UNIVERSITA' DEGLI STUDI DI L'AQUILA

FACOLTA' DI SCIENZE MM. FF. NN.

Corso di Laurea in Scienze dell'Informazione

TESI di LAUREA

PROGETTAZIONE E REALIZZAZIONE DI UN WEBSERVICE PER LA VALUTAZIONE DI MODELLI A RETI DI CODE BASATA SU TOOLS DIFFERENTI

RELATORE
prof. Vittorio Cortellessa

LAUREANDO
Samuel Zallocco

Anno Accademico 2005-2006

Ai miei Genitori e a Stefania

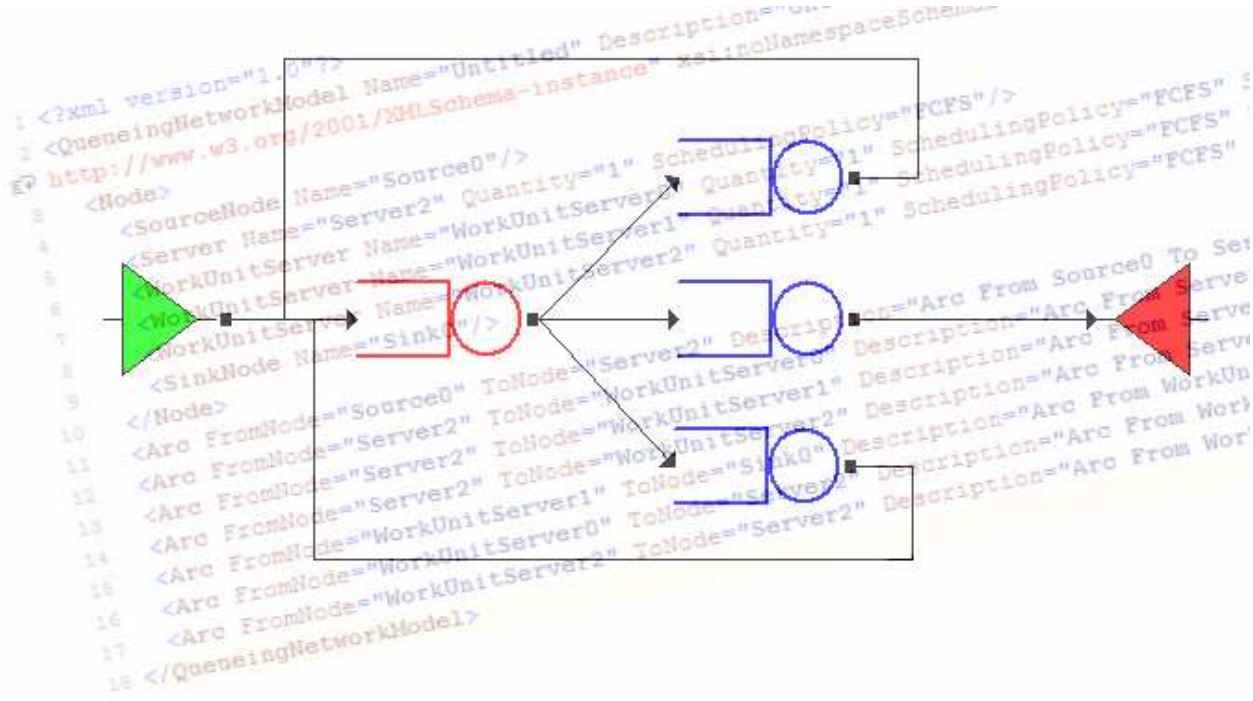
L'Aquila, li 25/03/2007 *Samuel Zallocco*

Indice

Keywords: Traffic problems, Queueing Theory, Networks models, Performance evaluation, Distributed systems, Web Service.

Mathematics Subject Classification: 90B20, 60K25, 68M20, 90B22, 90B10, 90B15.

ACM Classification: G.3, G. m, D.4.8, H.1.1, H.3.4, H.3.5, I.6, B.8, K.6.1, C.2.4, C.4



1 Introduzione

Oggetto della presente tesi è la realizzazione di un webservice per la valutazione delle prestazioni dei sistemi (*system performance evaluation*) a reti di code (*queueing networks*) descritti, o meglio modellati, nel linguaggio PMIF 2.0 (*Performance Model Interchange Format*), mediante l'utilizzo di alcuni tool, commerciali e di pubblico dominio, che implementano algoritmi tipici della teoria delle code (*queueing theory*).

Questa tesi vuole dimostrare come i webservice possano essere utilizzati per facilitare la condivisione di tool e algoritmi per l'analisi di modelli a reti di code tramite il web, senza la necessità di avere a disposizione materialmente in locale gli eseguibili dei tool.

I vantaggi di questo approccio sono numerosi ed evidenti, come la possibilità, da parte degli analisti che utilizzano tool di valutazione delle performance, di avere a disposizione un numero maggiore di algoritmi a costi più bassi. Potendo utilizzare una grande varietà di tool, avrebbero la possibilità di confrontare i risultati ottenuti e quindi di poter meglio valutare le prestazioni del sistema modellato. Alcuni tool, infatti, potrebbero analizzare in modo migliore di altri alcune caratteristiche del modello o mettere a disposizione funzioni o misure aggiuntive.

I ricercatori e le software house potrebbero mettere a disposizione i loro tool tramite webservice e far pagare solo il reale utilizzo. Questi avranno inoltre l'ulteriore vantaggio di poter modificare i tool e rendere disponibile sempre la versione più aggiornata dei loro prodotti senza i costi tipici del rilascio di nuove release (si eliminano gli intermediari, si eliminano i costi di spedizione e packaging). Inoltre, essendo i web service per loro natura indipendenti dalla piattaforma, si eliminerebbe la necessità di dover convertire il tool per poter essere eseguito su piattaforme HW e SW differenti.

Dopo una doverosa introduzione teorica su sistemi, modelli, metodologie di modellazione, teoria delle code, verrà descritto il linguaggio PMIF 2.0, i tools commerciali e di pubblico dominio adatti ad essere integrati nel webservice, e quindi verrà descritta la realizzazione del webservice oggetto di questa tesi. Il webservice renderà disponibili online una serie di funzioni, invocabili mediante il protocollo HTTP/SOAP, in grado di accettare come input una rete di code con i sui parametri tipici nel formato PMIF 2.0 e di restituire come risultato la valutazione delle performance del sistema modellato dalla rete di code stessa, effettuata invocando uno dei tool integrati.

2 Sistemi, Modelli e Processo di Modellazione

In questo capitolo daremo una introduzione alla teoria delle code, per una trattazione completa dell'argomento si rimanda ai testi fondamentali di Lazowska [8], Balsamo [7].

Definiamo un **sistema** (esistente o ipotetico) come un insieme di componenti (elementi, entità genericamente "oggetti") interdipendenti che collaborano per raggiungere un determinato obiettivo.
In particolare un sistema di elaborazione è un insieme di componenti hardware e software che permettono e facilitano l'elaborazione automatica delle informazioni eseguendo i programmi utente (software applicativo o job).

Il progetto ed il successivo sviluppo dei sistemi di elaborazione richiede l'analisi di diversi aspetti che includono [7]:

- lo studio della funzionalità,
- della correttezza,
- dell'affidabilità e della sicurezza,
- del costo ed economicità,
- delle prestazioni (*performance*).

Un sistema può essere valutato sia dal punto di vista qualitativo che quantitativo. L'analisi quantitativa si effettua definendo opportuni **indici di prestazione**, quali, ad esempio, l'utilizzazione delle risorse e i tempi di risposta forniti dal sistema, che descrivono l'efficienza dello svolgimento delle sue funzioni

La valutazione delle prestazioni di un sistema è quindi un aspetto fondamentale nel progetto, nello sviluppo, nella configurazione e messa a punto di un sistema di elaborazione e nel suo aggiornamento. Questo studio costituisce un'area di ricerca in informatica che ha portato alla definizione di metodologie e di strumenti per la creazione ed analisi di modelli dei sistemi (*modelling*) e allo sviluppo di tecniche di misurazione e caratterizzazione del carico. Inoltre, va detto che lo studio dei modelli e delle metodologie di valutazione delle performance trova applicazione, non solo nei sistemi di elaborazione, ma anche nei sistemi di comunicazione e telecomunicazione, così come nei sistemi di produzione e di traffico.

La crescente complessità dei moderni sistemi di elaborazione, la loro rapida evoluzione e il loro sempre più intenso ed indispensabile utilizzo in tutti gli ambiti di lavoro, ha fatto nascere l'esigenza di strumenti e tecniche che aiutino nella comprensione del comportamento di questi sistemi, e la necessità di dare delle risposte a questioni inerenti il costo e le prestazioni di un sistema durante tutto il suo ciclo di vita: durante la fase di progetto ed implementazione, durante la fase di dimensionamento, di acquisto, durante la fase di evoluzione ed upgrade – per far fronte alle aumentate esigenze e del **carico di lavoro** (*workload*). C'è quindi la necessità di dare delle risposte alle domande che da queste esigenze scaturiscono, e che possono portare a serie ripercussioni sul core business delle aziende coinvolte, nel caso di risposte errate.

Si possono affrontare queste questioni in modi differenti, ad esempio usando l'intuizione e l'esperienza, oppure la valutazione sperimentale delle alternative. Ad esempio la sperimentazione delle alternative è una opzione sempre valutabile, spesso necessaria e a volte risolutiva per la scelta del sistema. Tuttavia è una tecnica costosa, a volte proibitiva e non si presta a generalizzazioni. Questi due approcci si trovano in un certo senso su due estremi opposti. L'intuizione è rapida e flessibile, ma la sua accuratezza è sospetta, in quanto si basa sull'esperienza e sull'acume, due doti che sono difficili da acquisire e da verificare scientificamente. La sperimentazione conduce ad un'eccellente

accuratezza, ma è laboriosa e poco flessibile. Tra questi due approcci estremi, giace un terzo approccio che è la **modellazione** (*modelling*).

L'evoluzione di un sistema rispetto al tempo può essere descritta in ogni istante dallo stato del sistema. Lo **stato di un sistema** può essere descritto da un insieme di **variabili di stato**, che descrivono le entità del sistema e i loro attributi. Le **interazioni** tra le componenti nel tempo possono essere descritte mediante **regole di trasformazione** (o di passaggio) tra stati. L'insieme dei diversi stati di un sistema e la sua evoluzione è rappresenta dalla **storia degli stati**.

È ovvio come un sistema reale operi in un **ambiente** che può influenzare il comportamento del sistema stesso. Si pensi, ad esempio, ad un sistema che opera in condizioni di temperatura estrema. Come sappiamo la CPU, ma non solo, è un elemento di un sistema di elaborazione molto sensibile agli sbalzi di temperatura, e quindi adotta tecniche di protezione che possono portare anche al blocco temporaneo o definitivo del sistema o, come avviene ad esempio in sistemi portatili, all'abbassamento della frequenza di funzionamento della CPU stessa per ridurre il calore sviluppato, con conseguente impatto sulle performance dell'intero sistema.

C'è quindi la necessità di identificare senza ambiguità il sistema e la sua interfaccia verso l'ambiente esterno. Le variabili di stato si distinguono quindi in [7][10]:

- **variabili endogene**: variabili il cui valore dipende solo da attività interne al sistema;
- **variabili esogene**: variabili che sono influenzate dall'ambiente in cui opera il sistema.

Un sistema è detto:
- **chiuso**: quando il suo comportamento è indipendente dall'ambiente in cui opera (non ha variabili esogene);
- **aperto**: quando può essere influenzato dall'ambiente (presenta variabili esogene).

Una ulteriore classificazione dei sistemi può essere fatta in base al tipo di cambiamento dei valori delle variabili di stato:
- **sistemi continui**: le cui variabili variano in un insieme continuo (un tipico esempio è proprio la temperatura, i cui cambiamenti sono continui e graduali);
- **sistemi discreti**: le cui variabili di stato assumono valori solo in un set discreto di possibili valori (come ad esempio il numero di persone presenti in una stanza, il cui cambiamento avviene per passi discreti ed istantanei).

Per quanto riguarda il passaggio tra stati del sistema, si può effettuare una classificazione in:
- **sistemi deterministici**: in questa tipologia di sistemi, le regole di trasformazione determinano univocamente il cambiamento di stato del sistema (ad esempio una catena di montaggio è un tipico sistema deterministico in quanto in ogni istante lo stato di lavorazione corrente determina lo stato di lavorazione successivo di un prodotto);
- **sistemi stocastici**: in questa tipologia di sistemi, da uno stato è possibile raggiungere diversi altri stati secondo una legge di probabilità associata alla regola di trasformazione (ad esempio il numero di pazienti che si presentano in un pronto soccorso è un tipico esempio di sistema stocastico).

Ovviamente la natura stocastica o deterministica, continua o discreta di un sistema dipende fortemente dalla visione che ne viene data dall'osservatore e dalla sua esperienza così come dagli obiettivi perseguiti e dal metodo di studio adottato.

Le metodologie per la valutazione dei sistemi si possono distinguere in due categorie principali [7][8] e a loro volta suddivise in sotto categorie come schematizzato in Figura 2.1:

- le **tecniche di modellazione** calcolano degli indici applicando metodi e modelli analitici, oppure stimandoli mediante metodi e modelli simulativi,
- le **tecniche di misurazione** valutano gli indici di prestazione del sistema mediante l'utilizzo di opportune tecniche di misurazione diretta, benchmarking o prototipizzazione,

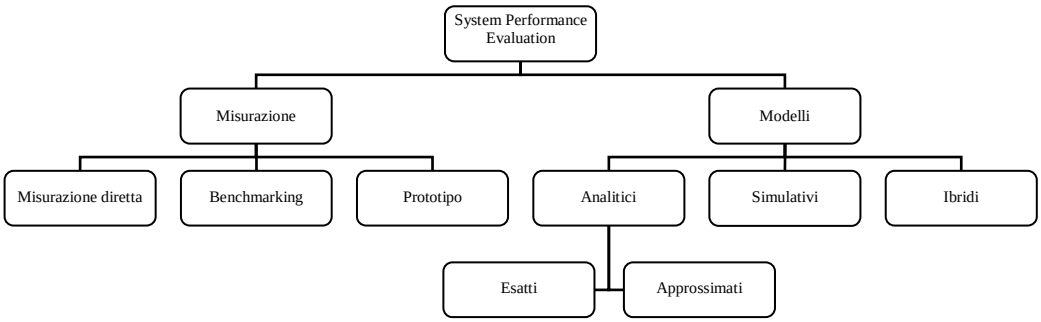


Figura 2.1 - System Performance Evaluation

I **modelli di simulazione**, che riproducono il comportamento dinamico nel tempo del sistema modellato, offrono uno strumento di analisi completo e versatile, rappresentando le componenti e le interazioni in termini di relazioni funzionali. Per valutare un modello simulativo si ricorre ad un programma di simulazione (o **simulatore**), che rappresenta l'evoluzione temporale del sistema e su cui si effettuano delle misure per stimare le grandezze di interesse. Tuttavia, oltre agli elevati costi di sviluppo ed implementazione, presentano alcuni limiti intrinseci. In particolare la fase di definizione e sviluppo di un modello di simulazione può avere un elevato costo che dipende dal livello di accuratezza del modello stesso. Inoltre la fase di analisi di un modello di simulazione può richiedere, per ottenere dei risultati sufficientemente accurati, tempi di elaborazione molto lunghi.

Nei **modelli analitici** le componenti del sistema oggetto di studio vengono rappresentate mediante l'utilizzo di variabili e parametri. Le interazioni fra le componenti del sistema vengono invece rappresentate da relazioni fra variabili e parametri. La soluzione o valutazione del sistema si effettua tramite metodi analitici e numerici. Una classe di modelli analitici nata inizialmente per lo studio dei sistemi telefonici è la teoria delle code (*queueing theory*), disciplina matematica nell'ambito della probabilità applicata [10]. I modelli analitici si suddividono in modelli esatti e modelli approssimati.

I **modelli analitici esatti**, che consentono la valutazione esatta degli indici di prestazione, si basano sullo studio dell'evoluzione di opportuni processi stocastici definiti a partire dalle caratteristiche del modello. Per una classe molto importante di modelli a rete, la classe delle reti Markoviane, il comportamento in condizioni di stazionarietà è analizzabile esattamente mediante lo studio del processo Markoviano associato alla rete. Gli indici di prestazione per le reti Markoviane si possono ricavare, sotto opportune ipotesi, dalla soluzione del sistema di equazioni di bilanciamento relative al processo. La dimensione di questo sistema lineare coincide con il numero di stati del processo Markoviano associato alla rete. Poiché il numero degli stati del processo Markoviano risulta dipendere esponenzialmente dai parametri della rete, la complessità, sia di tempo che di spazio, di questo tipo di analisi risulta talmente elevata da limitarne notevolmente l'applicabilità.
Per una sottoclasse delle reti Markoviane, le reti in forma-prodotto, sono state derivate, partendo principalmente dalle proprietà del processo Markoviano associato, delle forme chiuse, dette appunto forme prodotto, per la soluzione del sistema di equazioni di bilanciamento. Per i modelli di reti di code con blocco in forma-prodotto, che tuttavia costituiscono una ristretta classe, sono stati proposti

in letteratura efficienti algoritmi per la valutazione di un insieme di indici di prestazione la cui complessità computazionale è polinomiale nel numero di componenti del modello.
I **modelli analitici approssimati** consentono una valutazione approssimata degli indici di prestazione ad un costo, sia di spazio che di tempo, relativamente basso. I principi su cui si basano i metodi approssimati sono molto diversi tra loro. Alcuni metodi, ad esempio, si basano sull'analisi di un modello semplificato rispetto all'originale, altri metodi invece sfruttano delle proprietà particolari di alcune classi di modelli. L'interesse verso i modelli analitici approssimati è giustificato principalmente dalla difficoltà di generare risultati esatti, mediante i modelli analitici esatti, o stime con un'accuratezza controllabile, ottenibili con modelli di simulazione. La credibilità dei risultati forniti dai modelli approssimati deve essere tuttavia completata mediante un'analisi sperimentale che, oltre a determinare il livello di accuratezza, è utile per individuare delle proprietà tipiche del modelli che consentono di definire i migliori casi di applicabilità per il modello stesso.
I **modelli ibridi** utilizzano sia tecniche analitiche che simulative per lo studio del sistema. Tali modelli sono particolarmente adatti alla modellazione di sistemi di grandi dimensioni, che possono essere scomposti in sottosistemi più piccoli, alcuni modellati con tecniche analitiche ed altri con tecniche simulative. Il vantaggio è di sfruttare la potenza delle due classi di modelli: la semplicità e l'efficienza risolutiva dei modelli analitici, e la generalità della simulazione. Tuttavia la modellazione combinata analitico-simulativa usata nei modelli ibridi viene vista come un approccio empirico per l'analisi di sistemi complessi e non come una vera e propria metodologia.

Nella **misurazione diretta**, il sistema reale viene valutato in modo diretto, con opportune tecniche e strumenti, andando a misurare gli indici sul carico reale.
Il **benchmarking** è un tipo di misurazione con carico artificiale. Con questa tecnica vengono effettuate delle misurazioni sulla piattaforma reale ma con carico artificiale. Ad esempio rientrano in questa tipologia di misurazione i famosi benchmark [9] tipo le suite di test della serie SPEC [51]: SPEC CPU2000, SPECFS, SPECWeb, quelle della serie TPC [52]: TPC-C, TPC-R, TPC-H, TPC-W, quella della EEMBC [53], quella di Web Polygraph [54], e infine le famose: 3DMARK, SYSMARK e simili, questi ultimi molto in voga sulle riviste di computer per stabilire un indice di prestazioni di un sistema rispetto ad una suite di test prestabilita e ad un sistema base di riferimento. Il vantaggio del benchmarking è quello della ripetibilità. Nel caso della misurazione diretta invece, a causa della già citata complessità dei sistemi di elaborazione moderni, è difficile stabilire se le condizioni del sistema sono cambiate o meno tra un test e l'altro.
Entrambe le tecniche citate sopra richiedono un requisito fondamentale, ovvero la disponibilità della piattaforma reale su cui eseguire la misurazione o il benchmark. In assenza di essa, perché non ancora sviluppata o se troppo costosa da acquisire, si deve ricorrere ad un prototipo.
Il **prototipo** può essere realizzato via software, e si parla in questo caso di emulatore di sistema. Il principale svantaggio di questa tecnica è la sua scarsa modificabilità e flessibilità.

Ecco, quindi, che l'uso di modelli per la valutazione e lo studio dei sistemi non esistenti diventa indispensabile. A differenza dei prototipi, i modelli non scendono mai ad un livello di dettaglio troppo accurato, rendono quindi più semplice la scelta tra differenti alternative, e risultano più economici dei prototipi.

Un modello può essere descritto a differenti **livelli di astrazione**, includendo solo le componenti e le interazioni tra essi, ritenute interessanti per lo studio da effettuare. Una volta definito un modello, attraverso il già citato processo di astrazione, esso può essere **parametrizzato** per riflettere le differenti alternative del caso di studio, poi può essere valutato per studiare il comportamento del sistema rispetto alle varie alternative.

L'impiego di un modello presenta vantaggi e svantaggi rispetto alla misurazione diretta. Tra i vantaggi abbiamo che un modello [7]:

- facilita l'analisi del sistema;
- porta ad una maggiore comprensione del sistema modellato, come conseguenza del dover identificare le componenti funzionali e le interazioni tra esse;
- è più facilmente modificabile rispetto al sistema fisico;
- consente lo studio del sistema a diversi livelli di astrazione e dettaglio.

Tra gli svantaggi abbiamo che:

- una errata scelta del livello di astrazione può portare ad errori di valutazione;
- esiste il rischio di un uso errato del modello al di fuori delle assunzioni che ne hanno determinato la scelta ovvero oltre il suo ambito di applicabilità.

Un **sistema di congestione** è un sistema nel quale un determinato numero di utenti cerca di accedere ad un numero limitato di risorse. Tipici esempi di sistemi di congestione sono i sistemi di calcolo, i sistemi di comunicazione, i sistemi di traffico, i sistemi di produzione. Tutti questi sistemi presentano la caratteristica di un numero limitato di risorse (di calcolo, stampa, trasmissione, ricezione, portata) a cui un certo numero di utenti (processi, job, pacchetti, automobili, aerei, passeggeri, merci) cercano di accedere in modo più o meno ordinato.
Per valutare le prestazioni di un sistema di congestione, si devono prendere in considerazione aspetti quali: l'utilizzazione delle risorse (lato sistemistico) e i tempi di attesa (lato utente).
L'ottimizzazione di un sistema di congestione può quindi procedere per strade a volte contrastanti: *massimizzazione dell'uso delle risorse* vs la *minimizzazione dei tempi di attesa* che un utente deve aspettare per l'uso della risorsa stessa. È ovvio come aumentando il numero di risorse a disposizione, si minimizzi il tempo di attesa per il loro utilizzo (a scapito di un maggior costo del sistema), così come diminuendo il numero di risorse (o lasciandolo inalterato all'aumentare degli utenti), aumenti il carico di lavoro per singola risorsa e quindi se ne massimizza l'utilizzo.

