selezionato *archi*2 e *archi*4 come uniche architetture adattabili rispetto a quella in uso se si vuole sostituire il componente C_{31}. Questa figura mostra il riassunto delle funzioni OF di ognuna delle architetture selezionate. Immediatamente appare chiaro che l'availability dell'architettura *archi*2 è nettamente superiore a quella di *archi*4 quindi a prima vista l'architettura *archi*2 è quella che offre maggiori garanzie di funzionamento, di contro l'architettura *archi*4 è quella che permette di risparmiare sull'acquisto dei componenti.

Giunti al termine dell'analisi del caso di studio è bene fare un commento su questo studio: lo spazio iniziale di architetture è formato da 36 architetture. Lo studio permette di estrarne solamente due semplificando di molto il lavoro del progettista. Per quanto questo studio sia solo un tentativo di approccio al campo dei sistemi auto - adattativi quantomeno permette di ridurre di molto lo sforzo computazionale del progettista.

Per la comprensione dell'utilizzo del framework si rimanda il lettore al capitolo cinque.

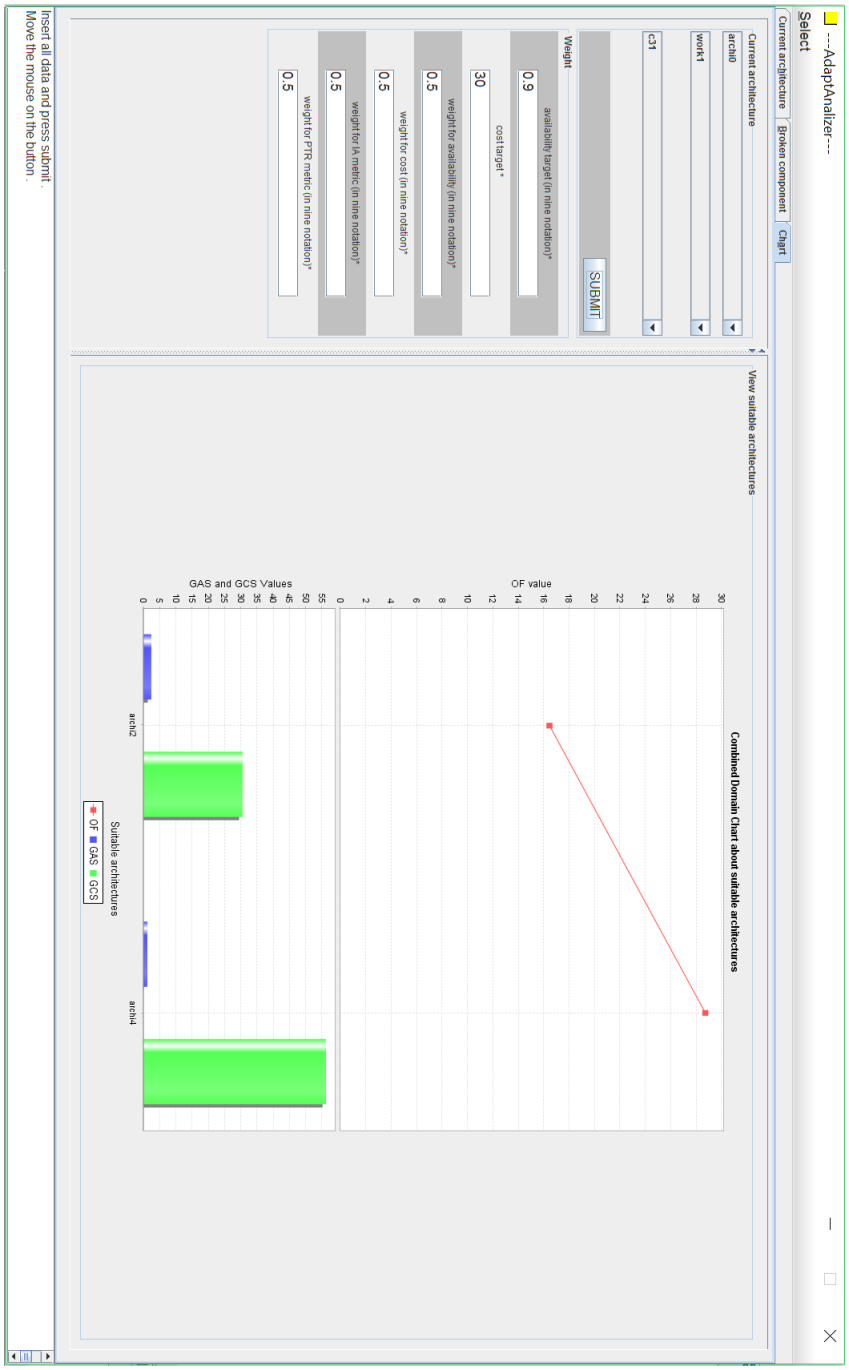

Figura 3.11: Analisi grafica riassuntiva della metrica OF per le architetture adattabili.

Capitolo 4

Strumentazione: il framework AdaptAnalyzer

Nel capitolo tre si è presentato un caso pratico di studio mostrando i risultati ottenuti tramite il framework AdaptAnalyzer sviluppato ad hoc, in questo capitolo si spiegherà nel dettaglio come utilizzarlo.

4.1 Introduzione

Il framework nasce con lo scopo di semplificare il lavoro del progettista. A fronte delle metriche che ho presentato per l'analisi dei servizi all'interno di una architettura risulta evidente che lo sforzo computazionale cresca rapidamente all'aumentare dei dati da analizzare quindi del numero di possibili architetture da considerare. L'obiettivo del mio studio relativo ai servizi è quello si portare il progettista ad una scelta più consapevole a fronte di informazioni più accurate per far questo però non si vuole obbligarlo ad estenuanti calcoli con il rischio di commettere errori. Adapt Analyzer vuole essere uno strumento di analisi automatica che riceva dal progettista i dati grezzi necessari all'analisi ed elabori i risultati finali.

Il framework è scaricabile al link [6].

In questa parte della tesi si vuole guidare il lettore passo passo all'approccio del framework per questo motivo inizialmente verrà spiegata la tipologia di framework realizzato, passando poi alla presentazione degli strumenti utilizzati per realizzarlo. A seguire si ha una parte relativa all'importazione del framework per chi volesse testarlo, infine verrà spiegato come utilizzare il framework nel dettaglio indicando i vari passaggi di inserimento dei dati fino all'analisi dei risultati ottenuti.

4.2 Il framework : un EJB3 JPA project

Verranno ora spiegati tutti gli acronimi per rendere il lettore consapevole del contesto di sviluppo in cui nasce il framework realizzato per lo studio presentato nel capitolo due.

- ***Enterprise JavaBean (EJB)*** : Essi sono i componenti software che vengono utilizzati lato server per implementare la logica di business di un'applicazione, web o desktop, sulla piattaforma Java EE. Lo scopo è fornire servizi pronti all'uso per la parte di front-end ovvero la parte che si interfaccia con l'utente. Essa è chiamata livello di presentazione o presentation layer. La maggiore parte dei casi questo livello di presentazione si trova in forma di pagina web se si tratta di un'applicazione web o di una semplice schermata desktop simile ai programmi quotidianamente lanciati se si parla di applicazioni lanciate dal menù dei programmi del proprio computer. Perchè questo sia possibile è necessario pensare ad una architettura software multi tier ovvero con più livelli in cui il livello deputato al server, con la logica di business, risiederà in remoto mentre il livello applicativo (application layer) sarà presente sul computer dell'utente. Spesso tale computer è definito come thin client ovvero il cosiddetto client leggero nel senso che non ha alcuna responsabilità nell'esecuzione dell'applicazione se non quello di fornire il livello di presentazione con cui l'utente si interfaccia al sistema. Le specifiche per gli EJB definiscono diverse proprietà che questi devono rispettare. La più importante di queste è la persistenza ma non sono da meno:

  - il supporto alle transazioni,
  - la gestione della concorrenza e della sicurezza
  - l'integrazione con altre tecnologie, come JMS, JNDI, e CORBA.

  Lo standard attuale, EJB 3.2, è stato completato nella primavera del 2013. Con la versione Enterprise JavaBeans 3.0 si introduce la possibilità di dichiarare e configurare gli Enteprise JavaBeans mediante il meccanismo delle annotations ovvero tramite l'inserimento di commenti con una @ davanti. Tali commenti indicano al sistema di che oggetto si tratta e il sistema agirà di conseguenza. Da questa versione in poi, un EJB non deve più estendere alcuna classe specifica. Tale modifica viene spesso citata come Plain Old Java Object (POJO). Cosa di notevole importanza: per la persistenza vengono abbandonati gli Entity Bean e c'è Java Persistence API (JPA).

  Le specifiche Enterprise JavaBean si pongono l'obiettivo di fornire una metodologia standard per implementare la logica di funzionamento delle applicazioni di tipo enterprise, ovvero quella categoria di applicazioni che forniscono servizi

via Internet su larga scala. Spesso e volentieri la realizzazione di tali applicazioni pone lo sviluppatore di fronte a una serie di problematiche tecniche piuttosto laboriose da risolvere. Queste specifiche vorrebbero fornire una soluzione per semplificare lo sviluppo descrivendo in dettaglio come realizzare un application server.