Nel **processo di modellazione** va quindi perseguito l'obiettivo di trovare il giusto compromesso tra i due estremi, mediante l'utilizzo degli strumenti matematici (reti di code e processi stocastici) ed anche mediante una buona dose di esperienza.
Condurre uno **studio di modellazione** (*modelling study*) vuol dire cercare di identificare le componenti principali di un sistema e il modo in cui esse interagiscono tra loro, fornendo ogni dettaglio che si dimostri utile.
Questo processo di scrematura dei dettagli irrilevanti comporta per forza di cose l'adozione di un certo numero di **assunzioni** che dovranno essere valutate e ben documentate durante il processo di definizione del modello.
Ci sono tre principali motivazioni che guidano la scelta delle assunzioni effettuate in fase di definizione del modello [7][8]:

- *Semplicità del modello*: Come incentivo ad eliminare dettagli irrilevanti del sistema, dove per irrilevante si intende ogni caratteristica che non ha un effetto primario (*primary effect*) sul risultato dello studio. Spesso un modello semplice riesce a fornire risposte sufficientemente accurate, specialmente nella fase di progettazione.
- *Adeguatezza delle misurazioni*: Gli strumenti di misurazione oggi disponibili per default nei moderni sistemi di elaborazione e sistemi operativi offrono, per ragioni storiche, un gran numero di dati, la maggior parte dei quali è di scarsa utilità nella fase di parametrizzazione del modello e di definizione delle variabili caratteristiche del sistema oggetto di studio. Ad esempio nel caso delle reti di code, esse necessitano di un piccolo numero di input attentamente selezionati, mentre i tool di misurazione disponibili nei moderni sistemi di elaborazione non forniscono questi input in modo diretto ed adatto alla parametrizzazione di un modello a reti di code.
- *Semplicità delle valutazioni*: Non tutti i modelli sono valutabili in modo efficiente. Questo comporta il dover scendere a compromessi nella rappresentazione di alcune complesse caratteristiche di un moderno sistema di elaborazione, al fine di aumentare l'efficienza della

valutazione del modello adottato. Ad esempio nei modelli a reti di code non si utilizza la teoria generale delle reti di code perché tali modelli sarebbero difficilmente e poco efficientemente risolvibili, ci si deve quindi per forza auto-limitare ad un subset di tale teoria.

Ovviamente queste assunzioni sono facilitate dall'esperienza e sono la chiave per una fruttuosa conduzione di un studio di modellazione.
In generale, è molto importante essere chiari nelle motivazioni che conducono alla scelta di determinate assunzioni, alle motivazioni che hanno portato alla loro introduzione e le argomentazioni per la loro credibilità. Questa esplicita motivazione delle assunzioni effettuate aiuta nella valutazione della sensitività dei risultati ottenuti.

Analogamente ai sistemi anche i modelli possono venire classificati in aperti e chiusi, continui e discreti, deterministici e stocastici. Tuttavia non è detto che un sistema continuo debba per forza di cose essere rappresentato da un modello continuo ma, ad esempio, può essere rappresentato da un modello discreto, attuando una discretizzazione del dominio di definizione delle variabili del sistema, per definire le variabili discrete del modello. Analogamente anche un sistema stocastico può essere rappresentato da un modello deterministico facendo ad esempio riferimento solo ai valori medi delle variabili del sistema.

Il livello di astrazione scelto per rappresentare il sistema influenza la natura del modello, così come quest'ultima è influenzata dagli obiettivi per i quali il modello è stato definito. Infatti il modello deve rappresentare le proprietà elementari delle componenti del sistema e le loro interazioni da cui dipendono le funzionalità, oggetto dello studio, che si vogliono rappresentare e valutare.

Il processo di creazione ed utilizzo di un modello (*modelling cycle*) passa attraverso varie fasi [7][8]:

1. *Definizione degli obiettivi*: Rientra in questa fase la comprensione e la definizione del sistema e delle sue componenti, dei suoi attributi, delle sue attività, delle interazioni e delle relazioni esistenti tra il sistema e l'ambiente esterno e, nel caso di sistemi già esistenti, l'acquisizione dei dati dal sistema e la misurazione del carico tramite opportune tecniche. Il risultato di questa fase di studio sarà l'identificazione delle variabili da valutare e quindi degli indici di prestazione. Contestualmente si stabiliranno i criteri per la valutazione dei risultati ottenuti dal modello.
2. *Definizione del modello e formulazione delle assunzioni e ipotesi*: Fissato un livello di astrazione opportuno, utilizzando lo studio effettuato nel punto precedente, si formalizzerà il modello. In questa fase tutte le componenti del sistema e le relazioni tra esse, gli attributi, le caratteristiche funzionali, le assunzioni e le ipotesi sono ben identificate ed espresse nel formalismo utilizzato per la modellazione. Proprio dalla formalizzazione e definizione del sistema deriverà la complessità del modello e la sua risolubilità.
3. *Parametrizzazione*: In questa fase vengono identificate le variabili del modello da misurare e i relativi strumenti di misurazione. Viene anche effettuata una caratterizzazione del carico di lavoro ovvero il modello che rappresenta il carico di lavoro del sistema e le tecniche per definirlo .
4. *Valutazione del modello (soluzione) e interpretazione dei risultati*: Una volta definito e parametrizzato il modello, si sceglie il metodo di soluzione più appropriato, considerando fattori di complessità computazionale e accuratezza dei risultati.
5. *Validazione del modello e valutazione dei risultati*: questa fase consiste nella valutazione della adeguatezza del modello e nella convalida dei risultati ottenuti e quindi, da un confronto con i requisiti stabiliti al punto 1, la capacità del modello di descrivere il sistema in modo adeguato. Se il modello dovesse risultare inadeguato a descrivere il sistema, allora si itera al passo 1, se si reputa che il problema sia nella definizione del sistema e degli obiettivi. Altrimenti si itera al passo 2 per modificare la definizione del modello, delle ipotesi e delle assunzioni effettuate. Se, pur essendo il modello ritenuto accettabile, i risultati non lo sono, allora si itera al passo 3

per ridefinire la parametrizzazione del modello. Quindi si conclude il procedimento, possibilmente con il passo successivo.

6. *Documentazione e analisi di sensitività*: Questa fase, se pur non essenziale per uno studio "personale" di un sistema, risulta indispensabile nel caso i risultati debbano essere diffusi e commentati da chi ha commissionato lo studio. La documentazione del modello e dei risultati deve includere sia i dettagli relativi al modello finale, le assunzioni, i metodi di soluzione scelti, gli eventuali risultati sperimentali, una analisi di validità, il costo, le conclusioni a cui si è giunti e le raccomandazioni, sia una descrizione del processo di definizione del modello. L'analisi della sensitività è un aspetto che non dovrebbe mancare in ogni documentazione di uno studio di un modello. L'analisi di sensitività consiste nello studio degli effetti che le assunzioni utilizzate hanno sui risultati ottenuti. Si hanno due tipici approcci per effettuare l'analisi di sensitività che sono [8]:
   - *Analisi di robustezza*: in questo caso il modello viene valutato più volte variando le ipotesi e le assunzioni per verificare l'incidenza sui risultati ottenuti;
   - *Studio dei casi limite*: in questo caso vengono identificati i valori limite e i casi estremi di applicazione del modello. Quindi il modello viene valutato sotto queste assunzioni ed ipotesi limite, i risultati ottenuti sono considerati dei limiti per gli indici del modello, i quali definiscono implicitamente dei limiti di applicabilità del modello (*bounds*), che può permettersi di valutare un sistema in domini non disponibili nella realtà.
7. Un possibile motivo per la mancata risolubilità di un modello potrebbe essere la presenza nel sistema di **colli di bottiglia** (*bottleneck*) o strozzature dovute ad un eccessivo utilizzo di una o più risorse. Nel processo di modellazione si dovrebbe quindi prevedere una ulteriore settima fase, di rimozione dei colli di bottiglia dal sistema o dal modello; il processo andrebbe iterato più volte in quanto la rimozione di un bottleneck potrebbe farne insorgere altri prima mascherati dal precedente.

Come linea guida generale, durante il processo di modellazione, si dovrebbero tenere sempre bene in mente non solo aspetti relativi alla significatività dei risultati, ma anche aspetti relativi alla complessità risolutiva del modello e alla sua semplicità di utilizzo.

3 Modelli a Reti di code

La teoria delle code (o file d'attesa) studia i fenomeni di attesa che si possono verificare quando si richiede un servizio. I suoi campi di applicazione, oltre ad essere quelli legati alla vita quotidiana, riguardano anche i sistemi di elaborazione, i sistemi di trasmissione dati, i sistemi flessibili di lavorazione, i sistemi di trasporto, di traffico e molti altri. Quindi oltre che migliorare la qualità della vita delle persone, la teoria delle code trova applicazione nel settore industriale, dei servizi e in moltissimi altri. Dal punto di vista storico, la formulazione del primo problema di teoria delle code risale al 1908, quando A. K. Erlang, un ingegnere impiegato presso la Danish Telephone Company di Copenaghen, ritenuto il fondatore della teoria delle code, voleva studiare come dimensionare centrali telefoniche allo scopo di mantenere ad un valore ragionevolmente basso il numero delle chiamate che non potevano essere connesse (chiamate perse) perché il centralino era occupato. Negli ultimi anni sono state proposte trattazioni sistematiche dell'applicazione di tale teoria e in particolare dei modelli a reti di code (*queueing networks*) che ben si adattano alla modellazione di sistemi di elaborazione, comunicazione, produzione e traffico.

Un semplice modello a coda singola è costituito da una rappresentazione di un **sistema di congestione** in cui gli utenti/clienti (*user*/*client o job*) che provengono da una determinata popolazione si accodano (secondo un processo di arrivo che segue una determinata distribuzione di probabilità) per ottenere un servizio da un insieme di risorse/serventi (*server*), ottenuto il quale lasciano il sistema e ritornano a far parte della popolazione Figura 3.1. L'insieme formato da coda e serventi è detto **centro di servizio**.

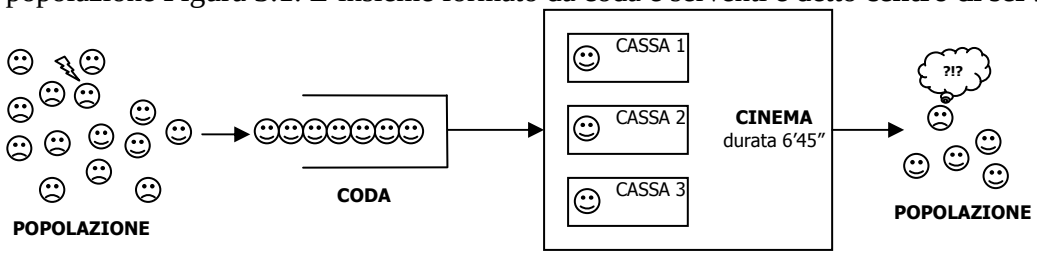


Figura 3.1 - Esempio di sistema di congestione

Per **modello a reti di code** di un sistema di calcolo, si intende una sua rappresentazione sotto forma di una rete di code che possa essere valutata in maniera analitica. Una **rete di code** è un insieme di **centri di servizio** (che rappresentano le risorse del sistema) e di **clienti** (che rappresentano gli utenti o le transazioni). I clienti (o *job*) che hanno il medesimo comportamento dal punto di vista delle esigenze di servizio e di cammino (*routing*) sulla rete di code possono essere raggruppati in **classi** (*class*) o **catene** (*chain*). Se nella rete di code circola una sola classe di clienti si parla di rete *single-class* o *single-chain*, altrimenti, nel caso di più classi di clienti, si parla di rete *multiple-class* o *multiple-chain*. Alcuni autori non usano class e chain come sinonimi, ma con il termine chain indicano un insieme di classi i cui clienti possono passare da una classe all'altra (*class switching*).
In genere, una volta definita una rete di code si usano appositi algoritmi per risolvere in maniera efficiente l'insieme di equazioni "indotto" dalla rete e dai suoi parametri.

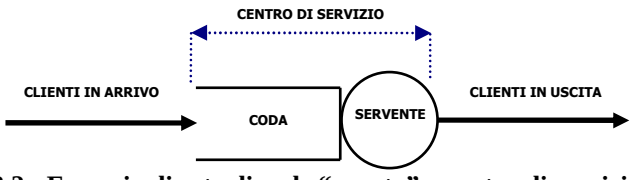


Figura 3.2 - Esempio di rete di code "aperto" a centro di servizio singolo

Osservando la Figura 3.2, si "intuisce" come per poter valutare un sistema a coda di questo tipo bisogna specificare due parametri [8]:

- **l'intensità del carico** (*workload intensity*): cioè la frequenza di arrivo dei clienti e
- **la domanda di servizio** (*service demand*): cioè il tempo medio di servizio richiesto dai clienti.

Le prestazioni del sistema modellato dalla rete di code saranno basate sui seguenti indici:

- **utilizzo** (*utilization*): percentuale di tempo in cui il centro di servizio è occupato;
- **tempo di residenza** (*residence time*): tempo medio passato da un cliente nel centro di servizio, comprensivo sia del tempo di attesa in coda che del tempo di servizio effettivo;
- **lunghezza della coda** (*queue length*): numero medio di clienti presenti e clienti che stanno usufruendo del servizio;
- **traffico in uscita** (*throughput*): frequenza di attraversamento del centro di servizio.

Andamenti tipici [8][11] di questi indici sono mostrati nella Figura 3.3:

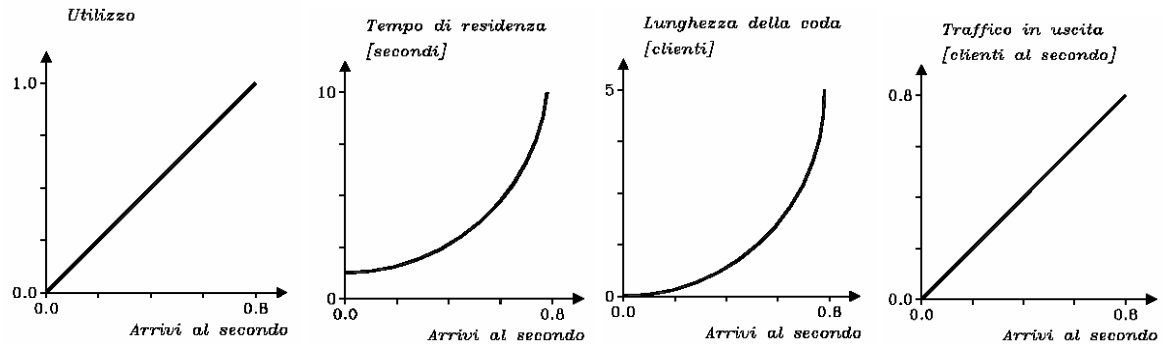


Figura 3.3 - Utilizzo, Tempo di Residenza, Lunghezza della Coda e Traffico in uscita

Questo modello a centro di servizio singolo, sarebbe di per se già sufficiente a descrivere e studiare il comportamento di un sistema nel suo complesso. Tuttavia, raramente nel mondo reale siamo interessati a casi così banali e generici. Per questo, durante la fase di studio e di modellazione, si procede in un ottica top-down. Prima si modella un sistema nel suo complesso, poi si raffina il procedimento includendo dettagli prima tralasciati al fine di meglio analizzare le componenti e le loro interazioni.

Ecco quindi che vengono definiti diversi livelli di astrazione secondo i quali si può condurre uno studio di modellazione ed analisi delle performance. Al livello di astrazione n ($n \geq 1$) è ovviamente associato un insieme di obiettivi e l'insieme dei risultati ottenibili a quel livello. In uno sviluppo top-down (gerarchico) è definita una relazione fra i modelli ai diversi livelli di astrazione; spesso gli obiettivi del livello n sono un sottoinsieme degli obiettivi del livello n+1, così come il livello n+1 è una estensione o uno sviluppo più dettagliato del modello al livello n.

Per definire il modello di un sistema complesso è necessario ricorrere a più centri di servizio collegati tra loro a formare una vera e propria rete, come illustrato per esempio nella Figura 3.4 (in particolare questa rete può essere utilizzata per rappresentare un sistema con più terminali e con carichi di tipo interattivo).

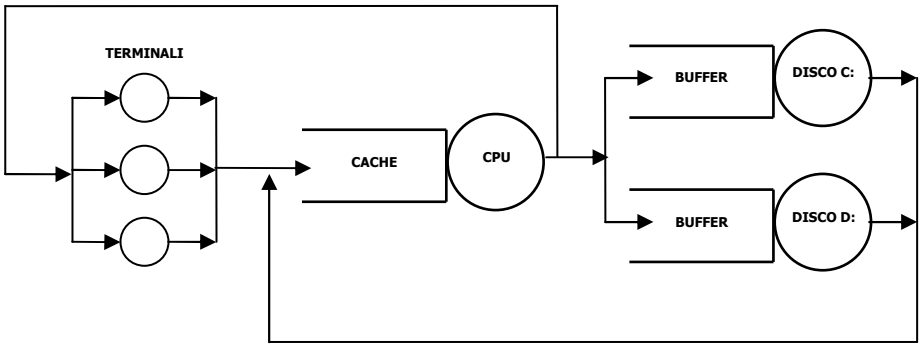


Figura 3.4 - Esempio di rete di code "chiusa" con più centri di servizio

In pratica a ogni risorsa del sistema di calcolo viene associato un centro di servizio per cui vengono specificati i parametri indicati prima, ovvero l'intensità del carico e la domanda di servizio.

In un generico modello a reti di code, i clienti rappresentano le transazioni che vengono effettuate nel sistema di calcolo, quindi l'intensità di carico corrisponderà alla frequenza con cui gli utenti mandano transazioni al sistema, mentre la domanda di servizio a un centro, sarà uguale alla richiesta complessiva di servizio da parte di una transazione nei confronti di una risorsa.

Osservando dall'esterno un sistema di congestione, come ad esempio quello modellato in Figura 3.5, si possono riscontrare alcune grandezze caratteristiche quali [7]:

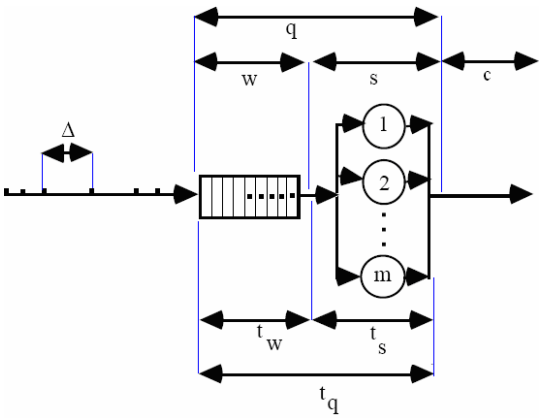


Figura 3.5 - Esempio di modello a coda singola con le sue grandezze tipiche

- Δ è il tempo che intercorre tra l'arrivo di un cliente ed il successivo;
- w è il numero di utenti in coda ($0 \leq w < q$);
- t_w è il tempo di attesa in coda, cioè il tempo che intercorre fra l'arrivo di un cliente in coda e l'istante in cui esso inizia ad essere servito;
- s è il numero di utenti attualmente serviti dagli m serventi ($0 \leq s \leq m$);
- m è il numero di serventi;
- t_s è il tempo di servizio, ossia il tempo che intercorre tra quando un cliente inizia ad essere servito e il completamento del servizio;
- q è il numero di utenti attualmente presenti nel centro di servizio ($q = w + s$);
- t_q è il tempo di risposta, ovvero il tempo che impiega un utente da quando entra nel centro di servizio a quando esce ($t_q = t_w + t_s$);
- c è il numero di utenti che hanno completato il servizio e lasciano il centro;

Tutte queste variabili, sono **variabili aleatorie** caratterizzate da una **distribuzione di probabilità**. Δ, m e t_s sono parametri del sistema, mentre le altre sono oggetto di analisi e valutazione.

Oltre a Δ, m e t_s, per poter caratterizzare in modo completo il comportamento di un sistema di congestione mediante una rete di code, bisogna definire altre grandezze [7][9]:

- il processo di arrivo dei clienti;
- la capacità della coda;
- la disciplina di servizio;
- il processo di servizio.

Il **processo di arrivo dei clienti** caratterizza il modo in cui i clienti arrivano nel centro di servizio e si mettono in coda. In generale, il processo di arrivo può essere rappresentato da una sequenza di tempi $a_1,a_2,\ldots,a_n$ (*arrival*) dove a_n è il tempo d'ingresso dell'*n*-esimo cliente nel sistema. Oppure, alternativamente, il processo di arrivo può essere descritto dalla sequenza dei tempi di interarrivo $i_1,i_2,\ldots,i_n$ (*interarrival*) dove i_n è il tempo che intercorre tra l'arrivo del *n*-esimo e l'*n+1*-esimo cliente. È chiaro quindi che i tempi di interarrivo Δ sono variabili casuali statisticamente indipendenti con la stessa distribuzione di probabilità. Nei modelli in cui vi sono differenti tipologie di clienti (classi), ad ogni tipologia si associa un processo di arrivo.

La **capacità della coda** è il numero massimo di clienti che possono essere presenti simultaneamente nel sistema (comprensivo cioè degli utenti in attesa e degli utenti che stanno ricevendo il servizio). Si possono distinguere due casi:

- code di capacità infinita (*infinite capacity queues*)
- code di capacità finita (*finite capacity queues*)

Nella prima tipologia di code, ogni cliente che arriva nel sistema viene sempre accettato in quanto il sistema dispone di una "sala d'attesa" infinita. Nel secondo caso, invece, nel processo di modellazione si deve specificare il comportamento del sistema quando la coda è piena. Generalmente, si assume che i clienti continuino ad arrivare, ma quando la coda è piena il cliente in eccesso viene perso. Alternativamente si potrebbe assumere di fermare il processo di arrivo, per farlo riprendere non appena si libera spazio.

La **disciplina di servizio** (o regola di *scheduling*) definisce la politica di assegnazione dei serventi ai clienti in coda, ovvero l'ordine con cui estrarre i clienti dalla coda. La disciplina di base usualmente utilizzata è la così detta fist-in-first-out (FIFO), altrimenti conosciuta come first-come-first-served (FCFS), in pratica il primo ad arrivare sarà anche il primo ad essere servito. Questa disciplina di servizio potrebbe sembrare normale, ma tuttavia esistono diverse motivazioni che possono portare ad una scelta diversa, per esempio:

- ordine di arrivo in coda;
- priorità dei clienti in coda;
- quantità di servizio di cui un utente ha già usufruito.

Si pensi ad esempio ai processi di un sistema di elaborazione, è ovvio che un processo del sistema operativo che gestisce un componente real-time debba essere servito con una priorità maggiore rispetto ad esempio ad uno spool di stampa.
Esistono altre discipline di servizio, tra cui [7][10]:

- LIFO/LCFS/LCFSPR: Last In First Out/Last Come First Served/ LCFS Priority Resume;
- ROUND ROBIN;
- PROCESSOR SHARING;
- RAND.