I tipi di Enterprise JavaBean che si possono trovare con le nuove versioni sono soltanto due, in quanto gli Entity Bean sono stati deprecati. Brevemente verranno ora specificati:

- ***EJB di sessione***: Detti anche Session EJB. Gestiscono l'elaborazione delle informazioni sul server, fanno quindi da interfaccia tra i client e i servizi offerti dai componenti disponibili sul server. Ne esistono di due tipi: con stato e senza stato.
  - ***Con stato***: Vengono anche detti stateful. Lo stato non è persistente, però l'accesso al bean è limitato ad un unico client.
  - ***Senza stato***: Detti anche stateless. Essere senza stato è una caratteristica che permette un accesso concorrente alle funzionalità offerte dal bean.
- ***EJB guidati da messaggi***: Detti anche Message driven EJBs. Essi utilizzano il Java Message Service (JMS) per iscriversi a un argomento (topic) o a una coda (queue) e, si attivano alla ricezione di un messaggio inviato all'argomento o alla coda a cui sono iscritti.
- ***EJB di Entità***: Vengono detti anche Entity EJB. Oggi non sono più supportati perchè con le vecchie versioni 2.0 e 2.1 dello standard EJB avevano dimostrato di mantenere un livello delle prestazioni molto basso ed i programmatori preferivano utilizzare chiamate JDBC dirette o framework di persistenza come Hibernate. Per ovviare a questo problema, è stato introdotto Java Persistence API. Essi inglobano gli oggetti sul lato server usati per la memorizzazione dei dati. Forniscono quindi la persistenza dei dati: in questo caso è il bean a occuparsi del salvataggio e recupero dei dati a cui fa riferimento. Il salvataggio può avvenire usando qualsiasi supporto, nella maggior parte dei casi si una base di dati [8].

Riassumendo si ha che EJB3 è una revisione profonda di EJB volta alla semplificazione dello sviluppo concentrandosi sulla scrittura di oggetti più semplici [1].

Il primo requisito per l'utilizzo di EJB3 richiede l'installazione di una versione java JDK compatibile. EJB3 fa un uso estensivo delle annotazioni che sono state introdotte con la versione 5 di java. Inoltre semplifica anche le API per gli entity bean usando Hibernate.

Il secondo requisito è quello di avere un application server compatibile con EJB3. Nel framework è stato utilizzato il server JBoss.

- ***Persistence Framework***: Java EJB3 persistence framework è una collezione di specifiche per la persistenza degli oggetti in un database relazionale. Queste specifiche in JBoss sono implementate dal framework Hibernate. EJB3 nasconde completamente la complessità di configurazione di Hibernate e questo permette di mappare facilmente gli oggetti col database senza necessità di ulteriori configurazioni. Tutto questo grazie sempre alle annotazioni introdotte con Java 5. Lo scopo della persistenza in un ambiente orientato agli oggetti è quella di salvare lo stato degli oggetti. Il persistence framework di EJB3 permette di utilizzare, per questa finalità, un database relazionale. Di fatto "mappa" una classe java con una tabella e gli attributi della classe con le colonne della tabella [17].

La Figura [4.1] mostra il funzionamento di JPA all'interno di un tool.

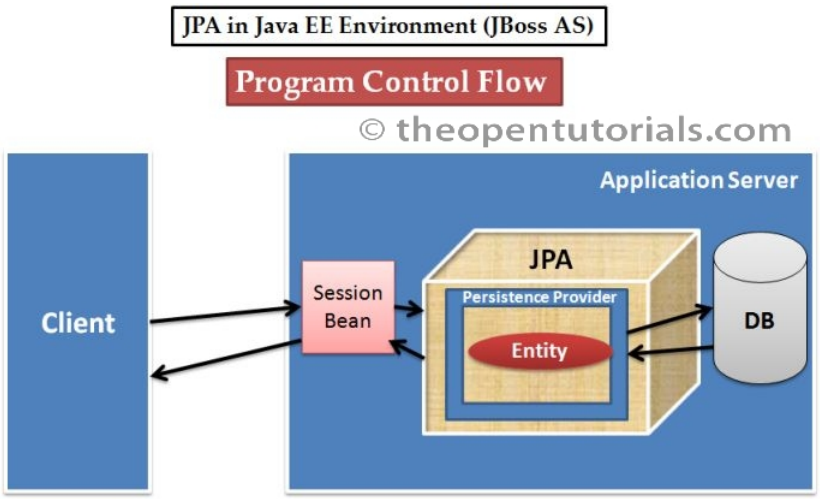

Figura 4.1: Come lavora il JPA [18].

- ***JDK (Java Development Kit)***: Esso è l'insieme degli strumenti che vengono utilizzati dai programmatori Java soprattutto per la realizzazione di applicazioni desktop. È un prodotto della Oracle. Oggigiorno è fortemente utilizzato quando è necessario sviluppare applicazioni più complesse, ad esempio sviluppo ed esecuzione di programmi IDE. Il JDK ha come componenti principali un set di strumenti di programmazione tra cui:
  - java: utilizzato per eseguire (interpretare) i file classe generati precedentemente dal javac.
  - javac: trasforma (compila) il file sorgente in bytecode.

  - javadoc: crea una documentazione di base a partire dai commenti inseriti nel codice sorgente.
  - jar: utilizzato per gestire i file JAR.

  Solitamente il JDK viene fornito insieme alla JVM (java virtual machine) e alle varie librerie di java (API Java), contenendo dunque anche il Java Runtime Environment (JRE) [11].

- ***JRE (Java Runtime Environment)***: Esso è un ambiente di esecuzione per applicazioni scritte in linguaggio Java, distribuito gratuitamente da Sun Microsystems. Contiene la Java Virtual Machine, le librerie standard (API Java) e un launcher per le applicazioni Java per poter avviare i programmi scritti in linguaggio Java e già compilati in bytecode. Non va identificato con un ambiente di sviluppo software perchè non contiene tool di sviluppo (compilatori e/o debugger): per poter sviluppare in Java serve avvalersi del JDK [13].

- ***Hibernate***: Essa è una piattaforma middleware open source per lo sviluppo di applicazioni Java, attraverso l'appoggio al relativo framework, che fornisce un servizio di Object-relational mapping (ORM) che permette la gestione della persistenza dei dati sul database. Questo strato software si frappone quindi tra il livello logico di business e quello di persistenza dei dati sul database. Hibernate è stato originariamente sviluppato da un team internazionale di programmatori volontari coordinati da Gavin King; in seguito il progetto è stato proseguito sotto l'egida di JBoss, che ne ha curato la standardizzazione rispetto alle specifiche Java EE. Hibernate viene distribuito con una licenza LGPL sotto forma di librerie software che sono linkabili nel progetto di sviluppo software. Il suo funzionamento prevede il salvataggio dei dati nel database e il loro reperimento producendo ed eseguendo automaticamente le query SQL necessarie. L'intento è esonerare lo sviluppatore dall'intero lavoro relativo alla persistenza dei dati. Hibernate è tipicamente usato sia in applicazioni Swing che Java EE facenti uso di servlet o EJB di tipo session beans. La versione 3 di Hibernate arricchisce la piattaforma con nuove caratteristiche tra cui una annotazione stile JDK 5.0 (Java's metadata feature). Hibernate 3 è vicino anche alle specifiche di EJB 3.0 ed è usato come spina dorsale per l'implementazione EJB 3.0 di JBoss [9].

- ***JNDI (Java Naming and Directory Interface)***: Essa è una API Java per servizi di directory che ricopre un ruolo molto importante all'interno di un application server. Viene utilizzata dai client java per scoprire e ottenere dati e oggetti attraverso un nome. Tipici usi di JNDI includono:

- connettere un'applicazione Java a un servizio di directory esterno ad esempio connessione all'indirizzo di una base di dati.
- permettere a un Java Servlet di ottenere informazioni di configurazione fornite dal web container [12].

Per poter cercare un oggetto si utilizza Context.lookup () a cui si passa il nome dell'oggetto che si desidera recuperare [4].

4.3 L'ambiente di sviluppo

In questa parte verranno introdotti gli strumenti utilizzati per sviluppare il tool. Prima verrà inserito un breve accenno alla storia di ognuno di loro e poi verrà indicata la versione specifica usata per il framework.

- ***Eclipse***: Eclipse è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma. Venne ideato da un consorzio di grandi società come Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP e Serena Software, chiamato Eclipse Foundation. Esso è un software libero distribuito sotto i termini della Eclipse Public License. Fu ideato per lo sviluppo di software di vario genere, si passa infatti da un completo IDE per il linguaggio Java (JDT, "Java Development Tools") a un ambiente di sviluppo per il linguaggio C++ (CDT, "C/C++ Development Tools") e a plug-in che permettono di gestire XML, JavaScript, PHP e persino di progettare graficamente una GUI per un'applicazione JAVA (Window Builder). Il toolkit grafico di Sun Microsystems si appoggia a SW che sono librerie di nuova concezione che conferiscono ad Eclipse un'elevata reattività. La piattaforma di sviluppo è incentrata sull'uso di plug-in, delle componenti software ideate per uno specifico scopo, per esempio la generazione di diagrammi UML. Essendo scritto in Java, Eclipse è disponibile per le piattaforme Linux, HP-UX, AIX, macOS e Windows [7]. La Figura [4.2] mostra il logo di Eclipse.

Figura 4.2: Il logo di Eclipse [5].

- ***JBoss***: Attualmente ha cambiato nome in WildFly mentre prima era noto come JBoss AS o semplicemente JBoss. Esso è un application server open source che implementa le specifiche Java EE. WildFly è un sistema multipiattaforma, interamente realizzato in Java. Il sistema è stato originariamente creato dalla società "JBoss Inc."; nel 2006 è stato acquistato da Red Hat per 420 milioni di

dollari. Viene gestito come progetto open source ed è sostenuto da una enorme rete di sviluppatori. A WildFly sono associati una quantità di altri prodotti, incluso Hibernate [16]. La Figura [4.3] mostra il logo di JBoss.

Figura 4.3: Il logo di JBoss [3].