La disciplina RAND determina il cliente da estrarre dalla coda e servire in modo casuale, secondo una distribuzione uniforme discreta.
Una caratteristica di alcune discipline di servizio è la **prelazione** ovvero il prerilascio delle risorse occupate. In tal caso è ammesso che il cliente attualmente in servizio venga interrotto da un cliente selezionato dalla coda perché ad esempio a priorità maggiore, o perché ha già occupato per un predeterminato tempo il servente. Al termine del servizio l'utente può riprendere da dove era stato interrotto, se l'interruzione era senza perdita, oppure ricominciare dal principio.
Nelle discipline che dipendono dalla quantità di tempo assegnato ad un cliente, come ROUND ROBIN o PROCESSOR SHARING, si assegna ad un cliente un "quanto" di tempo per il servizio, terminato il quale esso viene rimesso in coda in attesa del prossimo quanto. Questo è il caso del ROUND ROBIN, e se si fa tendere il quanto a zero, detto q il numero di clienti, ognuno riceve servizio con velocità pari a $1/q$ del tasso di servizio:

$$\lim_{q\to\infty} \frac{1}{q} t_S = 0$$

e si ha quindi l'impressione che ogni cliente sia servito quasi in tempo reale, anche se per brevi periodi (PROCESSOR SHARING).
Le discipline a priorità determinano l'ordine di servizio dei clienti sulla base di una priorità assegnata ai clienti secondo un determinato criterio. Le priorità possono essere assegnate in modo statico o dinamico. Possono essere basate su informazioni non dipendenti dalle richieste di servizio (astratte) oppure basate sulla domanda di servizio.
Anche le discipline a priorità possono essere con prelazione, quando il servizio è interrompibile, oppure senza prelazione se il servizio non lo è. Esistono anche forme miste di disciplina di servizio, ad esempio si può avere il caso di una disciplina di servizio a priorità, ma i clienti con pari priorità vengono serviti ad esempio con disciplina di servizio FIFO.
È da notare che, eccetto per i sistemi a singolo servente con disciplina di servizio FIFO, non è detto che i clienti lascino il centro di servizio nell'ordine in cui sono arrivati, infatti un cliente potrebbe necessitare di meno tempo presso il servente per compiere la sua operazione rispetto ad uno arrivato prima.

Il **numero di serventi**, può essere 1 (*sigle-server case*), un numero finito $m \geq 1$ (*multiple-server case*), oppure può essere infinito ∞ (*infinite-server o delay-server case*). Quest'ultima situazione avviene quando c'è sempre almeno un servente libero per servire un cliente che arriva nel sistema, ovvero c'è un numero infinito di risorse di servizio. Si parla quindi di *delay-server case*, in quanto il tempo di permanenza nel centro di servizio dipende solo dal tempo di servizio e non dalla lunghezza della coda. Generalmente si assume che tutti i serventi siano uguali, ma potrebbe capitare il caso di serventi con tempi di servizio diversi.

Il **processo di servizio** caratterizza le richieste dei clienti in termini di servizio. In generale, il processo di servizio può venire rappresentato da una sequenza di tempi $s_1, s_2, \ldots, s_n$ dove s_n è il tempo necessario per servire n-esimo cliente, e prende il nome di tempo di servizio (*service time* = t_s).
Le s_n sono variabili aleatorie e nel caso più generale esse possono avere distribuzioni di probabilità differenti. Tuttavia, nel caso più semplice e di più ampio interesse, queste variabili aleatorie sono indipendenti ed identicamente distribuite, ed anche indipendenti dalla sequenza dei tempi di interarrivo. In quest'ultimo caso, il processo di servizio è completamente caratterizzato dalla distribuzione di queste variabili, che prende il nome di *service time distribution*.
Se, invece arrivi e tempi di servizio hanno una distribuzione continua, allora si parla di *continuous time distribution* e il modello associato è un *continuous time model*. Un altro caso è quello in cui arrivi e servizi non possono avvenire in qualsiasi momento ma solo in tempi prestabiliti, multipli di una data quantità (*time unit*). In questo caso, il modello si dice *discrete time model* e le distribuzioni degli intervalli di arrivo e servizio sono detti *discrete time distribution*.

Per descrivere e definire i modelli a coda singola Kendall ha introdotto la seguente notazione, detta appunto **notazione di Kendall**:

A/B/c/n/p-Z o **A/B/c/n/p/Z**

Dove:

- **A**: denota la distribuzione di probabilità relativa al processo di arrivo dei clienti**;**
- **B**: denota la distribuzione di probabilità relativa al processo di servizio del/dei servitore/i;
- **c**: numero massimo di serventi di cui dispone il centro di servizio ed è un intero positivo;
- **n**: dimensione della coda, ovvero la capacità del centro ed è un intero non negativo;
- **p**: dimensione della popolazione dell'intero sistema, se $p=\infty$ allora può essere omessa;
- **Z**: è opzionale e denota la disciplina di servizio, se omessa si assume per default la disciplina FIFO.

Se $p = \infty$ e Z = FIFO, allora la notazione di Kendall si semplifica in A/B/c/n o A/B/c se anche $n=\infty$.

Per quanto riguarda le distribuzioni di A e B, abbiamo che esse potrebbero appartenere a qualunque classe di distribuzione, ma tuttavia nei modelli analitici risolvibili efficientemente, le distribuzioni utilizzate appartengono a particolari casi. Questo perché per le distribuzioni generali abbiamo pochi risultati teorici che ne consentono una efficiente risolubilità analitica, ovvero pochi metodi che consentono un'analisi esatta del modello.

Le principali distribuzioni utilizzate per A e B sono [9]:

M: (*Markov*) si tratta della distribuzione esponenziale negativa, detta anche distribuzione Markoviana per la sua proprietà di assenza di memoria. In base a questa proprietà abbiamo quindi che la probabilità che un cliente completi il proprio ciclo di servizio in un istante *t* è indipendente dal tempo di servizio già utilizzato dal cliente.

$$\Omega(t) = 1 - e^{-\lambda t} \quad e \quad \omega(t) = \lambda e^{-\lambda t} \quad con \quad t \geq 0 \; e \; \lambda > 0 \in R^{+}$$

L'unica cosa che ci serve per descrivere questa distribuzione è il parametro λ.

D: (*Deterministic*) tutti i valori della "distribuzione deterministica" sono gli stessi, si ha quindi una distribuzione costante (*Degenere*);

E_k: (*Erlang-k*) distribuzione Erlangiana a *k* fasi ($k \geq 1$). Per questa distribuzione abbiamo:

$$\Omega(t) = 1 - e^{-k\mu t} \sum_{j=0}^{k-1} \frac{(k\mu t)^j}{j!} \quad \text{dove } \mu > 0 \text{ è un parametro}$$

Questa distribuzione è molto popolare per la modellazione delle chiamate in arrivo ad una centrale telefonica.

H_k: (*Hyper-k*) distribuzione iperesponenziale con k fasi ($k \geq 1$). Per questa distribuzione abbiamo:

$$\Omega(t) = \sum_{j=1}^{k} q_j (1 - e^{-\mu_j t})$$

dove $\mu_i > 0$, $q_i > 0$, $i \in \{1 \ldots k\}$ sono parametri tali che $\sum_{j=1}^{k} q_j = 1$.

G: (*General*) la distribuzione generica, non altrimenti specificata. In molti casi si conosce almeno la media e la varianza.

GI: (*General Independent*) alcuni autori scrivono GI invece di G per enfatizzare il fatto che i tempi di interarrivo devono essere indipendenti identicamente distribuiti (iid), e riservano invece la lettera G al

caso ancora più generale dove tutti i tempi di interarrivo hanno la stessa distribuzione, ma non necessariamente sono indipendenti.

Ecco così che, ad esempio, con M/M/1/∞/∞-FIFO o M/M/1/∞/∞-FCFS si descrive un sistema a coda, con distribuzione dei tempi di interarrivo e di servizio di tipo Markoviano (ma con parametro che può essere diverso), singolo servente, dimensione della coda infinita, popolazione infinita e disciplina di servizio FIFO/FCFS. Come già detto, nella notazione di Kendall, tale sistema può essere abbreviato nella forma compatta M/M/1.

Lo stato di un sistema dinamico in un dato istante temporale rappresenta l'insieme informativo minimo che permette di conoscere l'evoluzione futura del sistema stesso, una volta note le relazioni dei fenomeni stocastici cui è soggetto. Lo stato di una coda è dato dal numero di clienti presenti nel sistema, dal tempo trascorso dall'arrivo dell'ultimo cliente. Infine, per ogni servitore, da un valore binario indicante se sta correntemente fornendo un servizio ed in questo ultimo caso anche il tempo trascorso dall'inizio del servizio.

Generalmente, siamo interessati nella soluzione del sistema "a regime", ovvero nello **stato stazionario** (*steady state solution*). Quando il sistema è già in funzione da un bel po' di tempo, si può ipotizzare che le condizioni iniziali non abbiano più influenza sul suo comportamento corrente. Nello stato stazionario, alcune distribuzioni di probabilità non cambiano molto, ovvero la distribuzione dei clienti nel sistema o la distribuzione dei tempi di risposta non cambia. Questo per ben distinguere dalla **fase transiente** (*transient phase*), quando le condizioni iniziali hanno ancora influenza sul comportamento del sistema.
Sfortunatamente, tanto maggiore è la complessità delle distribuzioni dei tempi di servizio e di arrivo dei clienti, tanto più complicata diventa la teoria delle code associata al sistema. Se entrambe le distribuzioni sono Markoviane, si può trovare una completa soluzione analitica per molte delle quantità di interesse. All'opposto, come ad esempio per il caso di una coda G/G/1, non si può fare molto, in quanto in generale i tempi medi di attesa non sono noti.

Per studiare, quindi, i sistemi a coda, non è sufficiente conoscere la dimensione del buffer, il numero di serventi, la strategia di servizio dei buffer, ma occorre anche conoscere la distribuzione di probabilità dei tempi di servizio e la distribuzione di probabilità degli arrivi.
Nella maggior parte dei casi, se si conosce:

- la distribuzione dei tempi di interarrivo dei clienti, e
- la distribuzione dei tempi di servizio

si possono ricavare grandezze importanti quali [8]:

- il numero medio dei clienti che esiste nel sistema,
- il numero medio dei clienti in attesa,
- il numero medio dei clienti in servizio,
- **l'utilizzazione**, definita come la percentuale di tempo in cui il sistema è occupato,
- **il throughput** o produttività che è una misura media della velocità di uscita dal sistema, ovvero il numero di medio di utenti serviti per unità di tempo,
- il ritardo medio subito dal cliente nel sistema, nella sua totalità o in ciascuna parte del sistema stesso, singolo buffer o servente.

Mentre nel capitolo successivo vedremo quali sono gli input e gli output necessari alla descrizione di un modello a rete di code, per quanto riguarda i metodi per la loro risoluzione (Analisi Operazionale, Analisi Asintotica e Soluzione Analitica) si rimanda all'Appendice A:.

4 Input e output di un modello a reti di code

In questo capitolo tratteremo solo le reti di code separabili, sottoinsieme delle reti di code, perché sono più semplici e richiedono un volume minore di calcoli. Da un lato, le richieste per l'applicabilità di questi modelli non sono sempre soddisfatte; dall'altro, le imprecisioni che risultano dall'applicazione di questi modelli a sistemi reali sono di solito trascurabili rispetto a quelle introdotte in altre fasi del processo di modellazione. Un'analisi più dettagliata delle reti di code generiche e delle ipotesi che debbono essere soddisfatte affinché un sistema possa essere rappresentato da una rete di code separabile può essere trovata in [8] e [9].
Una rete di code si dice essere **separabile** se ha le seguenti caratteristiche:

- *service center flow balance* : gli arrivi uguagliano i completamenti;
- *one step behavior* : in un dato istante un solo cliente risulta essere in movimento, in entrata o in uscita dal centro di servizio;
- *routing homogeneity* : il cammino dei clienti nella rete non dipende dallo stato del sistema;
- *device homogeneity* : il tasso di completamento dei job ad un dato service center non dipende dallo stato della rete;
- *homogeneus external arrivals* : i tempi di arrivo dei job dal mondo esterno non dipendono dallo stato della rete.

Queste assunzioni sono sufficienti a garantire che il modello sia separabile (e quindi può essere efficientemente valutato); inoltre, per poter applicare in maniera efficiente alcuni algoritmi che vedremo in seguito, si dovrà fare un'ulteriore assunzione:

- *service time homogeneity*: il tasso di completamento dei job ad un service center, mentre è occupato, non dipende dal numero di job presenti nel centro.

Quest'ultima assunzione, assicura che tutti i centri di servizio siano indipendenti dal carico, cioè il service rate del servente è indipendente dallo stato della coda al centro di servizio.

4.1 Input del modello con una sola classe di clienti

Per consentire l'analisi di un sistema con una sola classe di clienti è necessario descrivere il cliente medio che usufruisce dei servizi forniti dal sistema, specificare i centri che compongono il sistema e indicare il servizio che i clienti richiedono ai diversi centri.

Descrizione dei clienti
La descrizione del cliente medio permette di identificare l'intensità del carico che viene applicato al sistema in esame. Si distinguono tre diversi casi [11]:

1. Lavori **transazionali**. Il carico viene espresso in termini di frequenza di arrivo (λ). In questo caso, quando un cliente ha completato il servizio, lascia il sistema. La popolazione (numero di clienti presenti nel sistema) varia molto nel tempo.
2. Lavori **batch**. Il carico di lavoro viene indicato dalla popolazione (N), che in questo caso è fissa. Ogni job che ha completato il servizio lascia il sistema, ma viene immediatamente sostituito da un altro job in attesa in coda.
3. Lavori **interattivi**. Il carico di lavoro viene espresso da due parametri: N, indica il numero di terminali/clienti attivi, e Z, che indica il tempo medio di riflessione (*think time*).

Si definiscono **modelli aperti** quei modelli che si riferiscono a sistemi in cui il numero di richieste di elaborazione (clienti) presenti contemporaneamente nel sistema non è limitato (carico transazionale). Si chiamano **modelli chiusi** quelli che invece si riferiscono a sistemi in cui il numero massimo di richieste presenti è costante (carico di tipo batch) o comunque limitato superiormente (carico interattivo).

Si noti come un carico dovuto a lavori interattivi assomiglia ad un carico dovuto a lavori batch perché la popolazione (N) è costante in entrambi i casi, mentre il tempo di riflessione (Z) è nullo per i lavori batch. Però nel caso di carico dovuto a lavori interattivi, la popolazione all'interno del sottosistema centrale è variabile, cioè simile al caso transazionale. Quindi, nel sottosistema centrale, N è :

- variabile senza limite superiore per lavori transazionali;
- variabile fino al numero di terminali per lavori interattivi;
- fisso per lavori batch.

Descrizione dei centri di servizio

È necessario specificare il numero complessivo dei centri poi, per ogni centro, bisogna indicare se si tratta di un **centro ad accodamento** (il tempo di attraversamento comprende il tempo di servizio e il tempo di attesa in coda) oppure se si tratta di un **centro di ritardo** (ogni cliente ha un proprio punto di servizio e quindi il tempo di residenza equivale al tempo di servizio, infatti non c'è competizione per il servizio e quindi non c'è attesa in coda).

Queueing Center Delay Center

Figura 4.1 - Esempi di centro di accodamento (risorsa con coda d'attesa) e centro di ritardo (terminali)

Richieste di servizio

Il servizio che un cliente richiede a un centro di servizio del sistema può essere espresso in due modi:

1. specificando la richiesta di servizio S_i per ogni visita al centro, e il numero V_i degli accessi (visite) che ogni cliente richiede al centro;
2. specificando la domanda di servizio complessiva D_i, che può essere calcolata come il prodotto della richiesta di servizio e il numero di visite:

   $$D_i = V_i S_i$$

   Può inoltre, essere definita la domanda di servizio totale richiesta da un cliente a tutte le risorse come:

   $$D = \sum_{i=1}^{M} D_i \ .$$

Ricapitolando abbiamo la seguente tabella:

Descrizione dei Clienti (***Customer Description***)	Carico di lavoro, dato da: – λ: la frequenza di arrivo (***transaction workload***); – ***N***: la popolazione (***batch workload***); – ***N+Z***: popolazione e think time (***interactive workload***).
Descrizione dei Centri di Servizio (***Center Description***)	Può essere data specificando il numero ***M*** dei centri di servizio, e per ogni centro di servizio $1 \le i \le M$ si deve specificare il suo tipo: ***queueing center*** o ***delay center***.
Richieste di Servizio (***Service Demand***)	Per ogni centro di servizio $1 \le i \le M$ si deve definire la domanda di servizio: $D_i \equiv V_i S_i$

4.2 Output del modello con una sola classe di clienti

Gli output di un modello separabile a singola classe di clienti dipendono dai valori di tutti gli input del modello. Possiamo avere output relativi al singolo centro di servizio o all'intero sistema.

Output per singoli centri di servizio

Gli output ottenibili sono l'utilizzazione, il residence time, il throughput e la lunghezza della coda:

- U_i utilizzo dell'i-esimo centro (percentuale di tempo in cui il centro è occupato oppure numero medio di clienti che hanno ricevuto servizio da quel centro);
- R_i tempo di residenza complessivo al centro i, sia in servizio (S_i) che in coda (R_i-S_i); questo valore si riferisce a tutte la visite; il tempo di residenza per ogni singola visita è dato da: R_i / V_i ;
- X_i traffico in uscita dal centro i;
- Q_i clienti presenti nel centro i sia in servizio (U_i) che in attesa (Q_i-U_i).

Output per l'intero sistema

Gli output ottenibili sono il tempo di risposta, il throughput e il numero medio di clienti presenti nel sistema:

- R tempo di risposta del sistema, dato da $R = \sum_{i=1}^{M} R_i$;
- X traffico in uscita dal sistema. Se il sistema è stato parametrizzato in termini di D_i si riesce a calcolare X, ma non a suddividerlo nei vari X_i (la legge del flusso forzato è infatti X_i=$V_i X$); si nota quindi come la parametrizzazione in termini di S_i e V_i dia un maggiore livello di dettaglio;
- Q numero medio di clienti presenti nel sistema, che può essere calcolato sia come somma dei clienti presenti ai diversi centri:

$$Q = \sum_{i=1}^{M} Q_i$$

sia applicando la legge di Little [47] (che vedremo in seguito), da cui si ricava:

- $Q = N$ per clienti batch;
- $Q = XR$ per clienti transazionali;
- $Q = N$-XZ per clienti interattivi.

4.3 Input per modelli con più classi di clienti

Consideriamo ora il caso in cui vi siano più classi di clienti e con K indichiamo il numero di queste classi. Ogni parametro di ingresso deve essere riferito a una classe di clienti e perciò nella notazione fin qui usata verrà introdotto un pedice k.

Si distinguono:

1. modelli aperti: se tutte le classi sono di tipo transazionale
2. modelli chiusi: se tutti i lavori sono di tipo interattivo o batch;
3. modelli misti: se ci sono classi di tipo interattivo o batch e classi di lavori transazionali.

Descrizione dei clienti

Per ogni classe, l'intensità di carico è data specificando uno dei seguenti valori:

- λ_k frequenza di arrivo (transazionali);
- N_k popolazione (batch);
- N_k , Z_k popolazione e tempo di riflessione (interattivi).

risulta quindi che l'intensità del carico di lavoro in un modello con classi multiple è ben descritto da un vettore con una entry per ogni classe:

$\vec{\lambda} \equiv (\lambda_1, \lambda_2, \ldots, \lambda_M)$ per modelli aperti;
$\vec{N} \equiv (N_1, N_2, \ldots, N_M)$ e $\vec{Z} \equiv (Z_1, Z_2, \ldots, Z_M)$ per modelli chiusi;
$\vec{I} \equiv ((N_1 \; o \; \lambda_1), (N_2 \; o \; \lambda_2), \ldots, (N_M \; o \; \lambda_M))$ per modelli misti.

Descrizione dei centri di servizio
Per ogni centro si deve specificare se si tratta di un centro ad accodamento o di ritardo. Non specificheremo la disciplina di servizio, così come nel caso di singola classe di clienti, in quanto supporremo che la disciplina di servizio sia indipendente dalla classe (*class independent*), cioè essa non fa uso delle informazioni sulle classi dalle quali provengono i clienti.

Richieste di servizio
Per ogni classe *k* e centro *i* si specifica la richiesta di servizio indicando $D_{k,i}$ oppure $V_{k,i}$ e $S_{k,i}$ dato che $D_{k,i} \equiv V_{k,i} S_{k,i}$. Come per il caso della singola classe di servizio, possiamo definire la domanda totale di servizio di una classe di clienti *k* come:

$$D_k = \sum_{i=1}^{M} D_{k,i}$$

Ricapitolando abbiamo la seguente tabella:

Descrizione dei Clienti (***Customer Description***)	Deve essere specificato $1 \leq \boldsymbol{k} \leq \boldsymbol{K}$ il numero di classi di clienti; Per ogni classe $\boldsymbol{k}$ il suo carico di lavoro, dato da: – $\boldsymbol{\lambda_k}$: la frequenza di arrivo (***transaction workload***); – $\boldsymbol{N_k}$: la popolazione (***batch workload***); – $\boldsymbol{N_k}+\boldsymbol{Z_k}$: popolazione e think time (***interactive workload***).
Descrizione dei Centri di Servizio (***Center Description***)	Può essere data specificando il numero $\boldsymbol{M}$ dei centri di servizio, e per ogni centro di servizio $1 \leq \boldsymbol{i} \leq \boldsymbol{M}$ si deve specificare il suo tipo: ***queueing center*** o ***delay center***.
Richieste di Servizio (***Service Demand***)	Per ogni classe $1 \leq \boldsymbol{k} \leq \boldsymbol{K}$ e per ogni centro di servizio $1 \leq \boldsymbol{i} \leq \boldsymbol{M}$ si deve definire la domanda di servizio: $D_{k,i} \equiv V_{k,i} S_{k,i}$

4.4 Output per modelli con più classi di clienti

Output per singoli centri di servizio

Gli output possono essere ottenuti per classe o in modo aggregato. Si considerano dapprima i valori suddivisi per classe:

- $U_{k,i}$ utilizzo del centro *i*-esimo dovuto a clienti della classe *k*;
- $R_{k,i}$ tempo di residenza dei clienti della classe *k* all'*i*-esimo centro;
- $X_{k,i}$ traffico dei clienti della classe *k* in uscita dall'*i*-esimo centro;
- $Q_{k,i}$ clienti della classe *k* presenti nel centro *i*-esimo;

I valori aggregati, sono forniti da:

- U_i utilizzo del centro *i*-esimo, dato da: $U_i = \sum_{k=1}^{K} U_{k,i}$
- R_i tempo di residenza complessivo al centro *i*-esimo, dato da: $R_i = \sum_{k=1}^{K} \frac{R_{k,i} X_{k,i}}{X_i}$
- X_i traffico in uscita dal centro *i*-esimo, dato da: $X_i = \sum_{k=1}^{K} X_{k,i}$
- Q_i clienti presenti nel centro *i*-esimo, dato da: $Q_i = \sum_{k=1}^{K} Q_{k,i}$

Output per l'intero sistema

Anche i valori di uscita relativi al sistema complessivo possono essere suddivisi per classe oppure aggregati.