- ***MySQL***: Chiamato anche Oracle MySQL è un Relational database management system (RDBMS) composto da un client a riga di comando e un server. Entrambi i software sono disponibili sia per sistemi Unix e Unix-like che per Windows; le piattaforme principali di riferimento sono Linux e Oracle Solaris. MySQL è un software libero rilasciato a doppia licenza, compresa la GNU General Public License ed è sviluppato per essere il più possibile conforme agli standard ANSI SQL e ODBC SQL. I sistemi e i linguaggi di programmazione che supportano MySQL sono molto numerosi tra cui Java [14]. La Figura [4.4] mostra il logo di MySQL.

Figura 4.4: Il logo di MySQL [**?**].

- ***MySQL Workbench***: Esso è uno strumento visuale di progettazione per database, che integra sviluppo SQL, gestione, modellazione dati, creazione e manutenzione di database MySQL all'interno di un unico ambiente [15].

4.3.1 Le versioni adottate nel framework

Per la realizzazione del tool sono state adottate delle versioni specifiche degli strumenti appena elencati in base alla loro compatibilità.

- Per Eclipse è stata scelta la versione 3.7 intitolata ECLIPSE INDIGO.
- Per il server JBoss è stata scelta la versione 7.1 final.

- Per MySQL è stata scelta la versione 5.1.
- Per MySQL Workbench è stata scelta la versione 5.6.
- Per quanto riguarda Java si è scelto di lavorare con la JRE 6 update 29.
- Come connettore per la persistence tra server e database si è scelto mysql connector v.5.1.19.

Per avviare il progetto è necessario:

- installare la JDK 6.29 da [20]
- scaricare ECLIPSE INDIGO 3.7 da [19]
- scaricarlo JBoss AS 7.1.1 Final da [2]
- estrarre il file del server
- aprire eclipse
- cliccare su Windows > Preferences > Server >Server Runtime Environments e aggiungere il server JBoss AS 7.1.1 Final
- nella parte in basso della schermata di eclipse cliccare sulla scheda Servers, cliccare con il tasto destro e fare New. Ricercare la tipologia di server JBoss AS 7.1.1 Final e aggiungere come server a runtime il server creato al punto precedente
- avviare il server e verificare che non ci siano errori
- per eventuali problematiche relative all'installazione dell'ambiente di sviluppo fare riferimento a [18]
- scaricare il progetto da [6]
- copiare la cartella del framework in C/Utenti/il nome del proprio utente/workspace/.
- scaricare il dumb del database e caricarlo nel db.
- aprire eclipse e importare il progetto come «progetto già esistente nella workspace»
- importare tutte le librerie aggiunte nella cartella e tutte le librerie del server consigliate in [18]
- per la persistence e per qualsiasi collegamento con il database fare riferimento a [18]

- per quanto riguarda il collegamento al database consiglio di seguire con attenzione il paragrafo 11 di [18].
  - Vi verrà richiesto un connettore al database e vi consiglio di utilizzare mysql-connector-java-5.1.19.jar o mysql-connector-java-5.1.22.jar già presenti nella cartella di progetto
- deployare il progetto
- avviare da /AdaptAnalizerTool/ejbModule/com/ibytecode/graphicsInterface/Index.java

4.4 L'utilizzo del framework

Dopo aver spiegato la strumentazione necessaria per lo sviluppo ora verrà trattata tutta la parte relativa al suo funzionamento.

4.4.1 Schermata di avvio e login

La schermata di Login si presenta come in figura [4.5].

Figura 4.5: Schermata Login di AdaptAnalizerTool.

Le operazioni possibili sono:

- inserire le credenziali di accesso e cliccare Login. Il server effettuerà il controllo nel database per la verifica dei dati immessi e in casi affermativo permetterà l'accesso altrimenti un messaggio di errore verrà mostrato a video.
- registrare un nuovo utente e cliccare su submit. La logica di business prevede un controllo di tutti gli utenti già registrati per verificare che non vi siano utenti

con la stessa username. In caso di riscontro verrà notificato un messaggio di errore altrimenti la registrazione andrà a buon fine.

Nelle schermate successive verrà presentata la logica del framework. AdaptAnalizerTool è stato studiato per semplificare il lavoro al suo utilizzatore e per eventuali estensioni future.

Esiste una schermata per ogni passaggio logico di inserimento dei dati:

- Schermata di inserimento dei componenti : in essa sarà possibile:
  - prima inserire un componente,
  - poi inserire il servizio che fornisce
  - ed infine inserire il servizio richiesto.
- Schermata di inserimento delle attività : in essa si rappresenta il workflow del sistema da studiare:
  - prima si mostra lo spazio di tutti i componenti e delle architetture possibili in base ai componenti inseriti nel sistema. Verranno presentati solo i componenti che forniscono un servizio. Per ogni componente saranno calcolate le metriche, come esposto nel capitolo due, in base ai valori di availability e costo limite inseriti dall'utente. Per le architetture verrà adottata la stessa politica: verranno mostrate solo architetture costituite da componenti che forniscono un servizio.
  - a questo punto si creano le attività semplici intese come «il componente x» richiama «il componente y»,
  - poi si creano i path alternativi che sono sequenze di attività eseguibili un numero di volte definito dall'utente
  - poi si crea il workflow inteso come insieme di path alternativi.
- Schermata relativa all'analisi finale
  - nella prima scheda verranno riassunti tutti i dati dell'architettura attuale con i valori di tutti le metriche calcolate e il workflow delle attività
  - in base alla scelta della architettura attuale e del componente da sostituire, nella seconda scheda, verranno mostrate tutte le architetture che possono sostituire quella attuale con relative metriche
  - nell'ultima schermata si ha un riassunto dei dati salienti dell'analisi di confronto tra le architetture trovate al punto precedente e verranno mostrati dei grafici.

4.4.2 Schermata di inserimento dei componenti

In figura [4.6] è mostrata la schermata di inserimento dei componenti. All'avvio è attiva sulla scheda di inserimento del componente. Le operazioni possibili sono:

- inserire un nuovo componente:
  - digitare il nome del componente
  - selezionare un colore (che attiverà il bottone SUBMIT)
  - inserire obbligatoriamente i dettagli di costo, availability e time (sono permessi solo numeri e il «.» per indicare le cifre decimali).
  - il bottone SUBMIT verrà attivato quindi cliccare su di esso per salvare il componente.

  La logica sottostante controllerà che il componente non sia già inserito nel database e permetterà il salvataggio altrimenti mostrerà un messaggio di errore. Il framework avverte anche quando i dettagli non sono stati inseriti.

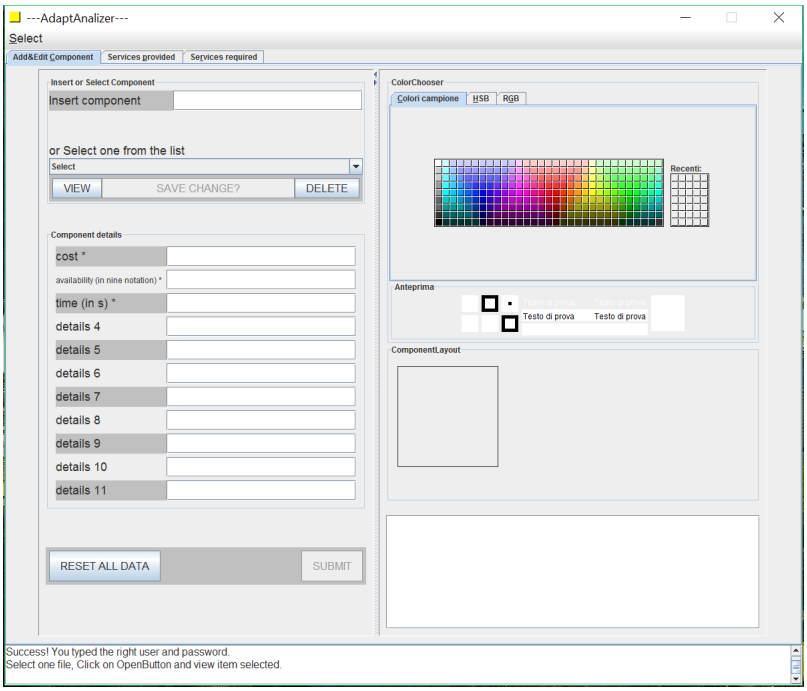

Figura 4.6: Schermata Inserimento componenti di AdaptAnalizerTool, scheda relativa all'inserimento dei componenti.

- selezionare un componente già esistente per modificarlo:
  - selezionare un componente tra quelli presenti nella combobox
  - cliccare il bottone VIEW per mostrarne i dettagli. In figura [4.7] è possibile visualizzare questa variante.

- eventualmente modificare i dettagli e cliccare su SAVE CHANGE? per salvare le modifiche
- cliccare il bottone DELETE per cancellare il componente scelto. Questa cancellazione ha effetto a cascata ovvero verrà cancellata ogni connessione tra componente e servizi che fornisce e componente e servizi che richiede.

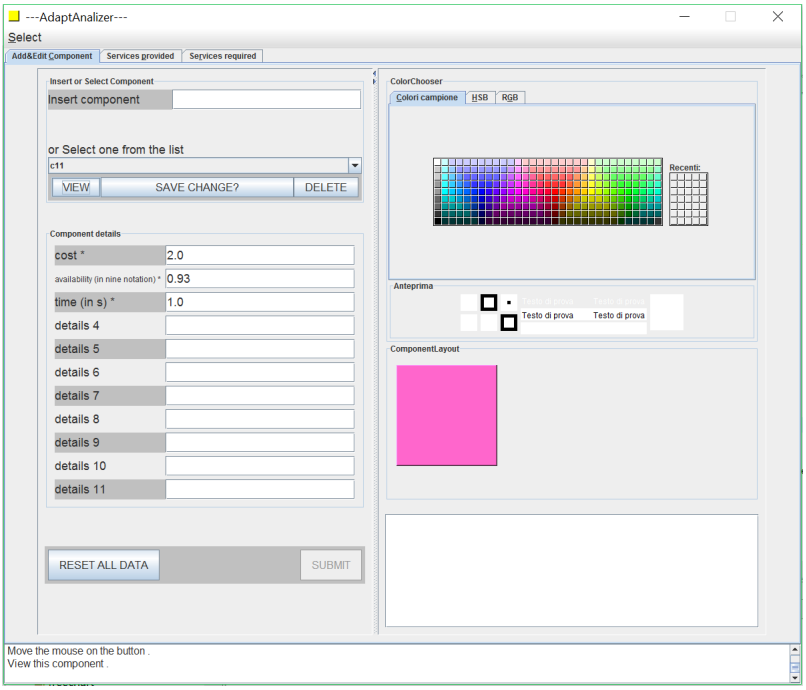

Figura 4.7: Schermata Inserimento componenti di AdaptAnalizerTool, scheda relativa all'inserimento dei componenti con visualizzazione di un componente già salvato.

4.4.3 Schermata di inserimento dei servizi forniti

In figura [4.8] è mostrata la schermata di inserimento dei servizi forniti dai componenti. Le operazioni possibili sono:

- inserire un nuovo servizio:
  - selezionare un componente tra quelli presenti nella combobox
  - inserire il nome del nuovo servizio che si vuole fornire
  - in questo caso non è obbligatorio inserire i dettagli del servizio. Sono presenti per eventuali sviluppi futuri dello studio.
  - il bottone SUBMIT è da cliccare per salvare il nuovo servizio fornito dal componente.

La logica sottostante controllerà che il componente non abbia già fornito questo servizio, o che il componente stia richiedendo questo servizio o che il servizio sia già presente nel database. In questo ultimo caso se il servizio è già inserito nel database basterà selezionarlo dalla combobox apposita. Questo viene fatto perchè esiste un limite implementativo che prevede l'aggiunta di massimo 30 servizi.

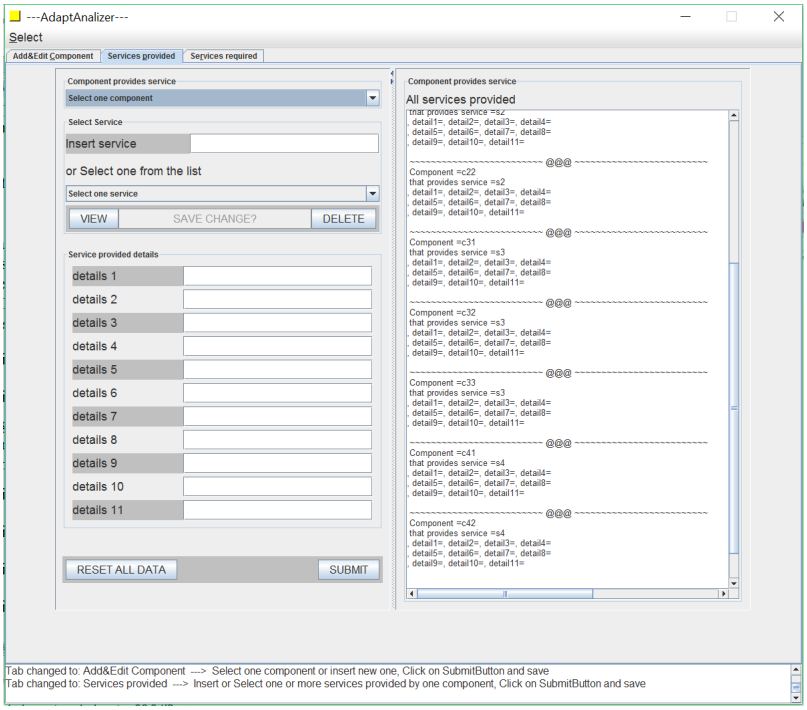

Figura 4.8: Schermata Inserimento componenti di AdaptAnalizerTool, scheda relativa all'inserimento dei servizi forniti dai componenti.

- selezionare un servizio già esistente:
  - selezionare un componente tra quelli presenti nella combobox
  - selezionare un servizio tra quelli presenti nella combobox
  - cliccare sul bottone SUBMIT per collegare il componente al servizio. Anche in questo caso la logica controlla che non vi siano problemi relativi. Il risultato del nuovo inserimento sarà visibile sul lato destro della schermata che riassume tutti i componenti con i relativi servizi forniti.
  - cliccare il bottone VIEW per mostrarne i dettagli della connessione tra componente e servizio. Verranno mostrati a video tutti i dettagli relativi a questa connessione solo se presente nel database.
  - eventualmente modificare i dettagli e cliccare su SAVE CHANGE? per salvare le modifiche

– cliccare il bottone DELETE per cancellare la connessione tra componente e servizio fornito. Attenzione in questo caso il framework cancella la connessione tra componente e servizio e non il servizio.

4.4.4 Schermata di inserimento dei servizi richiesti

In figura [4.9] è mostrata la schermata di inserimento dei servizi richiesti dai componenti. Le operazioni possibili sono:

- selezionare un componente tra quelli presenti nella combobox
- selezionare un servizio tra quelli presenti nella combobox

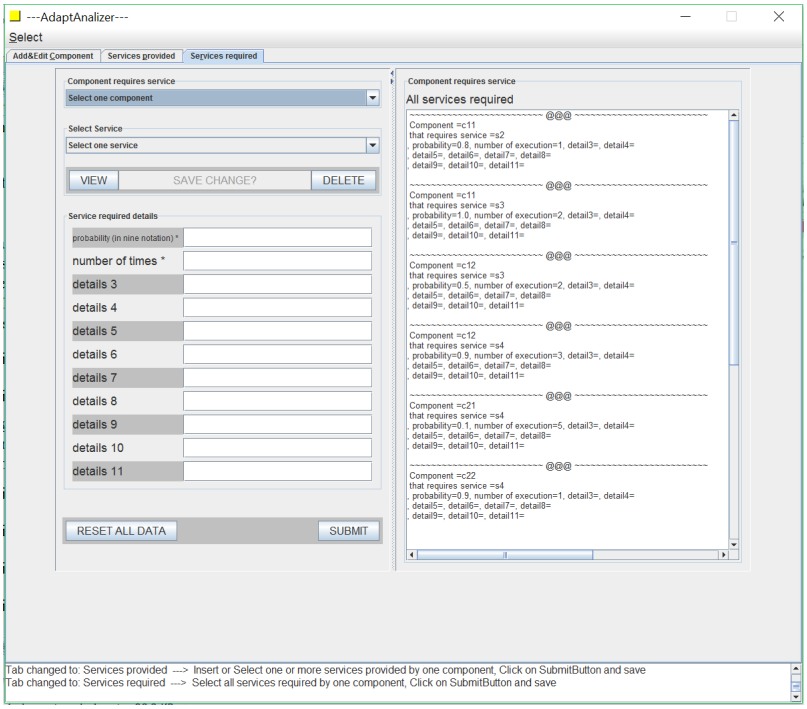

Figura 4.9: Schermata Inserimento componenti di AdaptAnalizerTool, scheda relativa all'inserimento dei servizi richiesti dai componenti.

- inserire obbligatoriamente i dettagli di probability (sono permessi solo numeri e il «.» per indicare le cifre decimali) e execution time (sono permessi solo numeri) . Poi cliccare SUBMIT. Il salvataggio nel database sarà permesso solo se il componente non richiede già questo servizio ne lo fornisce. Il riassunto dell'inserimento sarà mostrato nella parte destra della schermata.
- cliccare il bottone VIEW per mostrarne i dettagli relativi alla connessione tra componente e servizio richiesto. Sarà visibile a schermo solo se la connessione è presente nel database.

- eventualmente modificare i dettagli e cliccare su SAVE CHANGE? per salvare le modifiche.
- cliccare il bottone DELETE per cancellare la connessione tra componente scelto e servizio indicato.

4.4.5 Schermata dello spazio delle possibilità

Entrando nella sezione delle attività si ha come prima schermata un riassunto di tutti i componenti con i valori delle metriche calcolate e di tutte le architetture generabili dai componenti inseriti nel database.