Per classe abbiamo:

- R_k tempo di residenza per tutti i clienti della classe *k*;
- X_k traffico dei clienti della classe *k* in uscita dal sistema;
- Q_k numero medio di clienti della classe *k* presenti nel sistema;

I valori aggregati invece sono dati da:

- R tempo di residenza nel sistema, dato da: $R = \sum_{k=1}^{K} \frac{R_k X_k}{X}$
- X traffico in uscita dal sistema, dato da: $X = \sum_{k=1}^{K} X_k$
- Q numero medio di clienti presenti nel sistema, dato da: $Q = \sum_{k=1}^{K} Q_k$

5 PMIF 2.0

PMIF è l'acronimo di Performance Model Interchange Format. PMIF definisce un formato di interscambio per modelli di performance, ossia un meccanismo mediante il quale le informazioni relative ad un modello di un sistema possano essere trasferite da un tool di performance ad un altro. PMIF consente a tool diversi di scambiarsi informazioni e richiede come unico requisito che essi supportino PMIF o prevedano comunque un meccanismo o un'interfaccia per leggere e scrivere le specifiche del modello da o su un file.
I vantaggi di PMIF sono molteplici, primo tra tutti, come già detto, la possibilità di trasportare un modello relativo ad un sistema da un tool all'altro. Questo consentirebbe ad esempio di iniziare lo studio di un sistema con un tool e poi magari trasferire il modello in un altro tool più potente senza il bisogno di tradurre il modello nel linguaggio nativo del secondo tool.
I vantaggi nell'uso di PMIF possono essere quindi così sintetizzati:

1. Possibilità di comparare le soluzioni provenienti da tool multipli e scoprire così eventuali criticità e difetti;
2. Imparare un unico linguaggio di modellazione PMIF/XML, invece di imparare i linguaggi specifici di molti tool;
3. Possibilità di usare temporaneamente un tool diverso da quello usato per modellare una prima bozza del sistema, per analizzare con maggior precisione e più in dettaglio un modello;
4. Possibilità di poter migrare definitivamente un modello in un tool più potente senza doverlo riscrivere;
5. Analizzare le diverse caratteristiche di un sistema con i tool più adatti allo scopo;
6. Poter comparare le caratteristiche di tool differenti prima di acquistarne uno;
7. Poter validare algoritmi di risoluzione comparandoli con i risultati ottenuti con altri tool;
8. Permettere a tool di generazione di modelli a reti di code di avere un unico linguaggio target.

PMIF fornisce una base comune che i diversi tool di risoluzione di modelli di performance possono usare come interfaccia. Senza un linguaggio comune, i tool che volessero colloquiare e scambiare modelli dovrebbero sviluppare un sistema di import/export dei modelli ad hoc per ogni tool con cui voglio parlare. Per N tool che volessero poter scambiare modelli tra di loro bisognerebbe implementare un totale di:

$$4\binom{N}{2} = 4\frac{N!}{2!(N-2)!}$$

funzioni di import/export.
Mentre, se tutti i tool supportassero il linguaggio comune PMIF, il numero di funzioni di import/export sarebbero semplicemente 2N.

PMIF 2.0 [18] è un'evoluzione di PMIF (identificato anche come PMIF 1.0) [19][20]. PMIF 1.0 si basava su paradigma in standard EIA/CDIF (Electronic Industries Association/CASE Data Interchange Format). CDIF identifica una famiglia di standard che descrive una metodologia di trasferimento di informazioni tra CASE tool. Lo standard definisce un formato di interscambio che consente ai tool, che hanno una organizzazione interna dei database e dei formati di storage differenti, di scambiarsi le informazioni. Lo scambio avviene attraverso un file, e le informazioni interne del tool vengono tradotte nel formato di interscambio comune. La versione originale del formato CDIF prevedeva il LISP come linguaggio di definizione dei modelli. Nell'implementazione di PMIF 2.0 si è scelto invece XML come linguaggio finale. Questo in quanto XML è stato sviluppato proprio per questi scopi e ci sono molti tool che già supportano XML come formato di interscambio per basi di dati.

Nello standard CDIF, le informazioni che devono essere scambiate tra due tool prendono il nome di modello (*model*). Il contenuto del modello viene definito usando un meta-modello (*meta-model*). Un meta-model definisce la struttura delle informazioni di un piccolo insieme di CASE tool (come ad esempio data-modeling diagrams o data-flow diagrams) conosciuto come "*Subject Area*".

Nello standard CDIF un tool che vuole esportare (*export*) un modello deve fornire tutti i dati di cui dispone che sono previsti nel meta-model. Il tool dovrà anche prevedere dei valori di default per i dati essenziali del meta-modello CDIF, nel caso non siano previsti o disponibili dati reali.

Per poter importare (*import*) un modello in un formato CDIF, i tool dovranno usare i dati forniti, scartando quelli superflui e facendo delle assunzioni sui dati mancanti, non compresi nel meta-modello di base. Dopo aver importato un modello, il tool potrebbe richiedere all'utente di inserire i dati mancanti nel caso il tool di destinazione supporti delle feature non previste dal tool che ha generato il file di interscambio.

Il meta-modello CDIF che è stato usato per definire modelli per reti di code prende il nome di *QNM meta-model* da cui nasce la versione originale del *PMIF 1.0 Meta-Model* che usava come linguaggio di interscambio il LISP e successivamente dal *QNM meta-model 2.0* il *PMIF 2.0 Meta-Model* che usa XML. Di seguito descriveremo la versione 2.0.

5.1 QNM Meta-Model 2.0

Questo modello prende il nome di QNM meta-model, perché rappresenta un "modello" delle informazioni necessarie alla costruzione di un Modello a Reti di Code (in inglese abbreviato in QNM). Questo meta-modello serve a due scopi. Il primo è di fornire una rigorosa definizione delle informazioni richieste per un QNM che debba essere risolto mediante tecniche analitiche esatte. Il secondo scopo è di definire in modo formale PMIF, usando XML come formato di interscambio, derivandolo dal meta-modello. Il Class Diagram che segue mostra la definizione del QNM Meta-Model così come formulato da C.U. Smith [19][20] e successive modifiche [18].

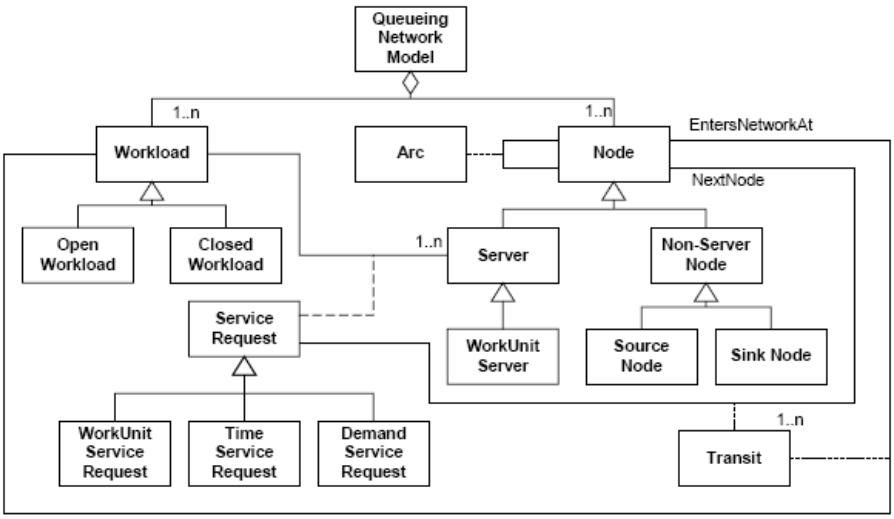


Figura 5.1 - QNM Meta-Model in UML

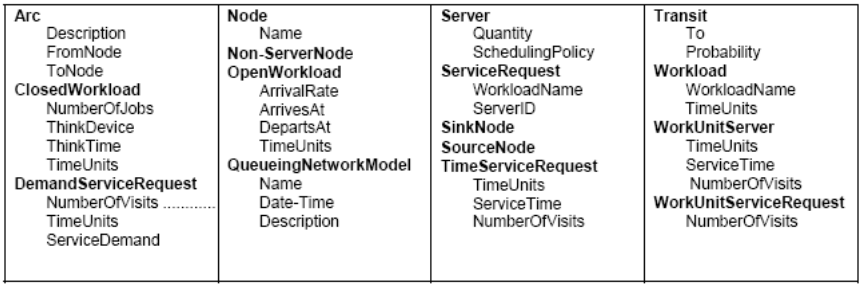


Figura 5.2 - Classi e Attributi del QNM meta-model

Dal diagramma si deduce che un modello a reti di code (*QueueingNetworkModel*) è composto da uno o più nodi (*Nodes*), zero o più archi (*Arcs*), e uno o più carichi di lavoro o classi di job (*Workloads*). Un arco connette un nodo ad un altro nodo.
Per costruire un modello a reti di code possono essere usati diversi tipi di nodi:

- ***Server***: rappresenta una componente dell'ambiente di esecuzione che fornisce qualche forma di servizio di elaborazione. Un *Server* può essere un *WorkUnitServer,* cioè un server che esegue una quantità fissa di lavoro (servizio di elaborazione) per ogni *Workload* che fa una richiesta di servizio.
- ***Non-Server Node***: Rappresenta dei nodi che mostrano la topologia del modello ma che non forniscono alcun servizio specifico. Ci sono due tipi di *Non-Server Node*:
  - ***Source Node***: Rappresenta il punto di ingresso o di origine per i job di un *OpenWorkload*.
  - ***SinkNode***: Rappresenta il punto di uscita di un *OpenWorkload*.

Un *Server* o un *WorkUnitServer* fornisce servizio a uno o più *Workload*. Un *Workload* rappresenta un insieme di transazioni o job che effettuano identiche richieste di servizio *ServiceRequest* ad un *Server*. Ci sono due tipi di *Workloads*:

- ***OpenWorkload***: Rappresenta un *Workload* con una popolazione potenzialmente infinita, dove le transazioni o job arrivano dal mondo esterno attraverso il *SourceNode*, ricevono servizio da un *Server*, e escono dalla rete attraversando il *SinkNode*. Un *OpenWorkload* ad un determinato istante t ha una popolazione variabile.
- ***ClosedWorkload***: Rappresenta un *Workload* con una popolazione fissa che circola attraverso i *Server*.

Dopo essere arrivato in un nodo ed aver ottenuto la sua elaborazione, un *Workload* transita (*Transit*) verso un altro nodo della rete con una data probabilità.
Una richiesta di servizio (*ServiceRequest*) associa i *Workload* con i *Server*. Essa specifica il cammino (attraverso i nodi della rete di code) che compiranno i job appartenenti a quel *Workload*, specificando il tempo medio di servizio (*TimeServiceRequest*), la domanda media di servizio (*DemandServiceRequest*) o il numero medio di visite (*WorkUnitServiceRequest*).

- Una ***TimeServiceRequest*** specifica il tempo medio di servizio ed il numero di visite fornito ad ogni *Workload* che visita un *Server*.
- Una ***DemandServiceRequest*** specifica la domanda media di servizio (che è data dal tempo medio di servizio moltiplicato per il numero di visite) fornita ad ogni *Workload* che visita un *Server*.
- Una ***WorkUnitServiceRequest*** specifica il numero medio di visite richieste da ogni *Workload* che visita un *WorkUnitServer*.

Al completamento di una *ServiceRequest*, il *Workload* transita (*Transit*) in un altro nodo con una data probabilità.

5.2 PMIF 2.0 XML Schema

Quanto sopra descritto è stato codificato da C. U. Smith ed altri [18][19][20] in XML mediante un W3C XML Schema XSD (XML-Schema Definition) riportato in Appendice C:. La versione ufficiale ed aggiornata di tale schema è reperibile all'indirizzo: http://www.perfeng.com/pmif/pmifschema.xsd.

Vediamo ora in dettaglio il significato degli elementi (*Entities*) che compongono un file in formato PMIF 2.0:

- **QueueingNetworkModel**: Un QueueingNetworkModel (Modello a Reti di Code) rappresenta una rete di Server interconnessi detti nodi della rete (Nodes), i quali forniscono un servizio di elaborazione (*processing service*) per i Workloads che ne fanno richiesta (ServiceRequest).
- **Node**: Un Node (Nodo) rappresenta una entità nell'ambiente di esecuzione (*execution environment*) dell'QueueingNetworkModel, che fornisce un servizio o è utile per designare la topologia del modello a reti di code (come nel caso di SinkNode e SourceNode, che non forniscono un servizio ma completano la topologia della rete di code). Un nodo della rete può essere di tre tipi:
  - **Server**: Un Server rappresenta un nodo dell'execution environment che fornisce un qualche tipo di servizio (*processing service*) ai Workloads che gli fanno visita. Il tempo di servizio non è specificato in quanto può variare per classi di transazioni o jobs (Workloads).
  - **WorkUnitServer**: Un WorkUnitServer è un Server che dedica lo stesso tempo di servizio (ServiceTime) a tutti i Workloads che lo attraversano. Un WorkUnitServer è cioè un Server che non è in grado di gestire le priorità tra Workloads. Si può pensare ad esempio ad un Hard Disk che ha un ben determinato tempo medio di accesso, e quindi, nel caso che sia il sistema operativo o che sia un'applicazione ad effettuare una richiesta di servizio, il tempo di accesso ai dati non cambia. Mentre, ad esempio, una CPU ha dei meccanismi interni tali da garantire che certe applicazioni ad alta priorità (come ad esempio il sistema operativo) siano eseguite più velocemente rispetto alle applicazioni utente a bassa priorità.
  - **Non-ServerNode**: Un Non-ServerNode (Nodo che non è un Server) rappresenta un nodo dell'execution environment utile solo al fine di completare la topologia del modello, ma che non fornisce alcun tipo di servizio di elaborazione.
    - **SourceNode**: Un SourceNode rappresenta un nodo dell'execution environment che rappresenta l'origine di un OpenWorkload, cioè il punto in cui è entrato nella rete di code. Un SourceNode non fornisce alcun tipo di elaborazione.
    - **SinkNode**: Un SinkNode rappresenta un nodo nel quale un OpenWorkload termina il suo cammino, dopo aver ricevuto servizio, uscendo da una rete di code. Un SinkNode non fornisce alcun tipo di servizio.
- ***Arc***: Un Arc (Arco) connette due Nodes di un QueueingNetworkModel. L'attraversamento di un arco rappresenta il completamento di una richiesta di servizio fatta al nodo FromNode (Nodo Origine) ed il nascere di una nuova richiesta di servizio fatta al nodo ToNode (Nodo Destinazione).
- **Workload**: Un Workload (carico di lavoro) rappresenta un insieme di transazioni o jobs che effettuano richieste di servizio simili ai Servers di un QueueingNetworkModel.
  - **OpenWorkload**: Un OpenWorkload (Carico di lavoro "Aperto") è un Workload con una popolazione potenzialmente infinita, dove le transazioni (*transactions*) o i job arrivano dal mondo esterno alla rete (attraverso un SourceNode), ricevono un servizio dai nodi della rete, e infine escono (attraverso un SinkNode). La popolazione di un OpenWorkload è, in ogni istante, variabile.
  - ***ClosedWorkload***: Un ClosedWorkload (Carico di Lavoro "Chiuso") è un Workload (Carico di Lavoro) con una popolazione fissa che circola tra i Servers.

- **ServiceRequest**: Una ServiceRequest è una richiesta di servizio diretta ad un Server. Una richiesta di servizio può essere fatta specificando o il tempo medio di servizio (e in tal caso si parla di TimeServiceRequest) o la domanda media di servizio fornita ad ogni Workload che visita il Server (e in tal caso si parla di DemandServiceRequest) o, infine, il numero medio di visite fatte ad un WorkUnitServer (e in tal caso si parla di WorkUnitServiceRequest).
  - **DemandServiceRequest**: Una DemandServiceRequest è una richiesta di servizio diretta ad un Server, fatta specificando la domanda media di servizio (tempo di servizio moltiplicato per il numero delle visite) fornito ad ogni Workload che visita il Server specificato.
  - **TimeServiceRequest**: Una TimeServiceRequest è una richiesta di servizio diretta ad un Server, fatta specificando il tempo medio di servizio ed il numero di visite fornite per ogni Workload che visita il Server specificato.
  - **WorkUnitServiceRequest**: Una WorkUnitServiceRequest è una richiesta di servizio diretta ad un WorkUnitServer, fatta specificando il numero medio di visite che i Workloads effettueranno al WorkUnitServer.

La differenza tra Server e WorkUnitServer, è molto sottile e di difficile comprensione. In realtà, non è ben chiaro quando si possa o debba usare uno e quando l'altro. Dagli esempi proposti dagli autori, sembra di intuire che la scelta sia in qualche modo legata alle scheduling policy, ma nella documentazione ufficiale non si fa alcun riferimento ai casi d'uso. Le due tipologie di nodo sembrano in qualche modo intercambiabili, e sembra esistere la possibilità di modellare la stessa rete di code in modi differenti.

Per alcuni degli elementi di un file PMIF 2.0 visti sopra, esiste la possibilità di specificare alcuni attributi (*Attributes*), vediamone in dettaglio il significato:

- **Date-Time**: È la data e ora di creazione del modello. Deve rispettare un formato ben preciso che è il seguente: AAAA-MM-GGThh:mm:ss ovvero, deve contenere l'anno compreso il secolo, il carattere "-", il mese, il carattere "-", il giorno, il carattere "T", l'ora, il carattere ":", i minuti, il carattere ":" ed i secondi.
- **Description**: È una descrizione testuale del modello a rete di code.
- **ArrivalRate**: É la frequenza media con cui le transazioni o jobs arrivano dal mondo esterno, ricevono servizio, ed escono dalla rete.
- **ArrivesAt**: É il nome del SoruceNode di un OpenWorkload.
- **DepartsAt**: É il nome del SinkNode di un OpenWorkload.
- **FromNode**: É il nome del nodo (Node) di origine di arco (Arc).
- **Name**: È il nome che si assegna all'elemento che si sta definendo, sia esso il modello o un nodo.
- **NumberOfJobs**: Indica la popolazione fissa (numero totale di transazioni o job) che circola attraverso i nodi (Nodes) della rete.
- **NumberOfVisits**: Indica il numero medio di visite inviate ad un Server da una ServiceRequest.
- **Probability**: È la probabilità di transizione da un nodo all'altro assegnata ad una ServiceRequest mediante l'elemento Transit presente nei tre tipi di richieste di servizio (TimeServiceRequest, DemandServiceRequest e WorkUnitServiceRequest).
- **Quantity**: Indica il numero di istanze di un dato Server. Quando è maggiore di uno, siamo in presenza di un nodo di tipo Multiple Servers. Un nodo multiple servers ha un unica coda per tutte le richieste di servizio, ma più unità di elaborazione.
- **SchedulingPolicy**: Indica la disciplina di servizio usata per selezionare, dalla coda di attesa, la prossima richiesta di servizio (ServiceRequest) che deve essere servita.
- **ServerID**: Indica il nodo a cui si riferisce una ServiceRequest.

- **ServiceDemand**: Indica la domanda di servizio totale per una ServiceRequest. La domanda di servizio è il prodotto del tempo di servizio (ServiceTime) per il numero di visite (NumberOfVisits).
- **ServiceTime**: Indica la quantità di tempo necessaria ad un Server per eseguire una unità di servizio. Una unità di servizio è la quantità di tempo fornita per ogni visita al Server.
- **To**: È il nome del nodo destinazione presente nell'elemento Transit.
- **ToNode**: É il nome del nodo destinazione di un arco.
- **ThinkTime**: Indica l'intervallo medio di tempo che trascorre tra il completamento di una transazione o job e l'arrivo della prossima transazione o job in un ClosedWorkload.
- **ThinkDevice**: È il nome del Server a cui si riferisce un ClosedWorkload.
- **TimeUnits**: Ove presente, indica l'unità di tempo usata per specificare una quantità temporale. Se viene omessa, tutte le quantità riferibili al trascorrere del tempo vengono assunte come avere la stessa unità di tempo.
- **WorkloadName**: È il nome assegnato al Workload. Come l'attributo Name è semplicemente il nome che identifica l'elemento che si sta definendo.

Le specifiche PMIF 2.0 sono poco chiare relativamente all'attributo TimeUnits, in quanto essendo opzionale in tutti gli elementi che compongono un file PMIF, non è chiaro come comportarsi nel caso in cui venga specificato in quasi tutti gli elementi con unità di tempo differenti (ore, minuti, secondi, ecc) tranne in uno. Nel caso sia assente in un solo elemento ma presente in tutti gli altri, la strategia di importazione dovrebbe decidere di convertire tutti i tempi in un'unica unità, ma quale? Ore? Minuti? o Secondi? L'algoritmo di importazione dovrebbe forse decidere di assumere l'unità inferiore? o quella superiore? Nella documentazione ufficiale di PMIF 2.0 non vi è alcun riferimento a questa particolare situazione, ma si fa solo riferimento ai tool *SPE·ED* e QNAP2, il primo usa i secondi come unità di default, mentre il secondo richiede soltanto che vi siano unità di misura "consistenti". Ovviamente si suppone che il modello PMIF sia stato esportato da un tool e che abbia quindi usato le stesse unità di misura per tutte le unità di tempo. Ma nel caso il file PMIF sia stato scritto a mano può capitare di imbattersi nella situazione descritta. Nell'esempio proposto dagli autori delle specifiche PMIF 2.0 si usa il tool QNAP2. Nel file di conversione XSLT per l'import di modelli da PMIF 2.0 a QNAP2, non vi è traccia di alcuna funzione di conversione per le unità di tempo, ne tanto meno di una qualche funzione che tenga conto del caso anomalo sopra indicato.

Vi è, infine, un problema con la definizione dei Workloads. La specifica mediante schema XML, prevedere per i Workload la seguente definizione:

```
...
<xsd:complexType name="WorkloadType">
        <xsd:choice minOccurs="1" maxOccurs="unbounded">
                <xsd:element name="OpenWorkload" minOccurs="0" maxOccurs="unbounded">
                        <xsd:complexType>
                                <xsd:sequence>
                                        <xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
                                </xsd:sequence>
                                 <xsd:attribute name="WorkloadName" type="xsd:ID" use="required"/>
                                <xsd:attribute name="ArrivalRate" type="nonNegativeFloat" use="required"/>
                                <xsd:attribute name="TimeUnits" type="TimeUnitsType" use="optional"/>
                                <xsd:attribute name="ArrivesAt" type="xsd:IDREF" use="required"/>
                                <xsd:attribute name="DepartsAt" type="xsd:IDREF" use="required"/>
                        </xsd:complexType>
                </xsd:element>
                <xsd:element name="ClosedWorkload" minOccurs="0" maxOccurs="unbounded">
                        <xsd:complexType>
                                <xsd:sequence>
                                        <xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
                                </xsd:sequence>
                                <xsd:attribute name="WorkloadName" type="xsd:ID" use="required"/>
                                <xsd:attribute name="NumberOfJobs" type="xsd:nonNegativeInteger" use="required"/>
                                <xsd:attribute name="ThinkTime" type="nonNegativeFloat" use="required"/>
                                <xsd:attribute name="TimeUnits" type="TimeUnitsType" use="optional"/>
                                <xsd:attribute name="ThinkDevice" type="xsd:IDREF" use="required"/>
                        </xsd:complexType>
                </xsd:element>
        </xsd:choice>
</xsd:complexType>
...
```

Come si può vedere la specifica prevede, tramite il tag <xsl:choice…> che vi sia almeno una (minOccurs="1") definizione di OpenWorkload o ClosedWorkload.

Tuttavia all'atto pratico di validare il modello risulta che un file PMIF 2.0 che non specifica alcun Workload risulti comunque valido.

Ad esempio consideriamo il seguente modello PMIF 2.0:

```
<?xml version="1.0"?>
<QueueingNetworkModel Name="Central Server Model" Description="Central Server Model Queuing System" Date-Time="2006-03-05T19:34:00"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://www.perfeng.com/pmif/pmifschema.xsd">
        <Node>
                <WorkUnitServer Name="CPU" Quantity="1" SchedulingPolicy="FCFS" ServiceTime="50"/>
                <WorkUnitServer Name="DISK1" Quantity="1" SchedulingPolicy="FCFS" ServiceTime="33.333333333333333333333333333"/>
                <WorkUnitServer Name="DISK2" Quantity="1" SchedulingPolicy="FCFS" ServiceTime="23.300246982618015750966960 24"/>
        </Node>
        <Arc FromNode="CPU" ToNode="DISK1"/>
        <Arc FromNode="CPU" ToNode="DISK2"/>
        <Arc FromNode="DISK1" ToNode="CPU"/>
        <Arc FromNode="DISK2" ToNode="CPU"/>
        <Workload>
                <!-- Non definiamo alcun Workload -->
        </Workload>
        <ServiceRequest>
                <WorkUnitServiceRequest WorkloadName="CHAIN1" ServerID="CPU" NumberOfVisits="10">
                        <Transit To="DISK1" Probability="0.667"/>
                        <Transit To="DISK2" Probability="0.233"/>
                        <Transit To="TERMINALS" Probability="0.1"/>
                </WorkUnitServiceRequest>
                <WorkUnitServiceRequest WorkloadName="CHAIN1" ServerID="DISK1" NumberOfVisits="11">
                        <Transit To="CPU" Probability="1"/>
                </WorkUnitServiceRequest>
                <WorkUnitServiceRequest WorkloadName="CHAIN1" ServerID="DISK2" NumberOfVisits="12">
                        <Transit To="CPU" Probability="1"/>
                </WorkUnitServiceRequest>
                <WorkUnitServiceRequest WorkloadName="CHAIN1" ServerID="TERMINALS" NumberOfVisits="13">
                        <Transit To="CPU" Probability="1"/>
                </WorkUnitServiceRequest>
        </ServiceRequest>
</QueueingNetworkModel>
```

Come si può notare, non vengono definiti Workload ma poi vengono richiamati in ServiceRequest.
Il motore di validazione di PHP5 e di altri tool, valida comunque il modello. Credo che i motori di validazione vengano ingannati dalla discrepanza di definizione esistente tra il minOccurs="1" del tag choice ed i minOccurs="0" presenti in OpenWorkload e ClosedWorkload, che in effetti legittimano la definizione di modelli senza alcun Workload.

5.3 Strategie di importazione ed esportazione dei modelli con PMIF

La strategia CDIF per quanto riguarda l'import e l'export dei modelli tra un tool e l'altro è "*export everything you know*" e "*import the parts you need*". Cioè un tool, che esegue l'export in formato PMIF 2.0 di un modello, deve salvare nel file XML tutto ciò che conosce, il maggior numero di informazioni possibile. Secondo questa strategia, non tutto ciò che si conosce è necessariamente tutto ciò che si userà riaprendo il modello. Tuttavia, ci si deve sempre e comunque mettere nell'ottica che il file salvato potrebbe essere aperto da un altro tool, il quale potrebbe non essere in grado di ricavare le informazioni mancanti. Mentre un tool che vuole importare un modello PMIF 2.0 deve estrarre dal file XML tutto ciò che gli serve o ciò che è disponibile. Le informazioni mancanti devono essere dedotte o assunte come default.
Se indichiamo con:

- $\boldsymbol{V_{k,c}}$ il numero medio di **V**isite alla risorsa **k** per i clienti della classe **c**;
- $\boldsymbol{S_{k,c}}$ il tempo medio di **S**ervizio per visita alla risorsa **k** per i clienti della classe **c**;
- $\boldsymbol{D_{k,c}}$ la **D**omanda di servizio alla risorsa **k** per i clienti della classe **c**.

Valgono le seguenti relazioni, utili per ricavare le informazioni mancanti da un file PMIF 2.0:

- $D_{k,c} = V_{k,c} \cdot S_{k,c}$ da cui si ricava
- $S_{k,c} = \frac{D_{k,c}}{V_{k,c}}$ e
- $V_{k,c} = \frac{D_{k,c}}{S_{k,c}}$

Il significato è il seguente:

- Domanda di Servizio = Numero di Visite × Tempo di Servizio
- Tempo di Servizio = Domanda di Servizio ÷ Numero di Visite
- Numero di Visite = Domanda di Servizio ÷ Tempo di Servizio

Queste relazioni possono essere codificate direttamente nel file di conversione XSL (da PMIF 2.0 al linguaggio nativo del tool), oppure inserite come funzioni di libreria all'interno del codice sorgente del tool.
In una prima versione del meta-model di PMIF, Smith e Williams [19][20] previdero l'uso del numero di visite al posto delle routing probabilities, dando per assunto il fatto che dal numero di visite, e conoscendo la topologia della rete di code, fosse possibile calcolare le routing probabilities. Questa assunzione, anche se vera in molti casi, non lo è nel caso generale. Per questo nelle specifiche di PMIF 2.0 [18] si è deciso di mantenere la possibilità (opzionale) di specificare il numero di visite con l'attributo: ***NumberOfVisit*** per ***DemandServiceRequest***, ***TimeServiceRequest*** e ***WorkUnitServiceRequest*** ma anche l'obbligo di specificare le routing probabilities attraverso il tag XML:

```
<Transit To=SERVER_ID Probability=FLOAT />
```

Il numero di visite, essendo opzionale, può comunque essere calcolato mediante la relazione:

$$V_i = p_{0,i} + \sum_{j=1}^{N} V_j \, p_{j,i} \qquad (i = 1, \ldots, N)$$

dove $\boldsymbol{p_{j,i}}$ è la probabilità (*routing probabilities*) che un job transiti dal server ***i*** al server ***j***, e $\boldsymbol{V_i}$ è il numero medio di visite di un job al'***i-esimo*** server. In una rete di code aperta, il server con indice **0** rappresenta il mondo esterno alla rete, da cui provengono i job.
La probabilità $\boldsymbol{p_{0,i}}$ si ottiene in modo immediato dalla frequenza di arrivo (*external arrival rate*) in quanto:

$$\boldsymbol{\lambda_{0,i} = \lambda \, p_{0,i}}$$

dove $\boldsymbol{\lambda}$ è la frequenza di arrivo dal mondo esterno alla rete nel suo complesso e $\boldsymbol{\lambda_{0,i}}$ è la frequenza di arrivo dei job dal mondo esterno al server ***i-esimo***. Da cui si ricava:

$$p_{0,i} = \frac{\lambda_{0,i}}{\lambda}$$

Mentre per una rete di code chiusa, la relazione diventa:

$$V_i = \sum_{j=1}^{N} V_j p_{j,i} \quad (i = 1, \ldots, N)$$

Essendoci solo N-1 equazioni indipendenti per calcolare la frequenza di visita in un modello chiuso, le $\boldsymbol{V_i}$ possono essere solo determinate a meno di una costante moltiplicativa, e si assume che $\boldsymbol{V_1} = 1$.

PMIF 2.0 contiene diverse informazioni ridondanti, proprio per consentire al maggior numero possibile di tool di ricavare le informazioni di cui necessita per specificare il modello nel proprio linguaggio interno. Il caso precedente è proprio un esempio di questa ridondanza, che rispecchia la filosofia di PMIF 2.0 di "*import-friendly*" [18].

Come già accennato in precedenza, esiste un problema con le unità di misura dei tempi, relativamente all'attributo **TimeUnits**. Tale attributo consente di specificare l'unità di misura per le quantità di tempo specificate in alcuni elementi di un file PMIF. Le unità di tempo che si possono utilizzare sono sei:

- day | Day = Giorni
- hr | Hr = Ore
- min | Min = Minuti
- sec | Sec = Secondi
- ms | Ms = Microsecondi
- ns | Ns = Nanosecondi

Le strategie di import delle specifiche PMIF 2.0 prevedono solo due casi:

1) Tutti gli elementi che specificano una quantità temporale dichiarano anche esplicitamente l'unità di tempo utilizzata.
2) Tutti gli elementi che specificano una quantità temporale non dichiarano a quale unità di tempo ci si sta riferendo.

Nel primo caso, si deve prevedere una funzione di conversione tra unità di tempo, a meno che il tool di destinazione non sia già in grado di supportare e specificare unità di tempo differenti. In tal caso si tratterebbe solo di convertire gli specificatori di unità nel linguaggio del tool.
Nel secondo caso, le specifiche PMIF 2.0 prevedono che venga assunto come default una unità di tempo relativa (RTU) non meglio specificata.
La casistica come si può facilmente dedurre non è esaustiva, esistono infatti altri casi:

3) Tutti gli elementi specificano una quantità temporale dichiarano anche l'unità di tempo, tranne uno che non la specifica.
4) Tutti gli elementi, che specificano una quantità temporale, dichiarano unità differenti per il tempo, tranne uno che non la specifica.

Nel terzo caso, su N elementi totali, l'algoritmo di conversione potrebbe decidere che essendo N-1 elementi riferibili all'unità di misura τ, anche l'N-esimo elemento, che non specifica una unità di tempo, debba per uniformità utilizzare l'unità di misura τ.
Nel quarto caso, ad esempio su N=7 elementi N-1=6 specificano unità di misura di tempo differenti (su 6 disponibili) come ad esempio: $N_{\tau 1}$ = "Day", $N_{\tau 2}$ = "Hr", $N_{\tau 3}$ = "Min", $N_{\tau 4}$ = "Sec", $N_{\tau 5}$ = "Ms", $N_{\tau 6}$ = "Ns" e $N_{\tau 7}$ = "?".
L'algoritmo di conversione deve quindi poter effettuare una scelta, per assegnare una unità di tempo all'N-esimo elemento. Una strategia potrebbe essere quella di assumere una unità di default, come ad esempio "Sec", e stabilire che, ove non diversamente specificato, si assume che l'unità di misura sia quella di default. Questa strategia non ci assicura che effettivamente la quantità di tempo sia espressa in secondi e potrebbe comportare errori di valutazione del modello. Un'altra strategia potrebbe essere

quella di bloccare la validazione del modello e chiedere all'utente di specificare l'unità di misura mancante. O in fine si potrebbe aggiungere un algoritmo di deduzione, che basandosi sulle similarità tra servers e tra workloads decida quale unità di misura assegnare.

5.4 Validare un modello PMIF 2.0

Avendo a disposizione lo schema XML di PMIF 2.0, è relativamente facile validare sintatticamente un modello PMIF 2.0, cioè verificare che esso contenga tutto ciò che si suppone ci debba essere, nel giusto posto, che gli IDREFS puntino ad ID effettivamente dichiarati, ecc. Quasi tutti i moderni linguaggi di programmazione, compreso PHP 5, hanno delle funzioni di libreria per trattare file XML, caricarli in un albero DOM, e validarli mediante uno schema XSD [31].
Altra cosa è validare semanticamente un modello PMIF 2.0, cioè verificare che tutti i nodi dichiarati siano effettivamente utilizzati, che i tag *Transit To* coincidano con nodi definiti mediante il tag *Arc*, che tutti i routing dichiarati siano validi, ecc. Tutti questi controlli non sono effettuabili con il solo schema XML, ma c'è bisogno di uno strato di logica addizionale allo schema XSD.
In Internet, all'indirizzo: http://dmi.uib.es/~cllado/pmif/validator è reperibile un "validatore" sintattico/semantico per PMIF 2.0 [48], realizzato in Java, da Daniel García Cousillas con la supervisione di Catalina M. Lladó, coautrice insieme a C. U. Smith delle specifiche PMIF 2.0.
Il tool sembra essere molto interessante e promettente; dispone di un'interfaccia grafica (Figura 5.3) ma, se si specifica un file PMIF sulla riga di comando, allora il tool si avvierà in modalità batch inviando il risultato della validazione allo standard output del sistema.

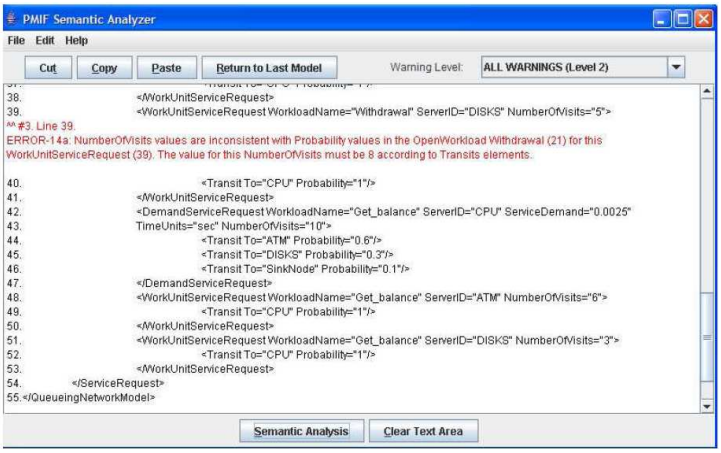


Figura 5.3 - PMIF Semantic Analyzer in modalità grafica

La sintassi, per l'invocazione in modalità batch è la seguente:

```
PmifValidation input.xml [output] [-warningLevel]
```

La capacità del validatore semantico di funzionare in modalità batch, lo rende facilmente integrabile all'interno di altri tool e quindi del nostro web service.

In [48] ci sono tutti i dettagli della sua implementazione (a cui rimandiamo per maggiori dettagli), mentre di seguito (Figura 5.4) riportiamo uno schema generale del suo processo di validazione.

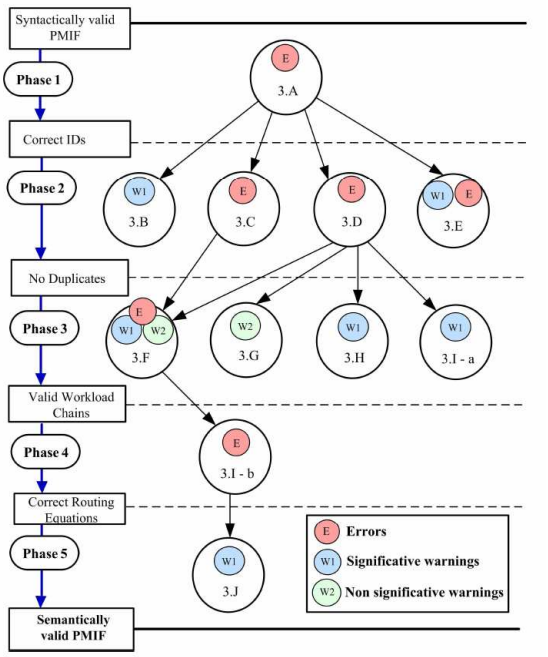


Figura 5.4 - Fasi del processo di validazione semantica (i numeri all'interno di ogni cerchio corrispondono alla sezione di [48] dove questa fase viene descritta)

5.5 Cosa manca a PMIF 2.0

PMIF 2.0 non è del tutto completo, nel senso che manca di definire alcuni aspetti della vita di un modello a reti di code.

Alcune caratteristiche assenti in PMIF 2.0, come ad esempio:

- scheduling policy addizionali come LIFO, QUANTUM, ecc,
- la possibilità di specificare le priorità dei Workload,
- la possibilità di specificare la distribuzione dei tempi di arrivo e di servizio: EXP, HEXP, CST, Erlang, Uniform, Cox, ecc,
- la possibilità di gestire l'aspetto grafico del modello, con informazioni di layout grafico dei nodi e degli archi;

sono facilmente implementabili modificando lo schema XML. Altre caratteristiche, invece, richiedono una definizione separata ed un diverso schema. Molti tool supportano caratteristiche avanzate e offrono la possibilità di definire un modello con elevata precisione ed in ogni suo aspetto. PMIF 2.0 è invece, nello stato attuale, molto più generico e lascia alle funzioni di import/export il compito di definire i dettagli non specificati nello schema. Ad esempio PMIF 2.0 non definisce il formato di output dell'analisi effettuata, ma si lascia intendere che il formato di output è quello del tool che esegue il modello. Sarebbe comunque utile avere delle specifiche comuni per i formati di output, al fine di poter confrontare i risultati prodotti dai diversi tool.

Molti tool consentono di specificare l'algoritmo di risoluzione del modello, come ad esempio algoritmo di convoluzione, algoritmo MVA, normalized convolution algorithm, iterative approximation, ecc. Mentre invece PMIF 2.0 non prevede alcun elemento dello schema che consenta di specificare l'algoritmo di risoluzione.

In conclusione PMIF 2.0 è uno standard in evoluzione, carente sotto alcuni aspetti, ma abbastanza generico da poter essere facilmente implementato e supportato dai tool esistenti e futuri. Nello stesso documento di specifica [18] C. U. Smith e C. M. Llado evidenziano queste carenze e si ripropongono di migliorare il formato secondo le indicazioni che la comunità scientifica e le ditte produttrici dei tool indicheranno.

6 Tools di risoluzione per modelli a reti di code

Esiste una grande varietà di tool di risoluzione e simulazione per modelli a reti di code. Una lista parziale può essere trovata ai seguenti indirizzi internet:

- http://www2.uwindsor.ca/~hlynka/qsoft.html lista mantenuta dal Dr. Myron Hlynka dell'Università di Windsor e;
- http://www.idsia.ch/~andrea/simtools.html lista mantenuta da Andrea Emilio Rizzoli dell'IDSIA (Istituto "Dalle Molle" di Studi sull'Intelligenza Artificiale affiliato all'università di Lugano).

Tuttavia alla prova dei fatti molti di questi tool si sono rivelati inadatti al nostro scopo. Molti di essi infatti sono basati su GUI (Graphical User Interface) XWindow o MS-Window® e non offrono la possibilità di essere invocati da linea di comando o comunque di accettare chiamate da programmi esterni. Altri sono sviluppati in linguaggi applicativi come macro di Excel™ o librerie per Mathematica™ difficilmente integrabili in altri programmi. Altri ancora sono tool commerciali non disponibili gratuitamente.

Dopo una lunga ricerca e dopo aver contattato numerose case produttrici di questi programmi, i tool che si sono rivelati più adatti al nostro scopo sono: SHARPE, QNAP2, PMVA, Pepsy/QNS, PDQ Analyzer, MVA Queueing Formalism Parser, MQNA1 ed alcuni altri tool di minore importanza.

Di seguito descriveremo alcuni dei tool utilizzati in questa tesi, dove definiamo un tool **integrabile** se:

1. tratta modelli a reti di code;
2. possiede un linguaggio (o formalismo) testuale di descrizione del modello a reti di code;
3. può funzionare in modalità batch (cioè può essere invocato dalla riga di comando senza bisogno che vi sia interazione con l'utente), accettando in input un modello a rete di code e fornendo in output il risultato della sua valutazione.

Un tool si definisce invece **non integrabile** se viene a mancare una delle tre caratteristiche suddette.

Un tool è invece **integrato** se è integrabile e:

4. si ha la disponibilità del codice sorgente o, in alternativa, dell'eseguibile per Windows/DOS o Unix/Linux.

Infine un tool viene detto **non integrato** se non è stato possibile integrarlo per altri motivi che saranno spiegati nella descrizione del tool che faremo di seguito.

6.1 SHARPE (Integrato):


prof. Kishor S. Trivedi

SHARPE [38][39][40] è un tool sviluppato dal prof. Kishor S. Trivedi, del CACC (Center for Advanced Computing and Communication), Department of Electrical and Computer Engineering, Duke University USA. SHARPE è un acronimo che significa: Symbolic Hierarchical Automated Reliability and Performance Evaluator ed è senza dubbio dopo QNAP il più importante tra i tool qui menzionati. SHARPE è un tool software che analizza modelli stocastici, pensato per essere utilizzato da tre gruppi di utenti: ingegneri, ricercatori in performance/reliability modelling e studenti dei corsi di ingegneria e scienze.

SHARPE è scritto in C, ed è stato sviluppato in origine in ambiente UNIX, ma può essere compilato su qualsiasi sistema dotato di compilatore C, di librerie matematiche e di input/output standard. Esistono versioni compilate per ambiente Linux e Windows NT/2000/XP fornite su richiesta direttamente dal prof. Trivedi.

SHARPE viene utilizzato da molte università come supporto per l'insegnamento di *fault-tolerant computing*, *performance evaluation*, *reliability engineering* e probabilità applicata.