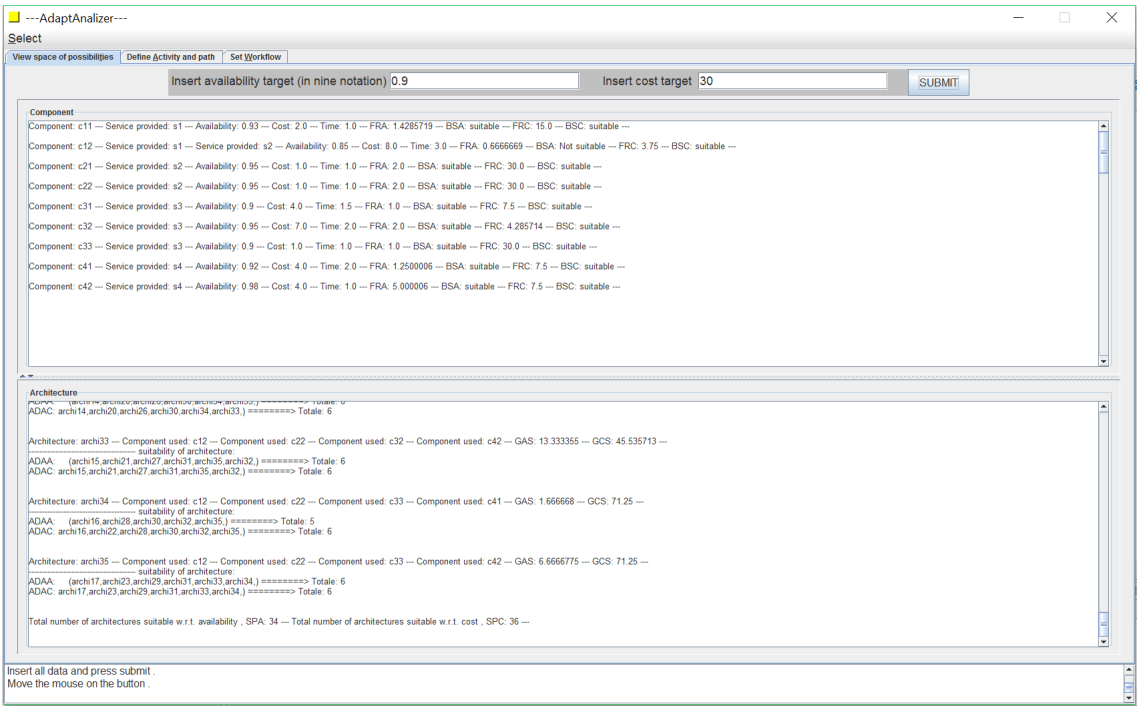

Figura 4.10: Schermata Inserimento Attività di AdaptAnalizerTool, scheda relativa alla visualizzazione dello spazio delle possibilità generato.

In figura [4.10] è possibile notare come gli unici dati richiesti all'utente siano availability e costo. Inserire solo numeri e il «.» per indicare la virgola. Premere SUBMIT per inviare. Si avvisa l'utente che questa analisi impiega un tempo dell'ordine di qualche minuto. Questo è dovuto al fatto che devono essere valutate tutte le architetture e per ognuna di essere si calcolano quelle ritenute adattabili. Tale tempistica aumenta all'aumentare del numero di architetture inserite.

4.4.6 Schermata di inserimento delle attività

La scheda di inserimento di attività semplici e costituzione del path merita un approfondimento aggiuntivo. Riprendiamo per continuità lo stesso esempio di

applicazione mostrato nel capitolo uno e poi esteso nel capitolo due.

In figura [4.11] è mostrato il workflow dell'applicazione.

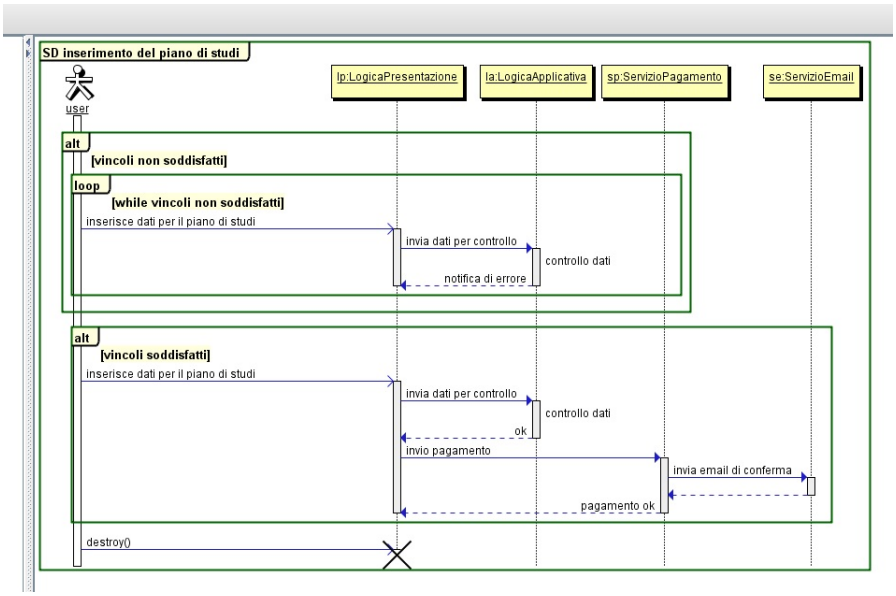

Figura 4.11: Workflow dell'applicazione mostrato secondo la rappresentazione comune.

Ai fini dello studio serve capire quante volte il componente che fornisce il servizio richiesto va in esecuzione.

Quindi si scompone ogni path in attività:

- Path di esecuzione 1:
  - l'attività A_1 = «aggiunta dell'esame» che richiede la presenza della logica di presentazione che richiama la logica applicativa
  - l'attività seguente A_2 = è «analisi dei vincoli universitari» che viene svolta dalla logica applicativa che richiama i propri metodi
  - infine avremo A_3 = «notifica di errore allo studente» con cui la logica applicativa richiama la logica di presentazione.
- Path di esecuzione 2:
  - l'attività A_1 = «aggiunta dell'esame» che richiede la presenza della logica di presentazione che richiama la logica applicativa
  - l'attività seguente A_2 = è «analisi dei vincoli universitari» che viene svolta dalla logica applicativa che richiama i propri metodi
  - A_4 = la logica applicativa manda una «notifica di procedi con il pagamento all'utente» e richiama la logica di presentazione

- l'attività successiva è A_5 = «l'utente effettua il pagamento», la logica di presentazione richiama il servizio di pagamento.
- A_6 = «pagamento andato a buon fine» il servizio di pagamento invia una notifica alla logica di presentazione per avvisare l'utente.
- A_7 = «notifica via email del pagamento» il servizio di pagamento invia una mail all'utente.

In figura [4.12] è mostrato il workflow dell'applicazione riscritto in base alle attività.

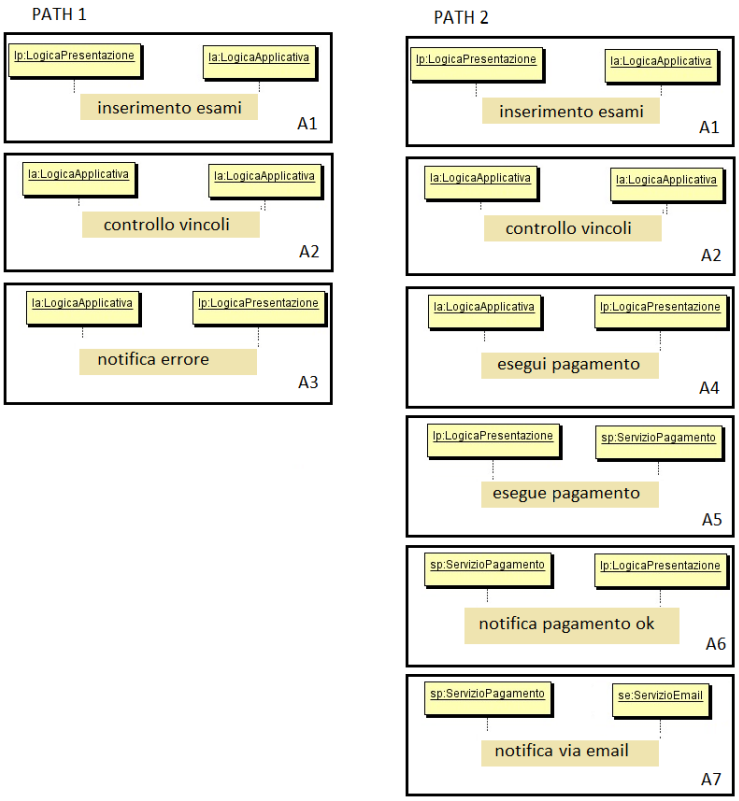

Figura 4.12: Workflow dell'applicazione riscritto rispetto alle attività.

Per fare questo si usa schermata del framework mostrata in figura [4.13].

Le operazioni possibili sono essenzialmente quattro ovvero inserimento e visualizzazione delle attività e inserimento e visualizzazione dei path:

- nella parte sinistra è possibile inserire una nuova attività:
  - digitare il nome dell'attività
  - selezionare il servizio di partenza e quello finale
  - cliccare il bottone SUBMIT per salvare l'attività.

  La logica sottostante controllerà che l'attività non sia già presente nel database e permetterà il salvataggio altrimenti mostrerà un messaggio di errore. Il framework avverte anche quando i servizi non sono selezionati.

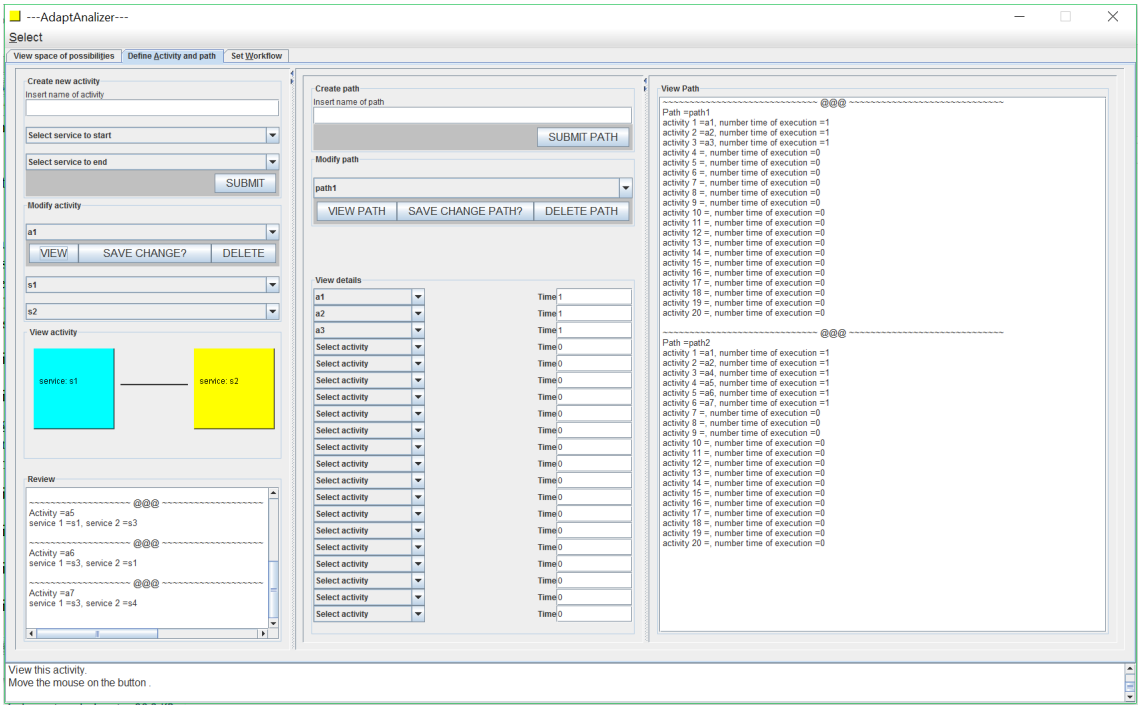

Figura 4.13: Schermata Inserimento attività di AdaptAnalizerTool, scheda relativa all'inserimento delle attività semplici e costituizione di path.