SHARPE consente di costruire ed analizzare modelli di prestazioni (*performance*), affidabilità (*reliability* inteso come: *system failure probability*), disponibilità (*availability*) e modelli di *performability* (una combinazione di reliability, availability e performance analysis). L'utente può impostare e risolvere diverse tipologie di modelli, comparare i risultati per differenti modelli dello

stesso sistema, vedere come le modifiche dei parametri del sistema influiscono sul comportamento del modello, e sperimentare tecniche di modellazione esatte o approssimate.
SHARPE dispone di un linguaggio di specifica e di analisi per i seguenti tipi di modelli:

- *reliability block diagrams* usati per *dependability analysis* (che è l'unione di *reliability*, *availability* e *safety analysis*);
- *fault trees* usati per *dependability analysis*;
- *reliability graphs* usati per *dependability analysis*;
- *series-parallel acyclic directed graphs* usati per *performance analysis*;
- *product-form (closed) queueing networks* usati per *performance analysis*;
- *Markov and semi-Markov chains* usati per *dependability* e *performability analysis*;
- *generalized stocastic Petri nets* usati per *dependability* e *performability analysis*;

Quest'ultima tipologia di modelli, è usata in SHARPE per specificare in modo coinciso modelli di Markov molto grandi. In realtà poi SHARPE provvede a convertire internamente un modello GSPN in un modello di Markov per la sua risoluzione. Questo è l'unico modello trattato da SHARPE che viene convertito in un altro tipo, tutti gli altri modelli vengono risolti con l'algoritmo di risoluzione più appropriato in base al tipo di modello senza alcuna conversione interna.
SHARPE può essere visto come la cassetta degli attrezzi a disposizione del modellista; esso prevede un linguaggio di specifica per costruire combinazioni di modelli singoli o gerarchici e per scegliere gli algoritmi di analisi del modello. Il "sistema", così come SHARPE lo vede, non deve essere per forza un'astrazione di un qualche particolare sistema del mondo reale. Per SHARPE, una catena di Markov è una catena di Markov e basta, e non deve per forza di cose essere un modello di un qualche sistema reale. Il vantaggio di questo approccio, è che l'utente si deve poter sentire libero di usare una qualsiasi combinazione valida dei modelli supportati da SHARPE. Questo, d'altro canto, vuol dire che viene lasciata all'utente la scelta del modello che più rappresenta una corretta astrazione del sistema oggetto di studio e l'interpretazione dei risultati in modo coerente con il particolare problema.
SHARPE consente di utilizzare i risultati dell'analisi di ogni tipo di modello di cui dispone, come input nella parametrizzazione di un altro tipo di modello, soggetta solo ad un controllo di validità sui parametri stessi. La "H" di SHARPE sta proprio ad indicare questa caratteristica, cioè SHARPE è gerarchico (*Hierarchical*) nel senso che l'output di un sottomodello può essere usato come input di un altro sottomodello.
Trascurando volutamente gli altri tipi di modelli, supportati da SHARPE, ci concentreremo sui modelli per reti di code in forma prodotto
SHARPE vede una rete di code come una collezione di centri di servizio (*service center*), ognuno dei quali contiene uno o più serventi (*server*) e una coda (*queue*) per contenere i lavori (*job*) che richiedono un servizio presso il centro. SHARPE supporta un sottoinsieme dei modelli a rete di code, cioè quelli che hanno una soluzione in "forma-prodotto". Inoltre, almeno nella versione attuale, SHARPE supporta solo reti di code chiuse. Non vi è quindi modo di specificare una rete di code con un nodo di ingresso e uno di uscita dalla rete, come accade in QNAP e come è previsto da PMIF 2.0. Questo vuol dire che quando un job ha finito il suo turno di servizio presso un service center, si sposta in un altro service center della rete, ma non può lasciare la rete di code. Questo risulterà leggermente limitante per quanto riguarda la risoluzione di modelli descritti con PMIF 2.0 ma ci consentirà comunque modellare un elevato numero di sistemi reali. La funzione di distribuzione per i tempi di servizio è assunta essere esponenziale. Le altre caratteristiche come: ordine in cui i job sono prelevati dalla coda per il servizio, grado di condivisione del server, uso della preemption, ecc, sono scelti da un insieme di possibilità che rendono risolvibile il modello di rete in forma-prodotto.
Una rete di code nel linguaggio di SHARPE, viene descritta specificando i service center, le probabilità di transizione dei job (probabilità di andare da un service center a l'altro), la tipologia di servizio e il service rate, ed in fine il numero di job presenti nella rete.
Consideriamo i due seguenti modelli a rete di code:

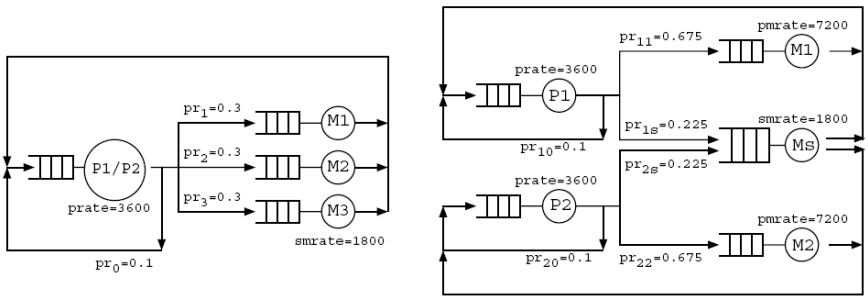


Figura 6.1 - Due modelli a Reti di Code con SHARPE

Il codice SHARPE necessario a modellare questi due sistemi è il seguente:

```
* global variable setting
        bind
                c 6
                c1 3
                c2 3
                p0 .1
                prate 3600
                pmrate 7200
                smrate 1800
        end
        * queueing network for 3-memory
        * two-processor system, design 1
        pfqn arch1-32
                * section 1: network shape
                        P1/P2 M1 (1-p0)/3
                        P1/P2 M2 (1-p0)/3
                        P1/P2 M3 (1-p0)/3
                        M1 P1/P2 1
                        M2 P1/P2 1
                        M3 P1/P2 1
                end
                * section 2: station types
                        P1/P2 ms 2,prate
                        M1 fcfs smrate
                        M2 fcfs smrate
                        M3 fcfs smrate
                end
                * section 3: # customers
                        customers c
        end
        var P32 2*prate*util(arch1-32,P1/P2)*p0
        ...
```

```
        ...
        * queueing network for 3-memory
        * two-processor system, design 2
        mpfqn arch2-11111
                * section 1: network shape and chain definition
                        chain 1
                                P1 M1 .75*(1-p0)
                                P1 Ms .25*(1-p0)
                                M1 P1 1
                                Ms P1 1
                        end
                        chain 2
                                P2 M2 .75*(1-p0)
                                P2 Ms .25*(1-p0)
                                M2 P2 1
                                Ms P2 1
                        end
                end
                * section 2: station types
                        P1 fcfs prate
                        end
                        P2 fcfs prate
                        end
                        M1 fcfs pmrate
                        end
                        M2 fcfs pmrate
                        end
                        Ms fcfs smrate
                        end
                end
                1 c1
                2 c2
        end
        var P11111 2*prate*mutil(arch2-11111,P1,1)*p0
        expr P32, P11111
end
```

SHARPE consente di avere reti di code “multi-chain”, nelle quali ci sono differenti classi (o catene) di job (o clienti). Ogni catena di job ha il suo set di probabilità di transizione (da un centro di servizio a l’altro), e i service center possono avere differenti distribuzioni dei tempi di servizio (ma non differenti tipi di servizio) per le varie classi.
SHARPE può calcolare nello stato stazionario (*steady-state*) il throughput, l’utilizzazione, il tempo medio di risposta e la lunghezza media della coda per ogni service center. Per le reti multi-chain, SHARPE può calcolare gli stessi indici per ogni singola catena.
Come già detto, SHARPE mette a disposizione diverse tipologie di modelli analitici adatti ad uno studio di performance analysis. Ognuno di questi modelli, ha delle limitazioni e vantaggi rispetto agli altri:

1. ***Series-Parallel Directed Acyclic Graphs***: Possono essere utilizzati per modellare problemi di *concurrency* e *synchronization* all’interno di programmi con risorse illimitate. Problemi con risorse limitate, non possono quindi essere modellati (dal punto di vista dello studio della *contention*) usando questa tipologia di modelli.
2. ***Product Form Queueing Networks (PFQN)***: Si dovrà ricorrere a questa tipologia di modelli se si vuole modellare l’effetto della *contention* per risorse limitate. D’altro canto in situazioni reali entrano in gioco anche problemi di concorrenza tra job, sincronizzazione, possesso simultaneo delle risorse ecc, che non possono essere modellate con questa tipologia di modelli, in quanto esse violano l’assunzione necessaria per una efficiente (*product-form*) soluzione del modello.

3. ***Markov chain***: Questa tipologia di modelli copre le carenze delle due tipologie di modelli precedenti. I modelli Markoviani forniscono un framework in grado di modellare tutte le caratteristiche delle due classi di modelli precedenti. Tuttavia, la costruzione di modelli basati su Markov chain può risultare molto complicata e non esente da errori. Ecco allora che i modelli di Petri (Generalized Stocastic Petri Nets) possono essere visti come un astrazione in grado di fornire un'interfaccia di alto livello per una concisa descrizione di questa tipologia di modelli. SHARPE genera automaticamente il corrispondente modello basato su catene di Markov, che può essere risolto usando i metodi previsti da questa classe di modelli.

Per lo studio delle performance di un sistema, SHARPE supporta quindi sei differenti tipologie di modelli:

- ***Series-parallel acyclic directed graphs;***
- ***Single or Multiple-chain Product-Form Queueing Networks;***
- ***Markov chains;***
- ***Semi-Markov chains;***
- ***Generalized Stochastic Petri Nets;***
- ***Una qualsiasi combinazione dei cinque modelli precedenti.***

Focalizzando la nostra attenzione sulle reti di code, che è l'oggetto di questa tesi, vediamo come SHARPE modella diverse tipologie di sistemi.

Central-Server Queueing System:
La Figura 6.2 mostra un esempio di sistema a coda con server centrale, in cui i job ricevono servizio dalla CPU, dopodiché attendono di ricevere servizio da uno dei due dischi e poi rientrano in coda per un altro time-slice della CPU:

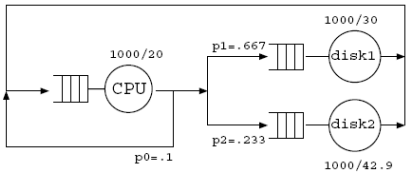


Figura 6.2 - Central-Server Queueing System con SHARPE

Il sistema modellato prevede un numero limitato di utenti (job) pari a N. Ogni server (CPU o DISK) ha tempo di servizio con distribuzione esponenziale, con service rate riportato sotto o a fianco del server stesso. Nella figura che segue abbiamo il codice SHARPE atto a modellare il sistema della figura precedente, mentre a fianco abbiamo l'output della sua esecuzione:

```
1  * central-server queueing system
2
3          bind
4                  p1 0.667
5                  p2 0.233
6          end
7          pfqn csm
8            * shape of the network
9                  cpu disk1 p1
10                 cpu disk2 p2
11                 disk1 cpu 1
12                 disk2 cpu 1
13           end
14           * fcfs servers
15                 cpu fcfs 1000/20
16                 disk1 fcfs 1000/30
17                 disk2 fcfs 1000/42.918
18           end
19           * number of jobs
20                 chain1 custs
21         end
22
23         loop i,2,10,2
24                 bind custs i
25                 expr tput(csm,cpu)
26                 expr util(csm,cpu)
27                 expr qlength (csm,cpu)
28                 expr rtime (csm,cpu)
29         end
30
31 end
```

Figura 6.3 - SHARPE Central-Server INPUT

```
i=2.000000
    custs <- 2.000000
    tput(csm,cpu): 2.9406e+01
    util(csm,cpu): 5.8811e-01
    qlength (csm,cpu): 8.2331e-01
    rtime (csm,cpu): 2.7998e-02

i=4.000000
    custs <- 4.000000
    tput(csm,cpu): 3.7976e+01
    util(csm,cpu): 7.5952e-01
    qlength (csm,cpu): 1.7202e+00
    rtime (csm,cpu): 4.5298e-02

i=6.000000
    custs <- 6.000000
    tput(csm,cpu): 4.1733e+01
    util(csm,cpu): 8.3465e-01
    qlength (csm,cpu): 2.6591e+00
    rtime (csm,cpu): 6.3717e-02

i=8.000000
    custs <- 8.000000
    tput(csm,cpu): 4.3753e+01
    util(csm,cpu): 8.7506e-01
    qlength (csm,cpu): 3.6209e+00
    rtime (csm,cpu): 8.2758e-02

i=10.000000
    custs <- 10.000000
    tput(csm,cpu): 4.4992e+01
    util(csm,cpu): 8.9983e-01
    qlength (csm,cpu): 4.5955e+00
    rtime (csm,cpu): 1.0214e-01
```

Figura 6.4 - SHARPE Central-Server OUTPUT

SHARPE prevede l'uso di variabili che possono semplificare la scrittura di modelli complessi e risultano utili per evidenziare le probabilità di transizione. La definizione di variabili avviene mediante l'uso della parola chiave "***bind***" (righe 3-6) con la sintassi:

```
bind
        <identificatore>  <valore>
        <identificatore>  <valore>
        ...
        <identificatore>  <valore>
end
```

In SHARPE un modello a reti di code in forma prodotto viene specificato mediante l'uso della parola chiave ***pfqn*** (Product Form Queueing Network) che ha la seguente sintassi:

```
pfqn <model name> { ( <parameters list>) }
        * section 1: station-to-station transition probabilities
                <station-name> <station-name> <expression>
                        ...
                <station-name> <station-name> <expression>
        end
        * section 2: station types and parameters
                <station-name> <scheduling discipline> <expression> [,<expression>, ...]
                        ...
                <station-name> <scheduling discipline> <expression> [,<expression>, ...]
        end
        * section 3: number of customers per chain
                <chain name> <expression>
end
```

La rete di code relativa al sistema viene specificata in tre sezioni del codice SHARPE, come ad esempio nella Figura 6.3 ad iniziare dalla riga 8. La prima sezione (righe 9-13) specifica la topologia della rete di code e le probabilità di transizione tra le varie stazioni che la compongono. Ogni riga è composta da tre elementi: stazione di origine, stazione di arrivo, probabilità che un job passi dalla prima alla seconda stazione dopo aver ricevuto servizio dalla prima. La seconda sezione (righe 15-18) specifica i parametri di servizio per le stazioni. Questa sezione deve contenere una riga per ogni stazione con le seguenti informazioni: nome stazione, disciplina di servizio e uno o più parametri di servizio dipendenti dalla disciplina di servizio adottata. Nel nostro caso avendo tutte le stazioni la disciplina FIFO (*first-come-first-served*) dovremo specificare un solo parametro di servizio che è il service rate.

La terza ed ultima sezione (righe 20-21) definisce il numero di job presenti nella rete di code. Questa sezione è costituita da una riga contenente un identificatore arbitrario seguito dal numero di job, che a sua volta può essere una variabile definita altrove. Nel nostro esempio si è usato l'identificatore arbitrario "*chain1*" per indicare che siamo in presenza di una rete di code a catena singola (*single-chain*); nel caso di reti di code multi catena, avremmo dovuto specificare una riga per ogni catena con un relativo identificatore (es. *chain1*, *chain2*, …, *chainN*) .
SHARPE prevede quattro "misure" (tutte *steady-state*), fornite da altrettante funzioni, per ogni server della PFQN: ***throughput*** (***tput***), ***utilization*** (***util***), ***response time*** (***rtime***) e ***queue length*** (***qlength***). Nell'esempio viene variato il numero di job da 2 a 10 per vedere come il numero di job alteri queste misure per la sola CPU. L'invocazione delle funzioni di misura avviene mediante la parola chiave "***expr***" seguita dal nome della funzione con i relativi parametri, che sono il nome del modello (nel nostro caso "***csm***") e dal nome dalla stazione da valutare. Notare anche l'uso della parola chiave "***bind***" per attualizzare il valore della variabile "***custs***" con il valore della variabile "***i***" del loop.
Come si vede dalla figura 7.4, SHARPE visualizzerà i risultati per ogni step del ciclo riportando i valori per ***tput***, ***util***, ***qlength*** e ***rtime***, al crescere del numero dei job.

Per quanto riguarda la definizione delle discipline di scheduling, SHARPE supporta le seguenti sei tipologie di servizio:

1. <station name> **is** <rate> : Definisce una stazione come "infinite server" ; ogni job quando arriva al server ha un service-time dato da una funzione di distribuzione cumulativa (in inglese abbreviata in **CDF** da Cumulative Distribution Function) esponenziale con parametro <rate>.
2. <station name> **fcfs** <rate> : Definisce una stazione con disciplina di servizio First Come First Served (altrimenti conosciuta come FIFO: First In First Out); i job che arrivano nella coda sono serviti uno alla volta; il job servito ha un service-time dato da una CDF esponenziale di parametro <rate>.
3. <station name> **ps** <rate> : Definisce una stazione di tipo Processor Sharing. Quando nella stazione ci sono *n* job, ogni job ha un tempo di servizio dato da una CDF esponenziale di parametro <rate>/*n*.
4. <station name> **lcfspr** <rate> : Definisce una stazione con algoritmo di scheduling del tipo Last Come First Served Pre-emptive Resume, il cui service-rate è dato sempre da una CDF esponenziale di parametro <rate>.
5. <station name> **ms** <number of servers>, <rate> : Definisce una stazione di tipo Multiple Server; il numero di server è dato dall'espressione <number of servers>. Ogni server ha il medesimo service rate dato da una CDF esponenziale di parametro <rate>.
6. <station name> **lds** [<rate> | <loop>], [<rate> | <loop>], … : Definisce una stazione con un server, il cui service rate dipende dal numero di job presenti nella stazione. La parola chiave lds è seguita da una serie di parametri di tipo <rate> o <loop>. Un <rate> è come nei casi precedenti un'espressione che definisce una CDF esponenziale di parametro <rate>, mentre un <loop> ha la forma: **loop**(*index*, *low*, *high*, *increment*, *expression*) con la quale si definiscono una serie di <rate>, cioè un loop viene espanso in una serie di rate. Dopo l'espansione di un loop, il primo <rate> si applica al caso in cui nella stazione vi sia un solo job, il secondo al caso in cui ci siano due job e così via. Se ci sono meno rate rispetto al massimo numero di job, l'ultimo rate presente nella riga è assunto come default per tutti job che superano il massimo definito in modo implicito dai rate.

Terminal-Oriented System:
Consideriamo come esempio il seguente sistema, con M terminali, una CPU e due dischi:

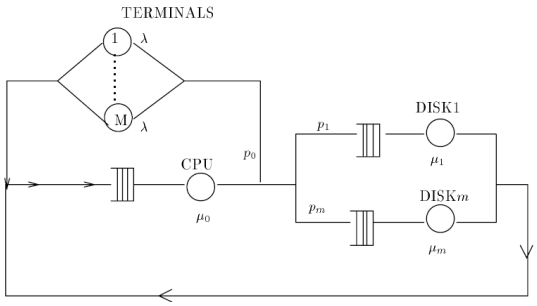


Figura 6.5 - Esempio di sistema con terminali

Il codice SHARPE necessario a modellare la rete di code precedente (per semplicità si è fissato il numero di dischi m = 2) è il seguente:

```
* central-server queuing network with terminals
        bind
                p1 0.667
                p2 0.233
                p0 1-(p1+p2)
                lambda 1/25
        end
        pfqn csm(jobs)
                * routing probabilities...
                        cpu terminals p0
                        cpu disk1 p1
                        cpu disk2 p2
                        disk1 cpu 1
                        disk2 cpu 1
                        terminals cpu 1
                end
                * servers
                        cpu FCFS 1000/20
                        disk1 FCFS 1000/30
                        disk2 FCFS 1000/42.918
                        terminals IS lambda
                end
                * number of jobs
                        chain1 jobs
        end
        func resp(i) qlength( csm, cpu; i ) + qlength( csm, disk1; i ) + qlength( csm, disk2; i )  / tput( csm, terminals; i )
        loop i,10,40,10
                expr tput( csm, cpu; i )
                expr util( csm, disk1; i )
                expr qlength( csm, disk2; i )
                expr resp( i )
        end
end
```

Figura 6.6 - Codice SHARPE per Terminal-Oriented System

L'esecuzione in ambiente DOS di questo codice SHARPE produce il seguente output:

```
C:\SHARPE>sharpe terminalcs.pfqn

i=10.000000
    tput(csm, cpu;i):  3.9164e+000
    util(csm, disk1;i):  7.8367e-002
    qlength(csm, disk2;i):  4.0588e-002
    resp(i):  2.7212e-001

i=20.000000
    tput(csm, cpu;i):  7.8195e+000
    util(csm, disk1;i):  1.5647e-001
    qlength(csm, disk2;i):  8.4441e-002
    resp(i):  4.7489e-001

i=30.000000
    tput(csm, cpu;i):  1.1706e+001
    util(csm, disk1;i):  2.3423e-001
    qlength(csm, disk2;i):  1.3192e-001
    resp(i):  7.1640e-001

i=40.000000
    tput(csm, cpu;i):  1.5570e+001
    util(csm, disk1;i):  3.1156e-001
    qlength(csm, disk2;i):  1.8343e-001
    resp(i):  1.0084e+000

C:\SHARPE>
```

Figura 6.7 - SHARPE Output per Terminal-Oriented System

Da notare la definizione della funzione "***resp***" mediante l'uso della parola chiave "***func***", cha calcola il response-time del sistema.

Multi-Chain Product Form Queueing Networks:
Le reti di code cha abbiamo visto in precedenza sono tutte "*single-chain*". Per *single-chain* (catena singola), intendiamo che tutti i job che circolano nella rete si comportano allo stesso modo, relativamente alle probabilità di routing e alle caratteristiche del servizio richiesto dalle stazioni. Se vogliamo che la nostra rete di code sia popolata da job con differenti comportamenti, dobbiamo usare le reti di code multi-chain. I job raggruppati per comportamento si dicono "*jobs classes*".

In SHARPE una rete di code multi-chain si definisce mediante l'uso delle parole chiave: **mpfqn** e **chain**, una rete di code multi-chain ha quindi la seguente sintassi generale:

```
mpfqn <model name> { ( <parameters list>) }
        * section 1: station-to-station transition probabilities
        chain <chain-name>
                <station-name> <station-name> <expression>
                        ...
                <station-name> <station-name> <expression>
        end
        ...
        [ chain <chain-name>
                <station-name> <station-name> <expression>
                        ...
                <station-name> <station-name> <expression>
        end ]
        * section 2: station types and parameters
        <station-name> <scheduling discipline> [<expression> [,<expression>, ...] | <number of servers>]
                [<chain name> <expression>
                        ...
                <chain name> <expression>]
        end
        ...
        [ <station-name> <scheduling discipline> [<expression> [,<expression>, ...] | <number of servers>]
                [<chain name> <expression>
                        ...
                <chain name> <expression>]
        end ]
        * section 3: number of customers per chain
                <chain name> <expression>
                        ...
                <chain name> <expression>
end
```

Per quanto riguarda i parametri aggiuntivi relativi alle varie discipline di servizio vale quanto già detto precedentemente, per le reti di code single-chain.

Consideriamo la seguente rete di code:

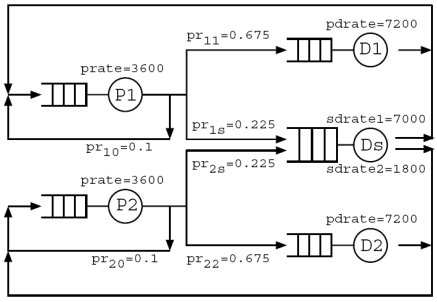


Figura 6.8 - Multi-Chain Queueing Network

Immaginiamo che in tale rete circolino due classi di job; nella prima, che chiameremo "***chain 1***", i job hanno a disposizione una loro CPU privata (**P1**) e un loro disco dedicato (**D1**), ma possono anche

accedere ad un disco condiviso con l'altra classe (**Ds**) per poi tornare alla CPU (**P1**) perché ad esempio hanno terminato il loro lasso di CPU o perché hanno finito il loro lavoro. La seconda classe di job, che chiameremo "***chain 2***", si comporta come la prima ma con la differenza che utilizza la CPU **P2** in modo esclusivo, così come il disco **D2**, e condivide con i job della prima classe il disco **Ds**.
Come si vede dall'immagine, relativamente al disco condiviso **Ds,** la rete di code si comporta in modo differente per le due classi di job. Infatti per la prima classe si ha un rate di servizio ad alta velocità con un rate **sdrate1 = 7000**, per i job che vengono eseguiti nella CPU **P1**; mentre assegna un rate più basso per i job eseguiti sulla seconda CPU **P2**, pari a **sdrate2 = 1800**.

Il codice SHARPE necessario a modellare questa rete di code multi-chain è il seguente:

```
* Multi-Chain Queueing Network

        bind
                pr11 0.675
                pr1s 0.225
                pr22 0.675
                pr2s 0.225
                prate 3600
                pdrate 7200
                sdrate1 7000
                sdrate2 1800
        end

        mpfqn serve2( c )
                * SECTION 1: station to station transition probabilities for each chain
                                chain C1
                                        P1 D1 pr11
                                        P1 Ds pr1s
                                        D1 P1 1
                                        Ds P1 1
                                end

                                chain C2
                                        P2 D2 pr22
                                        P2 Ds pr2s
                                        D2 P2 1
                                        Ds P2 1
                                end
                end

                * SECTION 2: station types and parameters
                                        P1 fcfs prate
                                end
                                        P2 fcfs prate
                                end
                                        D1 fcfs pdrate
                                end

                                        D2 fcfs pdrate
                                end
                                        Ds ps
                                        C1 sdrate1
                                        C2 sdrate2
                                end
                end
                * SECTION 3: number of customer per chain
                        C1 c/2
                        C2 c/2

        end

        loop c , 10 , 40 , 10
                expr mqlength ( serve2 , Ds ; c )
                expr mqlength ( serve2 , Ds , C1 ; c )
                expr mqlength ( serve2 , Ds , C2 ; c )
        end

end
```

Ricapitolando, SHARPE supporta le seguenti funzioni (***built-in***) per l'analisi di reti di code single e multi-chain:

- **tput**(*system_name*, *station* {;*arg_list*}) ⇒ Restituisce il throughput per una stazione di una rete di code in forma prodotto (**PFQN**) single-chain.
- **rtime**(*system_name*, *station* {;*arg_list*}) ⇒ Restituisce il tempo medio di risposta per una stazione di una PFQN single-chain.
- **qlength**(*system_name*, *station* {;*arg_list*}) ⇒ Restituisce la lunghezza media della coda di una singola stazione di una PFQN single-chain.
- **util**(*system_name*, *station* {;*arg_list*}) ⇒ Restituisce l'utilizzazione della stazione specificata di una PFQN single-chain.
- **mtput**(*system_name*, *station* {, *chain*} {;*arg_list*}) ⇒ Restituisce il throughput per una stazione di una PFQN multi-chain. Se si specifica una particolare catena viene restituito il throughput solo per quella specifica catena, altrimenti viene restituita la somma su tutte le catene.
- **mrtime**(*system_name*, *station* {, *chain*} {;*arg_list*}) ⇒ Restituisce il tempo medio di risposta per una stazione di una PFQN multi-chain. Se si specifica una particolare catena viene

restituito il response-time medio solo per quella specifica catena, altrimenti viene restituita la somma su tutte le catene.

- **mqlength**(*system_name*, *station* {, *chain*} {;*arg_list*}) ⇒ Restituisce la lunghezza media della coda di una singola stazione di una PFQN multi-chain. Se si specifica una particolare catena viene restituita lunghezza media della coda solo per quella specifica catena, altrimenti viene restituita la somma su tutte le catene.
- **mutil**(*system_name*, *station* {, *chain*} {;*arg_list*}) ⇒ Restituisce l'utilizzazione della stazione specificata di una PFQN multi-chain. Se si specifica una particolare catena viene restituita l'utilizzazione media per quella specifica catena, altrimenti viene restituita la somma su tutte le catene.

Per una descrizione più dettagliata e specifica delle altre caratteristiche di SHARPE si rimanda al manuale utente [38][39].

6.2 Pepsy/QNS™ (Integrato):


Dr. Gunter Bolch

Tool sviluppato dal Dr. Gunter Bolch della Friedrich-Alexander University of Erlangen-Nuremberg (http://www4.informatik.uni-erlangen.de/~bolch/). PEPSY-QNS è l'acronimo di Performance Evaluation and Prediction System for Queueing NetworkS. Questo interessante tool nasce all'inizio come tool per la verifica e la validazione degli algoritmi classici di analisi di modelli a reti di code, poi evolve in un sistema completo per la creazione e risoluzione di modelli a reti di code. PEPSY implementa ad oggi circa 50 differenti algoritmi di analisi e simulazione. PEPSY-QNS è composto dal sistema di base – chiamato PEPSY – e da un front-end grafico XPEPSY (per l'ambiente grafico X11-Windows-System).

PEPSY è composto da tre elementi distinti, progettati per funzionare in modo congiunto. Questi tre elementi sono:

- un modulo per l'inserimento (input) interattivo del modello (*eingabe*: dal tedesco immissione, immettere, ingresso),
- un modulo per la selezione guidata del miglior algoritmo di analisi (*auswahl*: dal tedesco scelta),
- un modulo di analisi (*analyse*: dal tedesco analisi).

PEPSY-QNS è stato progettato per funzionare in ambiente UNIX senza troppo riguardo all'ambiente DOS/Windows. D'altronde nel 1990 Windows non si poteva ancora considerare un sistema operativo degno di tale nome. Per questo, i sorgenti originali non prevedono alcuna opzione di compilazione per un porting in questo ambiente, né per Linux in quanto nel '90 non si era ancora affermato come S.O. di massa alternativo a Windows. Tuttavia con molto sforzo e qualche compromesso, alla fine si è riusciti ad ottenere la compilazione del modulo di analisi per Linux, che alla fine era quello che ci serviva per il nostro web service. La maggiore difficoltà nell'ottenere una compilazione stabile, che funzioni almeno in ambiente Linux, è rappresentata dal fatto che nei sorgenti i commenti, i nomi delle variabili e delle funzioni sono scritti in Tedesco.

PEPSY-QNS può gestire reti di code con uno o più classi aperte o chiuse o reti miste. Le specifiche delle informazioni di routing possono essere date mediante visit ratios o transition probabilities, ma i due tipi di specifica non possono essere mischiati per classi differenti. Le informazioni di routing devono essere specificate per singola classe separatamente e non sono ammessi class switching. Per quanto riguarda i service center, PEPSY prevede che la specifica del loro tipo venga data usando la notazione di Kendall, i tipi di service center supportato sono quindi:

- M/M/m-FCFS
- M/G/1-PS
- M/G/.-IS
- M/G/1-LCFS
- ./G/m-FCFS (preemptive e nonpreemptive priority policies)
- M/M/m-FCFS-ASYM
- M/G/m-FCFS-ASYM

Dato che PMIF 2.0 non consente di specificare il tipo del service center usando la notazione di Kendall, bisognerà dedurla, in fase di conversione, usando le sole informazioni previste da PMIF.

Abbiamo già detto che PEPSY supporta circa 50 metodi di analisi e risoluzione per modelli a reti di code, questi algoritmi sono classificati in categorie che comprendono:

- State Probabilities (Markov Analysis): Richiedono molto tempo computazionale, ma sono di grande importanza perché consentono di calcolare misure di performance esatte.

- Normalization Constant Computation: Ricadono in questa sezione l'algoritmo di Convoluzione ed il REcursion by Chain ALgorithm (RECAL). Questo tipo di algoritmi sono usati per il calcolo di reti di code chiuse in forma-prodotto.
- MVA based: Sono compresi l'algoritmo MVA Esatto, l'algoritmo di Bard e Schweitzer, SCAT (Self Correction Approximation Techinique) ed altri algoritmi che sono migliorie ed estensioni dei precedenti.
- Analysis by Decomposition: Sono algoritmi per il calcolo di reti non BCMP. Sono disponibili: l'algoritmo di Raymond Marie, di Kühn, di Whitt, Pojoulle, Gelenbe, Chylla e il Summation Method individualmente o integrati in un unico metodo di decomposizione universale che usa alcune tecniche euristiche per il calcolo dei valori di performance per le stazioni di servizio.
- Product-Form Approximation Techniques: Diffusion approximation method di Reiser/Kobayashi e l'EPF-method di SHUM.
- Simulation: PEPSY integra anche un modulo di simulazione ad eventi discreti che può essere usato come gli altri metodi analitici. L'unica differenza è il tempo impiegato per il calcolo dei risultati.

Per quanto riguarda l'output dei risultati, PEPSY prevede lo stesso formato per tutti i metodi supportati. Questo rappresenta un vantaggio da non sottovalutare e che spesso altri tool non hanno, perché consente di confrontare facilmente i valori ottenuti dai vari metodi disponibili. Le informazioni riportate da ogni metodo di risoluzione sono, per ogni nodo:

- throughput,
- visit ratio,
- average service time,
- utilization,
- average response time,
- average number of jobs,
- average waiting times, e
- average queueing length.

Queste misure sono poi sintetizzate:

- per ogni classe in modo separato, e
- come somma su tutte le classi.

I valori di performance per l'intera rete di code vengono calcolati per ogni job class in modo separato e comprendono:

- throughput,
- average response time,
- average number of jobs.

Descrizione ed inserimento di una rete di code:

Il primo passo quando si usa PEPSY è la creazione di un modello. La componente di PEPSY-QNS che si occupa di questo compito è il programma: "**`eingabe`**". Questo "comando" consente di effettuare un inserimento guidato di un modello a rete di code mediante una maschera interattiva. Il programma *eingabe* chiede all'utente le seguenti informazioni:

- numero e tipo di job-classes,
- numero di nodi,
- il tipo di ogni nodo,
- il service rate,
- le routing information.

A proposito delle routing information bisogna dire che, parlando di open job-classes, PEPSY-QNS dispone di un nodo speciale source/sink (combinato). Questo nodo ovviamente non è assolutamente necessario per le classi chiuse ma deve comunque essere presente. PEPSY-QNS usa questo nodo "esterno" come reference node per il calcolo delle misure di performance relative. Per esempio quando vengono specificate le transition probabilities, le frequenze delle visite dei nodi della rete saranno calcolate impostando automaticamente il visit ratio per il nodo esterno a 1.
Una volta specificate queste informazioni, *eingabe* chiederà all'utente il numero di jobs presenti nella rete ed il nome da dare al modello. Rispetto al nome inserito PEPSY/eingabe creerà un file con un prefisso "e_", quindi ad esempio se il nome che abbiamo dato al modello è "first" sul disco verrà creato un file "e_first".
A differenza degli altri tool studiati, PEPSY usa dei file di input molto descrittivi e senza parole chiavi particolari. Il file di input è composto da solo testo (parole chiavi in Inglese o Tedesco) suddiviso in varie sezioni con delle tabelle ASCII.

Esempio:

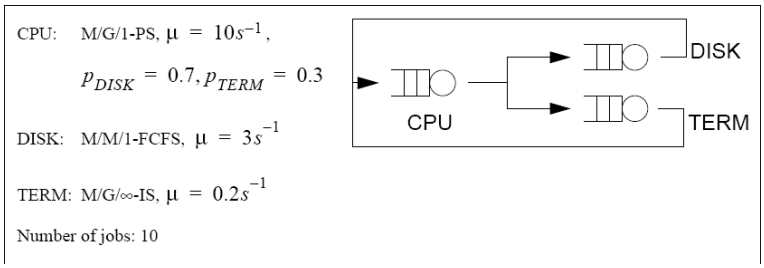


Figura 6.9 - Modello a rete di code di Central Server Computer System

```
#
# filename  e_first
#

NUMBER NODES: 3
NUMBER CLASSES: 1

NODE SPECIFICATION

  node   |      name          |     type
---------+--------------------+--------------------
     1   |      CPU           |     M/G/1-PS
     2   |      DISK          |     M/M/1-FCFS
     3   |      TERM          |     M/G/0-IS

CLASS SPECIFICATION

  class   | arrival rate   number of jobs
----------+---------------------------------
     1    |     -             10

CLASS SPECIFIC PARAMETERS

CLASS  1

     node           |  service_rate    squared_coeff._of_variation
--------------------+---------------------------------
     CPU            |  10              1
     DISK           |  3               1
     TERM           |  0.2             1


SWITCHING PROBABILITIES

from/to    | outside    CPU        DISK       TERM
-----------+-----------------------------------------------
outside    | 0.000000  1.000000  0.000000  0.000000
CPU        | 0.000000  0.000000  0.700000  0.300000
DISK       | 1.000000  0.000000  0.000000  0.000000
TERM       | 1.000000  0.000000  0.000000  0.000000
```

Figura 6.10 - File *e_first* generato da *eingabe*

La fase successiva consiste nella scelta del metodo di analisi. PEPSY-QNS supporta l'utente nella scelta del o dei metodi applicabili mediante un data-base interno contenente le limitazioni degli algoritmi. Il comando/programma del sistema PEPSY-QNS atto a questo scopo è: "**`auswahl`**". La sintassi di questo comando è:

```
auswahl <nome modello>
```

Automaticamente verrà caricato il file "e_<nome modello>" e dopo averlo analizzato verrà fornita una lista di algoritmi applicabili ed algoritmi applicabili ma con limitazioni o che necessitano di parametri aggiuntivi, come ad esempio il metodo sim2, che è un metodo di risoluzione basato sulla simulazione, il quale necessita che venga specificata la durata massima della simulazione e l'accuratezza desiderata.

Esempio:

```
first:
    usable                |  need further specification
----------------------------------------------------------
    ammva                 |
    bol_aky               |
    bounds                |
    cmva                  |
                          |  hm
    marie                 |
    mmva                  |
    monosum               |
    multisum              |
    num_app               |
    num_single            |
    pm_2                  |
    priomva2c             |
    priomva2m             |
    recal                 |
                          |  sim2
```

Figura 6.11 - Lista dei metodi applicabili al modello fornita dal comando *auswahl first*

Il passo finale consiste nell'analisi vera e propria. Il comando/programma che esegue l'analisi in PEPSY-QNS è: "**`analyse`**". La sintassi di questo comando è:

```
analyse <metodo> <nome modello>
```

Esempio:
Usiamo il metodo ***`marie`*** per la risoluzione del modello. La sintassi del comando sarà:

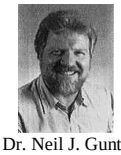
```
analyse marie first
```

ed il risultato fornito da PEPSY-QNS:

```
PERFORMANCE_INDICES FOR NET:  first

description of the network is in file 'e_first'

the closed net was analysed with method 'marie'  .


jobclass 1

marie      |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
CPU        |   3.922   1.000   0.100   0.392   0.156   0.613   0.056   0.221
DISK       |   2.746   0.700   0.333   0.915   1.276   3.503   0.943   2.588
TERM       |   1.177   0.300   5.000   0.000   5.000   5.883   0.000   0.000



characteristic indices:

marie      |  lambda     mvz     maa
-----------+--------------------------
           |   3.922   2.549  10.000


legend

e    : average number of visits            mue    : service rate
rho  : utilisation                         lambda: mean throughput
mvz  : average response time
maa  : average number of jobs
mwz  : average waiting time
mwsl: average queue-length
```

Figura 6.12 - Le misure di performance del modello "*first*" calcolate con il metodo *marie*

6.3 PDQ Analyzer (Integrato):


Dr. Neil J. Gunther

Sviluppato da Neil J. Gunther (Performance Dynamics Company[SM] http://www.perfdynamics.com/), è una **libreria di funzioni** open source, originariamente scritta in C, abbinata al libro dello stesso autore: "The Practical Performance Analyst" [34]. Successivamente Gunther ha tradotto la libreria in PERL ed ha abbinato la sua uscita alla nuova edizione del libro: "Analyzing Computer System Performance with PERL::PDQ" [35].

PDQ è l'acronimo di "Pretty Damn Quick" che tradotto significa letteralmente: "Carino Dannatamente Veloce". Gunther nelle sue due edizioni ci tiene a specificare che PDQ è un risolutore di circuiti a rete di code e non un simulatore. PDQ incorpora come metodo di risoluzione l'algoritmo Mean Value Analysis (MVA). PDQ non si può definire "un'applicazione" ma piuttosto una libreria di funzioni per l'analisi di sistemi a rete di code. La versione originale scritta in standard C e l'ultima scritta in PERL sono per loro natura facilmente portatili e ricompilabili potenzialmente su qualsiasi sistema HW/SW.

Dato che PDQ è una libreria di funzioni, è ovvio che per creare un modello e per risolverlo c'è bisogno di un minimo di programmazione e di conoscenza del C o del PERL.

PDQ usa il concetto di **circuito di code** (*circuit of queues*), mentre nella letteratura formale della teoria delle code un sistema di code viene spesso chiamato *queueing network*. Chiaramente, come Gunther stesso dice nel suo libro, le parole ***circuit*** e ***network*** sono in questo contesto interscambiabili. L'uso del termine queueing network deriva da un retaggio storico in cui i modelli a rete di code venivano ampiamente usati nelle reti dati (*data networks*). Gunther, invece, preferisce mettere in evidenza le similitudini con altre discipline come: ***control theory*** e ***signal processing***.

Le similitudini si possono sintetizzare come segue:

- i circuiti (***circuits***) implicano dei flussi (***flows***), come ad esempio elettroni o richieste;
- i circuiti hanno ***input*** e ***output*** ben definiti;
- i circuiti possono essere combinati in serie (***series circuits***), in parallelo (***parallel circuits***);
- i circuiti possono essere partizionati in sottocircuiti (***subcircuits***);
- i sottocircuiti possono essere raggruppati per semplificare le tecniche di risoluzione;
- i circuiti possono includere ***feedback*** (***returned output***) che impongono dei cicli chiusi (***closed loop***).

In accordo con questa terminologia, ogni programma PDQ deve avere un ben definito insieme di input ed un ben definito insieme di output. Gli input possono essere ad esempio il traffico, la popolazione di utenti attivi, i tassi di servizio. Questi dati possono essere collezionati da un'analisi del sistema oggetto di studio oppure stimati, nel caso non si disponga del sistema reale. Il ruolo di PDQ è appunto quello di fornire come output un insieme di metriche di prestazione come: utilizzazione delle risorse, lunghezza della coda e tempo di residenza.

Creare un modello con PDQ è molto facile e si concretizza nei seguenti sei passi di programmazione:

1. Definire ogni istanza di una coda nel circuito di code, attraverso una chiamata alla funzione **`CreateNode()`**;
2. Per ogni circuito aperto (***open circuit***), definire il flusso di traffico (***traffic stream***) attraverso una chiamata alla funzione **`CreateOpen()`**;
3. Per ogni circuito chiuso (***closed circuit***), definire il carico di lavoro (***workload stream***) per ogni utente batch o interattivo attraverso una chiamata alla funzione **`CreateClosed()`**;
4. Specificare la domanda di servizio (***service demand***) per ogni workload definito, per ogni queueing center nel circuito, attraverso una chiamata alla funzione **`SetDemand()`**;
5. Risolvere il modello attraverso una chiamata alla funzione **`Solve()`**, specificando come parametro il metodo di risoluzione;
6. Visualizzare un report con i dati elaborati da PDQ attraverso una chiamata alla funzione **`Report()`**.

Una limitazione di PDQ è che, almeno nella versione attuale, gestisce solo centri di servizio a singolo servente. Nei sorgenti C della libreria ci sono tracce di costanti e funzioni che fanno pensare al tentativo di eliminare questa limitazione ma ad oggi non sono disponibili versioni (C o PERL) della libreria che implementano questa estensione.

6.3.1 Realizzazione di PDQ Shell e PDQ Model Language

Come libreria C, PDQ non sarebbe stato integrabile. Avendo a disposizione i sorgenti C e PERL si è deciso quindi di convertirli in PHP5 e di scrivere un tool (PDQsh.php) che accettasse in ingresso uno pseudo linguaggio e che fornisse in output il risultato della risoluzione del modello specificato. La versione PHP5 della libreria è stata quindi inviata al prof. Gunther, che ha provveduto alla pubblicazione sul suo sito.
Descrivere tutta la libreria PDQ.php non è possibile in quanto composta da più di 2900 righe di codice. I sorgenti completi della libreria e del tool possono essere trovati nel CD allegato a questa tesi mentre la documentazione può essere reperita sul sito http://www.perfdynamics.com/. Qui ci limiteremo a descrivere il linguaggio del tool.

Il tool è stato chiamato “PDQsh” cioè “PDQ Shell”. Questo nome è stato scelto perché, una volta eseguito senza parametri aggiuntivi sulla linea di comando, si comporta come una Shell. Viene visualizzato un prompt ed è possibile fornire dei comandi nel suo pseudo linguaggio.
I comandi ricordano fedelmente le funzioni di libreria di PDQ, e devono essere usati nello stesso ordine. I comandi disponibili sono visualizzabili mediante il comando HELP e EXTHELP.
All’avvio, la shell visualizza un breve messaggio di benvenuto e mostra il prompt di default:

```
C:\PDQ>pdqsh.bat

PDQ (Pretty Damn Quick) Performance Analyzer Shell V 1.0
Created by Samuel Zallocco - University of L'Aquila - ITALY
Using Library: PDQ Analyzer v3.0 111904

Type HELP for available commands description.

PDQ> _
```

Specificando sulla riga di comando il flag “-s” seguito dal nome di un file, la shell funzionerà in modalità batch (o stream mode). Il file dovrà quindi contenere il modello specificato in PDQ language con la sequenza dei comandi da eseguire.

I comandi o costrutti del linguaggio utilizzabili sono:

INIT <sps>
Inizializza l’ambiente PDQ, <sps> è il nome da assegnare al modello. In modalità interattiva, cambia il prompt aggiungendo il nome del modello.

CREATEOPEN <s> <f>
Permette di definire un open workload, <s> è il nome da assegnare al workload ed <f> è l’arrival rate per unità di tempo.

CREATECLOSED <s> <BATCH|TERM> <f1> <f2>
Permette di definire un closed workload, <s> è il nome del workload, <f1> è il numero di richieste o visite, <f2> è il delay o think-time che deve trascorrere prima che una richiesta rientri nel circuito.

CREATENODE <s> <CEN|DLY> <FCFS|FIFO|ISRV|IS|LCFS|LIFO|PSHR|PS>
Consente di definire un nodo della rete, specificando il nome: <s> il tipo: CEN = centro di accodamento o DLY = centro di ritardo ed in fine la disciplina di servizio.

SETDEMAND <s1> <s2> <f>
Consente di specificare la domanda di servizio. <s1> è il nome del nodo, <s2> è il nome assegnato al workload, <f> è la domanda di servizio (espressa in unità di tempo) richiesta dal workload quando transita attraverso il nodo.

SETVISITS <s1> <s2> <f1> <f2>
Consente di specificare il numero di visite che il workload <s2> effettuerà al nodo <s1>. <f1> è il numero di visite (adimensionale), <f2> è il tempo di servizio che il workload richiede al nodo (in unità di tempo).

SETWUNIT <s>
Consente di specificare una stringa <s> per dare un nome alle unità di lavoro (il default è "Job").

SETTUNIT <s>
Consente di specificare una stringa <s> per dare un nome alle unità di temp (il default è "Sec").

SETDEBUG <TRUE|FALSE>
Attiva o disattiva la modalità di debug della libreria PDQ.

SOLVE <APPROX|CANON|EXACT>
Risolve il modello definito con le istruzioni precedenti, utilizzando uno dei tre metodi supportati dalla libreria PDQ.

GETRESPONSE <TRANS|TERM|BATCH> <s>
Stampa il system response time per il workload <s>.

GETTHRUPUT <TRANS|TERM|BATCH> <s>
Stampa il system throughput time per il workload <s>.

GETUTILIZATION <s1> <s2>
Stampa l'utilizzazione del nodo <s1> rispetto al workload <s2>.

REPORT
Stampa un report completo della risoluzione del modello.

HELP
Visualizza un help in inglese con la descrizione di questi comandi.

EXTHELP
Visualizza un help in inglese sui comandi estesi.

EXIT
Esce da PDQ Shell e torna al sistema operativo.

[SN|SETNAME] <sps>
Cambia il nome del modello, impostandolo a <sps> senza reinizializzare PDQ.

PROMPT <ON|OFF|DEFAULT|COMPLETE|<sps> >

Consente di visualizzare o nascondere il prompt, cambiarlo impostandolo a <sps>, oppure reimpostarlo al default "PDQ> ", infine consente di completare un prompt con il simbolo "> " aggiungendo uno spazio dopo il “maggiore”.

[MESSAGE|MESSAGES|SHOWMESSAGE] <ON|OFF>
Mostra o nasconde i messaggi di esecuzione della shell di PDQ.

[ERROR|ERRORS|SHOWERROR] <ON|OFF>
Mostra o nasconde gli errori di esecuzione della shell di PDQ.

[ECHO|PRINT] <sps>
Stampa la stringa <sps> senza andare a capo.

[ECHON|PRINTN] <sps>
Stampa la stringa <sps> e va a capo.

[VER|VERSION]
Stampa la versione della libreria PDQ in uso.

[VAR|SET|LET|INT|INTEGER|DOUBLE|REAL|FLOAT] <s> = <f>
Consente di definire una variabile numerica chiamata <s> ed inizializzata a <f>. In PDQ Shell tutte la variabili sono trattate internamente come valori reali.

[GETVAR|PRINTV|PRINTVAR|VARSHOW|SHOWVAR] <s>
Stampa la variabile <s> senza andare a capo.

[HIST|HISTORY|SHOWHIST]
Stampa la cronologia dei comandi eseguiti con successo.

[CLEARHIST|CLEARHISTORY]
Cancella la cronologia dei comandi.

VARSTAT <s>
Mostra tutte le variabili definite ed il loro valore attuale.