- selezionare una attività già esistente per modificarla:
  - selezionare il nome dell'attività dalla combobox
  - cliccare il bottone VIEW per mostrarne i dettagli. In figura [4.13] si può notare come ci siano due riquadri colorati in azzurro e giallo. L'implementazione in questa parte è stata pensata per dare all'utente la sensazione di creare una attività esattamente come descritta sopra.
  - si può anche cambiare i servizi che intervengono nell'attività e cliccare su SAVE CHANGE? per salvare le modifiche
  - cliccare il bottone DELETE per cancellare l'attività selezionata. Questa cancellazione ha effetto a cascata ovvero verrà cancellato ogni path che prevede l'uso si questa attività e di conseguenza il workflow che utilizza il path appena cancellato.

  in fondo alla colonna delle attività è presente un riassunto di tutte le attività già inserite nel database.

Nella parte centrale si può gestire l'inserimento e la visualizzazione dei path:

- inserire un nuovo path:
  - digitare il nome del nuovo path che si vuole inserire

  - selezionare almeno una attività coinvolta nel path. Per ragioni implementative non è possibile inserire più di 20 attività per path.
  - inserire il numero di volte che si vuole eseguire l'attività all'interno del path. In questo caso è consentito inserire solo numeri interi senza virgole o punti.
  - cliccare il bottone SUBMIT PATH per salvare il path.

La logica sottostante controllerà che il path non sia già presente nel database e permetterà il salvataggio altrimenti mostrerà un messaggio di errore. Il framework avverte anche quando non viene selezionata alcuna attività.

- selezionare un path già esistente:
  - cliccare il bottone VIEW per mostrarne i dettagli. Verrà mostrato a video il nome di tutte le attività coinvolte nel path con il numero di volte che viene eseguita ogni attività.
  - si può anche cambiare le attività coinvolte nel path e/o il numero di volte che vengono eseguite e cliccare su SAVE CHANGE? per salvare le modifiche
  - cliccare il bottone DELETE per cancellare il path selezionato. Questa cancellazione ha effetto a cascata ovvero verrà cancellato ogni workflow che utilizza il path appena cancellato.

  a lato è possibile vedere un riepilogo di ogni path creato.

4.4.7 Schermata di inserimento dei workflow

In figura [4.14] è mostrata la schermata di inserimento dei workflow che possono essere eseguiti dall'applicazione. Le operazioni possibili sono:

- inserire un nuovo workflow:
  - inserire il nome del workflow
  - inserire almeno un path. Per ragioni implementative non è concesso inserire più di 10 path
  - il bottone SUBMIT è da cliccare per salvare il nuovo workflow.

  La logica sottostante controllerà che il workflow non sia già stato inseritonel database. In questo ultimo caso la logica manderà un messaggio di errore.

- selezionare un workflow già esistente:
  - selezionare il nome tra quelli presenti nella combobox

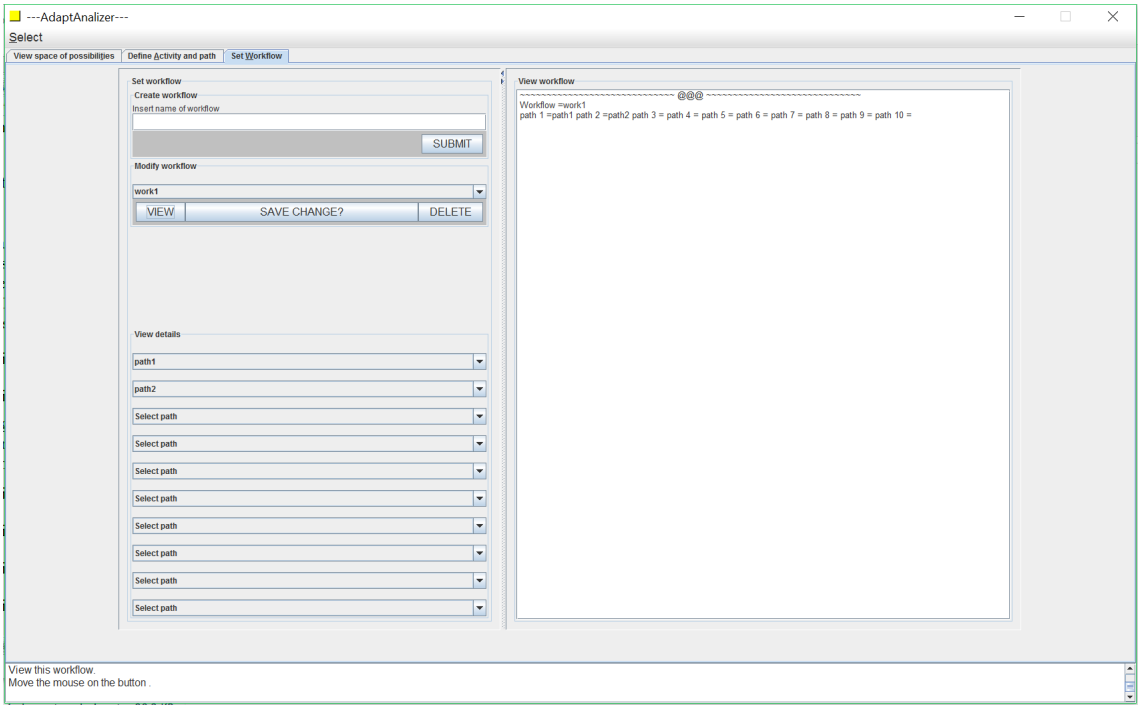

Figura 4.14: Schermata Inserimento componenti di AdaptAnalizerTool, scheda relativa all'inserimento dei servizi forniti dai componenti.

- cliccare il bottone VIEW per mostrarne i dettagli ovvero mostrare tutti i path che lo compongono.
- eventualmente è possibile modificare i path e cliccare su SAVE CHANGE? per salvare le modifiche
- cliccare il bottone DELETE per cancellare il workflow.

4.4.8 Schermata di analisi finale dell'architettura attuale

In questa sezione si hanno tre schede che riassumono le analisi effettuate dallo studio mostrato nel capitolo due.

La prima scheda è mostrata in figura [4.15] e riguarda l'analisi dell'architettura attuale. Per effettuare l'analisi è necessario:

- selezionare il nome dell'architettura attuale dalla combobox
- selezionare il workflow dell'architettura dalla combobox
- inserire obbligatoriamente i pesi associati ad availability, costo, probabilità di essere in esecuzione e peso associato alla metrica «in action». Sono permessi solo numeri e il «.» per indicare le cifre decimali.
- Poi cliccare SUBMIT. A questo punto i dati verranno elaborati e nella schermata centrale compariranno i risultati mentre nella parte destra verrà ripro-

dotto il workflow dell'architettura colorando i quadratini in base al colore del componente che entra in esecuzione per soddisfare il servizio presente nell'attività.

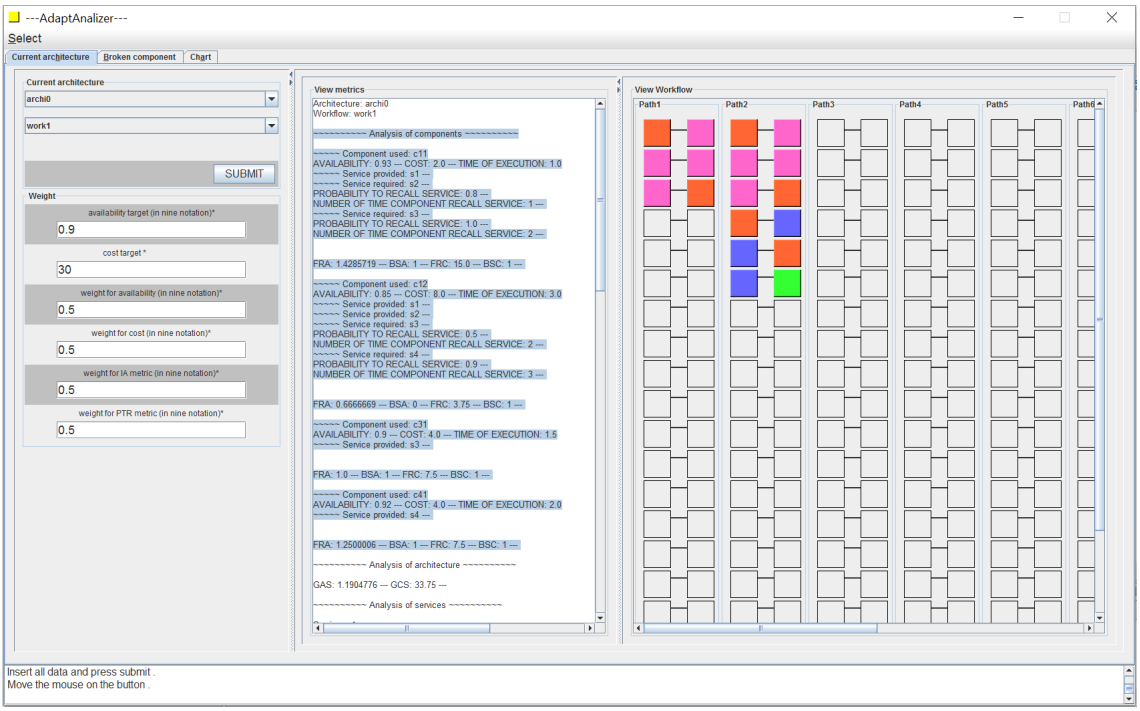

Figura 4.15: Schermata Analisi finale di AdaptAnalizerTool, scheda relativa all'analisi dell'architettura attuale.

4.4.9 Schermata di analisi finale delle architetture adattabili

La seconda scheda della sezione analisi finale è mostrata in figura [4.16] e riguarda l'analisi delle architetture adattabili rispetto all'architettura attuale. Per effettuare l'analisi è necessario effettuare gli stessi inserimenti della scheda precedente:

- selezionare il nome dell'architettura attuale dalla combobox
- selezionare il workflow dell'architettura dalla combobox
- inserire obbligatoriamente i pesi associati ad availability, costo, probabilità di essere in esecuzione e peso associato alla metrica «in action». Sono permessi solo numeri e il «.» per indicare le cifre decimali.
- Poi cliccare SUBMIT. A questo punto i dati verranno elaborati e nella parte destra verranno mostrati i risultati.

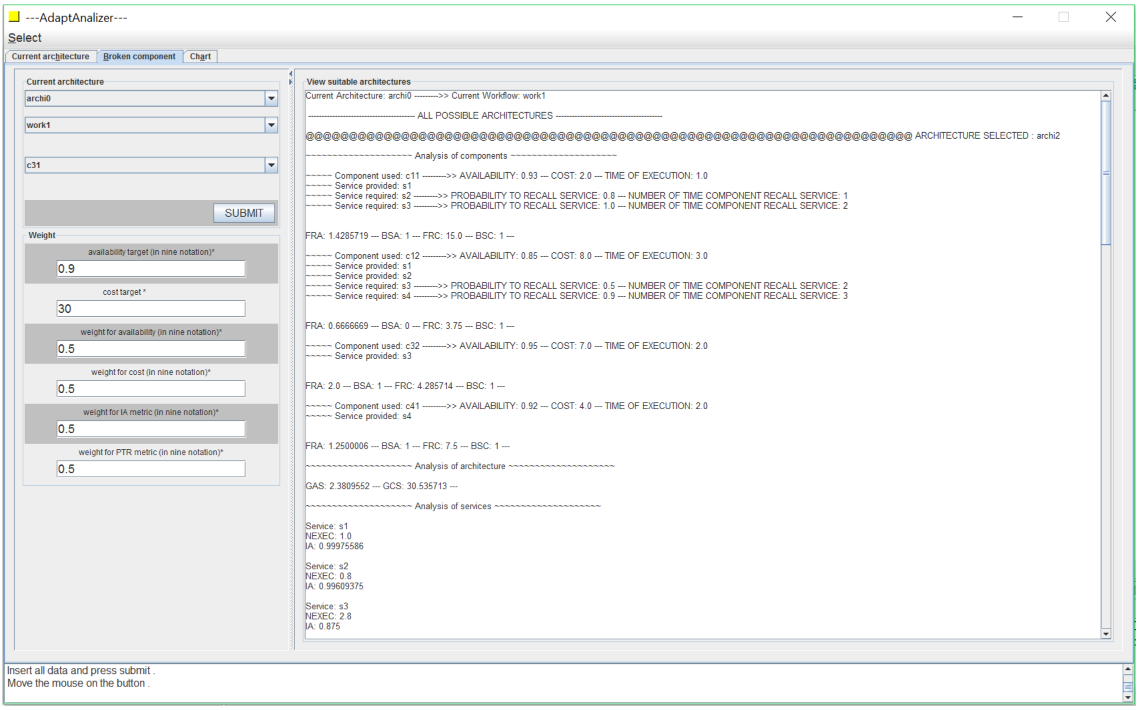

Figura 4.16: Schermata Analisi finale di AdaptAnalizerTool, scheda relativa all'analisi delle architetture adattabili.

4.4.10 Schermata di analisi finale con i grafici

La terza e ultima scheda della sezione analisi finale è mostrata in figura [4.17] e riguarda l'analisi delle architetture adattabili rispetto all'architettura attuale. Per effettuare l'analisi è necessario effettuare gli stessi inserimenti della scheda precedente:

- selezionare il nome dell'architettura attuale dalla combobox
- selezionare il workflow dell'architettura dalla combobox
- inserire obbligatoriamente i pesi associati ad availability, costo, probabilità di essere in esecuzione e peso associato alla metrica «in action». Sono permessi solo numeri e il «.» per indicare le cifre decimali.
- Poi cliccare SUBMIT. A questo punto i dati verranno elaborati e nella parte destra verranno mostrati i risultati sotto forma di grafici.

Nel capitolo successivo verranno riassunti tutti i limiti riscontrati nello studio e nell'implementazione e gli sviluppi ritenuti plausibili per il futuro.

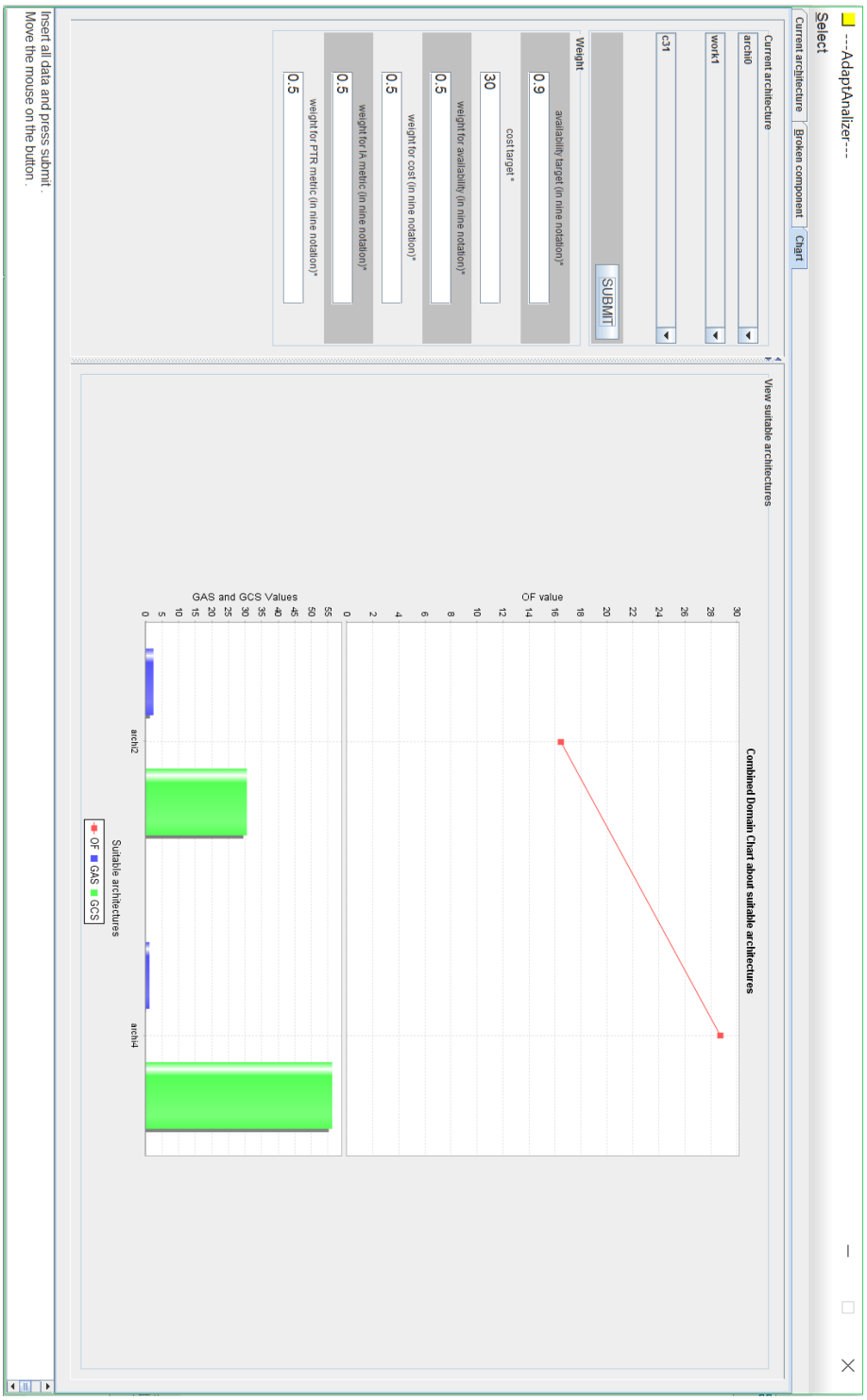

Figura 4.17: Schermata Analisi finale di AdaptAnalizerTool, scheda relativa ai grafici riassunti all'analisi delle architetture adattabili.

Conclusioni

Il campo dei sistemi auto - adattativi è stato studiato solo negli ultimi anni per rispondere all'esigenza degli utenti di avere applicazioni in grado di adattarsi alle loro esigenze.

Proprio per questo motivo, come più volte detto, non esistono metriche riconosciute universamente che possano essere usate per valutarli. Quanto descritto in questa tesi è un tentativo di approccio ai sistemi auto - adattativi analizzando alcuni loro aspetti.

L'obiettivo è cercare di fornire metriche utili che aiutino il progettista a rendere l'adattabilità di tali sistemi fattibile.

Nel capitolo uno è stato spiegato un primo tentativo di fornire tali metriche. Purtroppo però i sistemi auto - adattativi sono stati affrontati solo dal punto di vista statico analizzando componenti e architettura senza mai considerare il loro comportamento durante l'esecuzione dell'applicazione. In questo studio i componenti inseriti nelle architetture vengono considerati tutti ugualmente importanti senza diversificare il servizio in base al tempo in cui esso è in esecuzione e l'importanza che assume all'interno del sistema

Di fatto questo rappresenta il più grande limite di questo studio perchè fornisce un'informazione poco accurata sui componenti. Per questo motivo ho sentito l'esigenza di effettuare uno studio ulteriore che si concentrasse sull'analisi dei servizi cercando di fornire al progettista un'informazione ulteriore relativa ai componenti. Quello che si è mostrato nel capitolo due è stato il mio tentativo di estendere lo studio delle metriche di adattabilità analizzando i servizi nel dettaglio, differenziando la loro importanza nel sistema e mostrando il componente da cui può dipendere maggiormente il degrado delle prestazioni del sistema. Per poter perseguire il mio scopo si è reso necessario considerare il workflow di funzionamento del sistema e con esso una serie di calcoli relativi alla probabilità di trovare un servizio in funzione piuttosto che i calcoli relativi al numero di volte che il componente viene richiamato.

Queste valutazioni richiedono calcoli lunghi che aumentano di complessità con l'aumentare dei componenti inseriti nell'architettura.

Inoltre per determinare quali architetture siano maggiormente adattabili rispetto a quella in uso si rende necessario effettuare tali calcoli per ogni possibile architettura

che si possa creare in base all'insieme dei componenti disponibili.

Questo rende il lavoro del progettista piuttosto laborioso e soggetto ad errori di conteggio per questo motivo si è scelto di implementare un framework apposito che svolgesse il lavoro per il progettista.

L'obiettivo finale quindi diventa duplice, non solo fornire al progettista delle informazioni più accurate per permettergli di effettuare una scelta più consapevole ma anche quello di semplificare il suo lavoro. Con l'adozione di Adapt Analyzer il progettista non dovrà far altro che inserire i valori di availability target e cost target, i pesi associati all'importanza di questi due valori, i pesi con cui valutare se sia più importante sostituire un componente con maggiori probabilità di rottura alla chiamata o il componente maggiormente richiamato durante l'esecuzione del sistema ed infine i dati delle attività svolte dal sistema.

Inseriti questi valori il progettista si troverà a video tutti i risultati dello studio. Avrà a disposizione sia i valori delle metriche espresse nel capitolo uno sia quelle che ho proposto nel capitolo due e che valutano maggiormente i servizi forniti nell'applicazione.

Per semplificare ulteriormente il lavoro del progettista esiste una schermata riassuntiva in cui si prongono i grafici delle architetture selezionate dal framework. Il proposito del grafico è quello di mostrare in maniera esplicita e di immediata comprensione il risultato di tutta la ricerca in modo che non sia necessario controllare ogni risultato numerico delle metriche per poter effettuare una scelta.

Lo studio che ho proposto per l'analisi di adattabilità dei sistemi auto - adattativi si basa su ipotesi e pertanto richiede alcune assunzioni. Purtroppo tali assunzioni corrispondono anche a dei limiti. Esaminiamo questi limiti:

- ho assunto che possano esistere componenti che forniscono più di un servizio. Finora per questi casi ho supposto che essi siano responsabili solo per servizi che il componente fornisce e che non sono mai entrati in esecuzione prima. Si parla quindi di servizi che nascono dal momento in cui il componente viene richiamato. Questa assunzione mi permette di creare un ordine di attivazione dei servizi che semplifica l'implementazione del framework perchè non si considerano tutti i casi di attivazione. Uno sviluppo ulteriore potrebbe andare in questo senso ovvero studiare tutte le varianti di attivazione dei servizi da parte di qualsiasi componente.

- ho assunto che i componenti sostituibili possano essere solo uno alla volta per non aumentare di molto lo sforzo computazionale. Ulteriori sviluppi potrebbero approfondire questo aspetto.

- ho assunto che i servizi forniti dai componenti siano parte integrante di essi e ho studiato solo il comportamento dei servizi richiesti senza considerare alcuna caratteristica dei servizi forniti dando per scontato che sia forniti sempre

con probabilità 1 dal componente. Si potrebbe però analizzare anche questo aspetto. Ho implementato il framework per gestire questo sviluppo in quanto il componente viene tenuto separato dal servizio fornito ed esiste una schermata apposita di inserimento dei dettagli del servizio fornito dal componente che è del tutto personalizzabile.

- per quanto riguarda i servizi richiesti ho ritenuto interessante considerare la probabilità e il numero di volte con cui essi vengono richiesti ma potrebbero esserci altri parametri rilevanti di analisi. Essendo uno studio basato su ipotesi se ne potrebbero fare altre.

- ho effettuato lo studio rispetto ai requisiti di availability e costo per continuità con lo studio del capitolo uno ma si potrebbero cercare altri requisiti interessanti. Per esempio ho provato ad effettuare un'analisi delle tempistiche con la metrica WRT. La metrica per ora funziona perfettamente in un sistema isolato ma quando si parla di tempistiche di esecuzione di un componente (ovvero il tempo impiegato ad espletare una richiesta) è necessario considerare anche se il sistema si trova a dover fornire un servizio da remoto. Si potrebbe procedere in questo senso chiaramente non è di facile sviluppo essendo del tutto aleatorio il tempo di totale di fornitura di un servizio quando esso viene richiesto da remoto.

Per quanto riguarda un aspetto puramente implementativo esistono i seguenti limiti:

- per ora esiste l'accesso al sistema solo agli utenti autorizzati con l'inserimento delle credenziali poi però si accede al medesimo database di componenti e servizi. L'idea iniziale era di permettere l'accesso al framework solo al team di sviluppo dei progettisti. Volendo si potrebbe differenziare le informazioni da mostrare in base all'utente che vi può accedere così da lasciare la totalità delle informazioni al progettista ed eventualmente permettere ai proprietari di sistema di esaminare solo i risultati finali.

- per quanto riguarda la costruzione di ogni architettura ho permesso l'inserimento fino ad un massimo di 30 componenti supponendo che i sistemi reali non abbiano necessità di sforare questo limite. Per quanto riguarda il workflow non è possibile eccedere il limite di 10 path da 20 attività l'uno. Ho ritenuto fosse una buona approssimazione delle reali attività svolte da una applicazione generica. Tali limiti sono dovuti al tempo totale di computazione dello spazio delle possibili architetture e della ricerca delle architetture adattabili. Tempo che cresce notevolmente all'aumentare dei componenti per ogni architettura e del numero di quest'ultime.

Nonostante i limiti implementativi e di studio, i risultati ottenuti mostrano che sono arrivata lo stesso ad un miglioramento dell'adattabilità dei sistemi auto - adatta-

tivi infatti ora il progettista ha a disposizione un'informazione importante: quale componente può inficiare tutto il funzionamento del sistema. Inoltre ritengo di aver semplificato il lavoro del progettista fornendogli uno strumento automatico di analisi.

Bibliografia

[1] http://ejb3.jboss.org/.

[2] http://jbossas.jboss.org/downloads/.

[3] https://commons.wikimedia.org/wiki/File:JBoss_logo.svg.

[4] https://docs.oracle.com/javase/tutorial/jndi/ops/lookup.html.

[5] https://eclipse.org/.

[6] https://github.com/mickyzanotti/AdaptAnalyzer.

[7] https://it.wikipedia.org/wiki/Eclipse_(informatica).

[8] https://it.wikipedia.org/wiki/Enterprise_JavaBeans.

[9] https://it.wikipedia.org/wiki/Hibernate.

[10] https://it.wikipedia.org/wiki/ISO/IEC_9126.

[11] https://it.wikipedia.org/wiki/Java_Development_Kit.

[12] https://it.wikipedia.org/wiki/Java_Naming_and_Directory_Interface.

[13] https://it.wikipedia.org/wiki/Java_Runtime_Environment.

[14] https://it.wikipedia.org/wiki/MySQL.

[15] https://it.wikipedia.org/wiki/MySQL_Workbench.

[16] https://it.wikipedia.org/wiki/WildFly.

[17] https://scm.cedrc.cnr.it/wiki/index.php/Guida_minimale_EJB3_(Prima_Parte).

[18] http://theopentutorials.com/examples/java-ee/ejb3/how-to-create-ejb3-jpa-project-in-eclipse-jboss-as-7-1/.

[19] http://www.eclipse.org/downloads/packages/eclipse-classic-372/indigosr2.

[20] http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase6-419409.html/.

[21] Felix Maximilian Roth. Christian Krupitzer. A survey on engineering approaches for self-adaptive systems. *Elsevier*, (17):184–206, 2014.

[22] Subramanian N. Chung L. Process-oriented metrics for software architecture adaptability. *IEEE Computer Society*, pages 311–311, 2011.

[23] Raffaela Mirandola. Diego Perez-Palacin. On the relationships between QoS and software adaptability at the architectural level. *Elsevier*, (87):1–17, 2013.