[SREP|SMARTREPORT|SREPORT]
Stampa un report compatto della risoluzione del modello.

ERLANG <f1> <f2>
Stampa un report con il calcolo delle funzioni Erlang-B e Erlang-C per <f1> servers e <f2> traffic intensity.

REPAIR <f1> <f2> <f3> <f4>
Calcola la soluzione esatta per M/M/m/N/N repairmen model dove <f1> = numero di servicemen, <f2> = tempo medio di servizio, <f3> = numero di macchine, <f4> = Mean time to failure (MTTF).

Alcuni comandi, anzi quasi tutti, hanno degli alias o abbreviazioni:

- **[CO|COPEN]** aliases di **CREATEOPEN**
- **[CC|CCLOSED]** aliases di **CREATECLOSED**
- **[CN|CNODE]** aliases di **CREATENODE**
- **[SD|SDEMAND]** aliases di **SETDEMAND**

- **[SV|SVISIT]** aliases di **SETVISIT**
- **[SW|SWU|SWUNIT|SETWORKUNIT]** aliases di **SETWUNIT**
- **[ST|STU|STUNIT|SETTIMEUNIT]** aliases di **SETTUNIT**
- **[DBG|SDBG|SDEBUG]** aliases di **SETDEBUG**
- **[?|H]** aliases di **HELP**
- **[??|HH]** aliases di **EXTHELP**
- **[XIT|END|BYE|LOGOUT|QUIT]** aliases di **EXIT**
- **COMPUTE** alias di **SOLVE**
- **GR** alias di **GETRESPONSE**
- **GU** alias di **GETUTILIZATION**
- **GT** alias di **GETTHRUPUT**
- **REP** alias di **REPORT**

Esistono delle variabili di sistema, ereditate dalla libreria PDQ e accessibili dalla Shell, che sono:

- CREATEOPENCOUNT: Contiene il numero di open workload definiti;
- CREATECLOSEDCOUNT: Contiene il numero di closed workload definiti;
- CREATENODECOUNT: Contiene il numero di nodi definiti;
- RESPONSE: Contiene il response time del sistema;
- THRUPUT: Contiene il Throughput del sistema;
- UTILIZATION: Contiene l'utilizzazione del sistema;
- NODES: Come CREATENODECOUNT, contiene il numero di nodi definiti, ma punta direttamente alla variabile omonima della libreria PDQ: $PDQ_nodes;
- STREAMS: Contiene il numero totale di workload definiti siano essi aperti o chiusi. Questa variabile, come NODES, punta direttamente alla omonima variabile presente nella libreria PDQ: $PDQ_streams;

Infine alcune precisazioni:

– **<sps>** = É una stringa che può contenere spazi, specificata anche mediante i doppi apici, come ad esempio: "io sono una stringa";
– **<s…>** = É una stringa che non deve contenere spazi al suo interno;
– **<f…>** = É un numero a virgola mobile o intero, oppure è il nome di una variabile definita in precedenza;

Quando si usa PDQsh in modalità interattiva, allora la Shell visualizzerà sempre i suoi messaggi di risposta con un doppio simbolo: ">>" tutti gli altri messaggi di risposta che non iniziano per ">>" provengono dalla libreria PDQ, sono cioè messaggi interni della libreria previsti ed implementati dal prof. Gunther.

6.4 MVA Queueing Formalism Parser (Integrato):


Dr. C. Chereddi

Tool sviluppato dal Dr. Chandrakanth Chereddi, Department of Electrical & Computer Engineering, and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. Si tratta di un piccolo tool scritto in C, sviluppato come parte di un lavoro di studio di alcuni algoritmi approssimati per la risoluzioni di modelli a reti di code chiuse in forma-prodotto (separabili), basati sull'MVA [36]. Pur nella sua semplicità, questo tool è stato ritenuto interessante per il gran numero di algoritmi implementati. Più che di un tool per la risoluzione di modelli a reti di code, si tratta di una libreria di algoritmi con una interfaccia di input/output ed un formalismo per la descrizione della rete di code. Gli algoritmi implementati da Chereddi sono:

- Exact MVA Algorithm
- Large Customer Population Algorithm (LCP)
- Proportional Estimation Algorithm (PE)
- Queue Line Algorithm (QL)
- Fraction Line Algorithm (FL)
- Aggregate Queue Length Algorithm (AQL)
- Proportional Approximation Method (PAM)
- Proportional Approximation Method Improved (PAMI)

Per testare questi algoritmi, Chereddi ha dovuto implementare un parser in grado di accettare in input un formalismo scritto in uno pseudo-linguaggio tabellare. Questo formalismo è stato chiamato "queueing formalism" ed il parser "queueing formalism parser", e anche se Chereddi non ha dato un nome al suo tool in questa tesi ci riferiremo ad esso come "MVA Queueing Formalism Parser".

Un formalismo può contenere numeri e commenti. I commenti sono linee di testo che iniziano con il simbolo "#". I numeri sono così interpretati:

- La prima riga che viene incontrata, contenente un numero intero positivo, è l'ID del formalismo.
- La riga successiva, contenente due numeri separati da uno spazio viene interpretata come segue. Il primo numero rappresenta il numero di catene (chains) o classi (Chereddi, nel suo lavoro usa come sinonimi intercambiabili i termini "multiple customer classes" e "routing chains"). Il secondo numero rappresenta il numero di service center (o servers).
- La successiva riga di numeri deve contenere le popolazioni, separate da spazi, da assegnare ad ogni classe. Quindi ci devono essere tanti numeri interi positivi diversi da zero, per quante sono le classi.
- Segue una serie di numeri reali positivi (anche zero), che rappresentano il Think Time per Customer Class.
- Il formalismo si conclude con una serie di righe che rappresentano una matrice di dimensione: "server"-righe x "class"-colonne, contenente dei valori reali che esprimono il service demand per customer class per service center.

Un file di input può contenere anche più di un formalismo e l'elaborazione si conclude quando il file è finito o quando si incontra la parola riservata "END".

Esempio:

```
# Formalism – 10        <- Commento
10                      <- ID del formalismo
2 3                     <- Numero di classi Numero di service center
# Population            <- Altro commento
2 2                     <- Popolazione per classe
# Think time            <- Altro commento
92.076 19.953           <- Think Time per customer class
# Service demands       <- Altro commento
7.106 16.156 5.905      <- Service Demand relativi alla prima classe
11.678 10.078 13.243    <- Service Demand relativi alla seconda classe
# **Done**              <- Commento
END                     <- Fine dei formalismi
```

Una caratteristica del tool di Chereddi è l'espandibilità. Per implementare un nuovo algoritmo è sufficiente scrivere una funzione che rispetti il seguente prototipo:

```
[algorithm_name](IN qf *fm, OUT mvals *mv)
```

dove *qf* è una struttura con la seguente definizione:

```
typedef struct queueing_formalism {
      /* Queueing formalism Identifier */
      unsigned int id;
      /* Population vector of size 'dim' 1D */
      unsigned int *p;
      /* Number of customer classes */
      unsigned int dim;
      /* No of servers/service centers */
      unsigned int servers;
      /* Service demand C x K matrix */
      float **sd;
      /* Think time of size 'dim' 1D */
      float *think_time;
} qf;
```

e *mval* è una struttura con la seguente definizione:

```
typedef struct mean_values {
      /* Queueing Formalism Identifier */
      unsigned int id;
      /* ID of the MVA algorithm used */
      unsigned int algo_id;
      /* Mean residence time C x K matrix */
      float **res_time;
      /* Mean queue length C x K matrix */
      float **mq_len;
      /* Throughput - 1D Vector */
      float *tput;
} mvals;
```

6.4.1 Modifica del tool per consentire il naming dei service center e delle classi

Dato che il tool non consentiva di nominare né le classi né i service center, e data la disponibilità del codice sorgente, distribuito da Chereddi con licenza GNU General Public License, si è deciso di modificare leggermente il formato di input e la routine del parser per consentire la specifica dei nomi degli oggetti presenti in una rete di code. Questa modifica è stata implementata sfruttando i commenti, estendendo il parser per renderlo in grado di rilevare alcune parole riservate presenti dopo il simbolo di commento "#". La sintassi del formalismo esteso prevede i seguenti meta comandi:

- **M**: consente di dare un nome al modello, la sintassi esatta è la seguente:
  `#M "nome del modello"`
- **C**: consente di dare un nome alle classi, la sintassi generale è la seguente:
  `#C "numero-classe" "nome-classe"`
- **S**: consente di dare un nome ai service center
  `#S "numero-service-center" "nome-service-center"`

Questa estensione, sfruttando il tag di commento, ci consente di creare modelli retro-compatibili con la versione originale distribuita da Chereddi.

Esempio:

```
# This is a MVACCKSW Model:
#  - Name: ClosedQueueExample
#  - Description: SIMPLE CLOSED QUEUEING PMIF 2.0 MODEL
#  - Date-Time: 2006-09-02T22:09:04
# Generated from the original PMIF 2.0 Model using PMIF2_to_ MVACCKSW.xsl
# that was part of the Queueing Network Solver Webservice Project
#
# Model Type: Single-Chain Closed Queueing Network Model
#
#M "ClosedQueueExample"  // MODEL NAME

# CLASS DEFINITION
#  NUM  NAME
#C "0" "ACCESS"

# NODE DEFINITION
#  NUM  NAME
#S "0" "CPU"
#S "1" "DISK1"
#S "2" "DISK2"

# Queueing formalism number:
1
# Number-of-Classes Number-of-Servers
1 3
# Population
12
# Think time
1.56
# Service demands for Class
# SERVER1 ... SERVERn
2.34 4.08 0.585
# **Done**
END
```

A differenza di altri tool il formalismo di MVACCKSW non consente di specificare la scheduling policy dei nodi, mentre prevede la possibilità di specificare il Think Time per ogni classe di clienti.

Questo apre un problema pratico relativamente a come trattare il centro di servizio che funge da Think Device, e nella fase di conversione da PMIF al formalismo del tool si dovrà decidere come convertire questa informazione.

Ad Esempio consideriamo il seguente modello PMIF 2.0:

```
<?xml version="1.0"?>
<QueueingNetworkModel Name="ClosedQueueExample" Description="SIMPLE CLOSED QUEUEING PMIF 2.0 MODEL"
Date-Time="2006-08-30T09:08:32" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.perfeng.com/pmif/pmifschema.xsd">
  <Node>
    <Server Name="CPU" Quantity="1" SchedulingPolicy="IS"/>
    <WorkUnitServer Name="DISK1" Quantity="1" SchedulingPolicy="FCFS" ServiceTime="0.12"
                                                                        TimeUnits="Sec"/>
    <WorkUnitServer Name="DISK2" Quantity="1" SchedulingPolicy="FCFS" ServiceTime="0.015"
                                                                        TimeUnits="Sec"/>
  </Node>
  <Arc FromNode="CPU" ToNode="DISK1" Description="Arc From CPU To DISK1"/>
  <Arc FromNode="DISK1" ToNode="CPU" Description="Arc From DISK1 To CPU"/>
  <Arc FromNode="CPU" ToNode="DISK2" Description="Arc From CPU To DISK2"/>
  <Arc FromNode="DISK2" ToNode="CPU" Description="Arc From DISK2 To CPU"/>
  <Workload>
    <ClosedWorkload WorkloadName="ACCESS" NumberOfJobs="12" ThinkTime="1.56" TimeUnits="Sec"
                                                                        ThinkDevice="CPU">
      <Transit To="DISK1" Probability="0.5"/>
      <Transit To="DISK2" Probability="0.5"/>
    </ClosedWorkload>
  </Workload>
  <ServiceRequest>
    <WorkUnitServiceRequest WorkloadName="ACCESS" ServerID="DISK1" NumberOfVisits="1">
      <Transit To="CPU" Probability="1"/>
    </WorkUnitServiceRequest>
    <WorkUnitServiceRequest WorkloadName="ACCESS" ServerID="DISK2" NumberOfVisits="1">
      <Transit To="CPU" Probability="1"/>
    </WorkUnitServiceRequest>
  </ServiceRequest>
</QueueingNetworkModel>
```

Come si può notare il centro di servizio CPU funge da Think Device, questo modello può essere convertito nei seguenti due modi:

1° Modo:

```
#M "ClosedQueueExample"  // MODEL NAME
# CLASS DEFINITION
#  NUM  NAME
#C "0" "ACCESS"
# NODE DEFINITION
#  NUM  NAME
#S "0" "CPU"
#S "1" "DISK1"
#S "2" "DISK2"
# Queueing formalism number:
1
# Number-of-Classes Number-of-Servers
1 3
# Population
12
# Think time
1.56
# Service demands for Class
# SERVER1 ... SERVERn
0.0 0.12 0.015
# **Done**
END
```

2° Modo:

```
#M "ClosedQueueExample"  // MODEL NAME
# CLASS DEFINITION
#  NUM  NAME
#C "0" "ACCESS"
# NODE DEFINITION
#  NUM  NAME
#S "0" "DISK1"
#S "1" "DISK2"
# Queueing formalism number:
1
# Number-of-Classes Number-of-Servers
1 2
# Population
12
# Think time
1.56
# Service demands for Class
# SERVER1 ... SERVERn
0.12 0.015
# **Done**
END
```

Il primo modo sembra il più corretto perché lascia traccia nel modello della presenza del nodo CPU.

6.5 PMVA for Windows (Integrato):

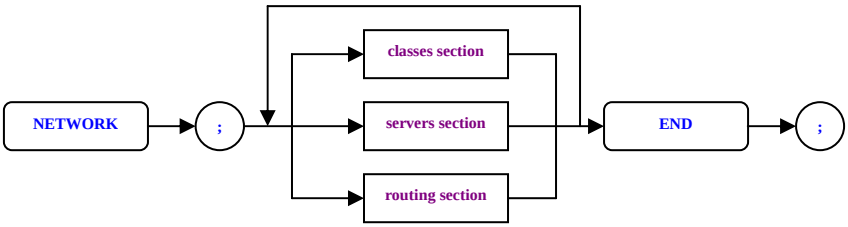

Dr. Greg Brewster

É un piccolo tool, sviluppato nel 1981 da Jeff Brumfield (Dep. Of Computer Sciences, Perdue University, Indiana) che ci è stato fornito dal Dr. Greg Brewster (DePaul University – School of Computer Science, Telecommunications and Information System).
Come recita l'abstract del manuale utente [49], PMVA è un programma per l'analisi di modelli a reti di code chiuse in forma-prodotto. Un modello viene dato in input al programma usando un linguaggio di descrizione della rete di code basato su keywords. Questo linguaggio permette di dare una concisa, ma facilmente leggibile, descrizione della rete di code. Le misure di performance della rete possono essere calcolate invocando uno dei molti algoritmi analitici basti sull'analisi MVA di cui dispone. Una caratteristica di PMVA è quella di poter variare alcuni parametri della rete di code a runtime, e di poter rivalutare la rete con questi cambiamenti usando ad esempio tecniche differenti. Il programma, di cui si dispone solo dell'eseguibile per ambiente DOS/Windows, è stato scritto in PASCAL e si compone di una routine di input del modello, di diverse routine di risoluzione, e di una routine di output.

Il keyword language di PMVA

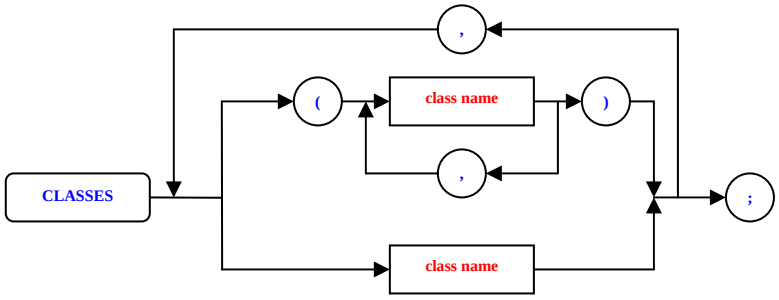
Diamo la sintassi del linguaggio di PMVA così come riportata nel manuale utente:

Comando NETWORK:



Il comando NETWORK è usato per definire un modello a rete di code. La descrizione della rete viene validata e convertita in una rappresentazione interna per essere usata con i vari algoritmi di risoluzione. Quando in un file si incontra un'altra istruzione NETWORK, la definizione precedente viene persa.
L'ordine con cui si descrivono le sezioni non è importante, ma è importante definire tutte le classi e i centri di servizio prima di utilizzarli in altre sezioni. Generalmente si preferisce specificare prima la class section, poi la servers section ed in fine la routing section.

La sezione CLASSES:



Le diverse classi devono avere nomi differenti, almeno nei primi 10 caratteri. L'ordine in cui vengono specificate le classi influenza l'ordine con cui verranno stampate le misure di performance. Se la sezione CLASSES viene omessa, allora PMVA assumerà che la rete di code sia a singola classe.
L'uso delle parentesi serve per raggruppare le classi in catene (*chain*); le classi che fanno parte della stessa catena hanno utenti che possono effettuare un cambiamento di classe (*class switching*). Il class switching può essere definito anche in modo implicito all'interno della ROUTING section.

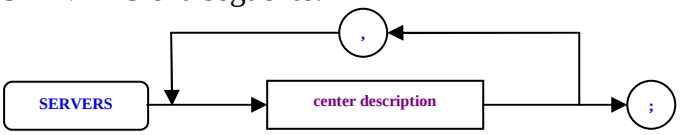
Esempi:

1. `CLASSES BATCH, TSO, TPS;`
2. `CLASSES A, B, C;`
3. `CLASSES (A), (B), (C);`
4. `CLASSES A, (B, C, D), (E, F);`

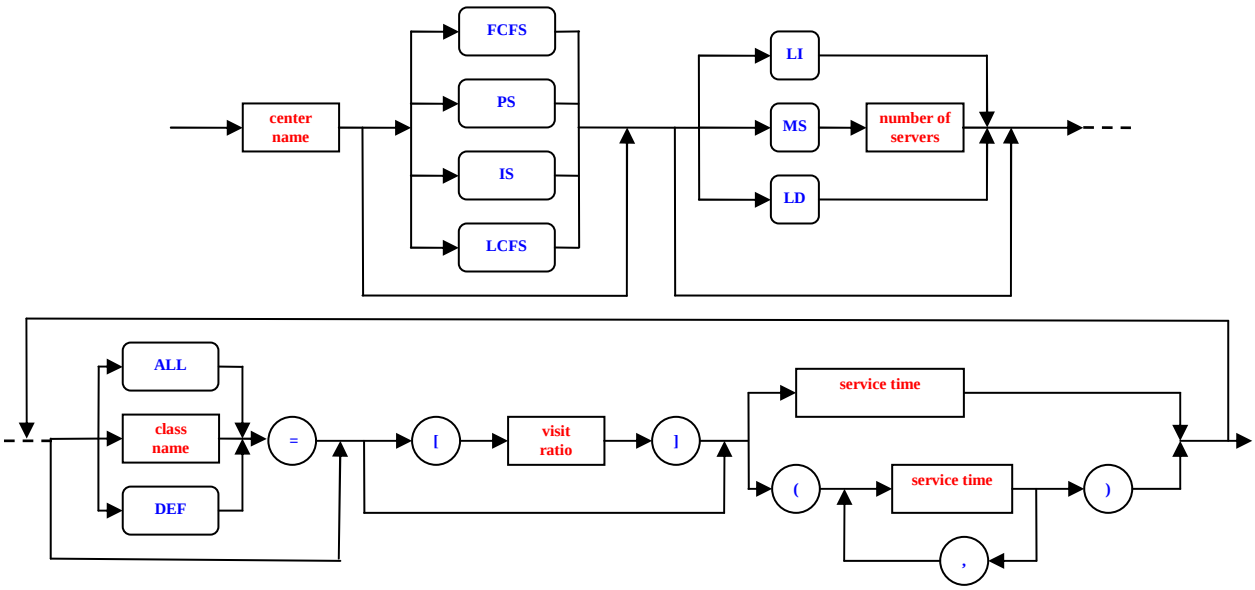
   I primi tre esempi definiscono tre classi di clienti distinte, senza class switching (a meno che esso venga definito in modo implicito nella ROUTING section). L'ultimo esempio definisce sei classi e tre catene, la prima è composta dalla sola classe di clienti A i cui clienti non possono effettuare alcun class switch, la seconda catena è composta dalle classi B, C e D, i cui clienti possono saltare da una classe a l'altra, l'ultima catena è composta dalle classe E ed F anche questa con class switching dei clienti tra una classe e l'altra.

La sezione SERVERS:

La sintassi della sezione SERVERS è la seguente:



La sintassi di ogni center description è invece la seguente:



Come per i nomi delle classi, anche il nome del service center deve essere unico nei primi 10 caratteri. L'ordine in cui si specificano i service center influenza l'ordine di stampa delle misure di performance. Le discipline di servizio disponibili sono le solite First Come First Served (FCFS), Processor Sharing (PS), delay o Infinite Server (IS) e, cosa rara rispetto agli altri tool analizzati, Last

Come First Served preempt/resume (LCFS). Un service center può contenere uno o più serventi. Il tempo medio di servizio dei clienti può o non dipendere dal numero dei clienti presenti al centro di servizio. Sono supportati quindi i seguenti tre tipi di servizio:

- Load Independent (LI), sono centri di servizio dotati di un singolo server. Il tempo di servizio è indipendente dalla lunghezza della coda.
- Multiple-Server (MS), sono centri di servizio con due o più server indipendenti dal carico (LI).
- Load Dependent (LD), sono centri di servizio con un singolo server. Il tempo di servizio dei clienti dipende dal numero totale di clienti presenti nel centro. Quando si parla di dipendenza dal numero di clienti ci si riferisce a clienti di qualsiasi classe. Mentre non è supportata la distinzione tra classi di clienti.

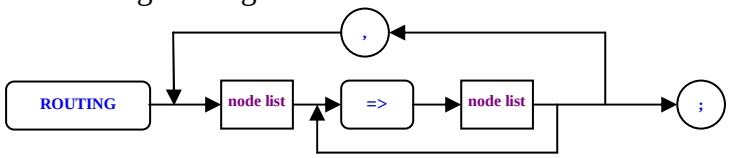
Se viene omessa la specifica del tipo di service center, allora PMVA assumerà come default LI.
Come restrizione, si ha che gli Infinite Server (IS) devono per forza di cose essere Load Independent (LI). La specifica dei tempi di servizio (e, come opzione, del *visit ratio*) deve essere specificata per ogni classe di clienti. Ovviamente tutti i nomi di classe utilizzati devono preventivamente essere stati definiti nella CLASSES section. Se c'è un'unica classe di clienti o se la CLASSES section è stata omessa allora la classe (ed il simbolo di uguale) possono essere omessi. Se i tempi di servizio sono uguali per tutte le classi che visitano il service center allora può essere utilizzata un unica parola ALL al posto dei nomi delle classi, in questo modo i tempi di servizio che seguono il segno di uguale vengono applicati a tutte le classi. Se, invece, i tempi di servizio sono uguali per alcune classi, ma non per tutte, si può utilizzare la parola chiave DEF al posto della prima definizione di classe. Inizialmente vengono assegnati i tempi specificati a tutte le classi, ma poi è possibile ridefinire singolarmente le classi che hanno service time differenti. Il numero di visite (*visit ratios*) devono essere specificati per tutti i service center e per tutte le classi o per nessuno di essi. Se il numero di visite non viene specificato verrà calcolato in base a quanto definito nella ROUTING section. Per quanto riguarda i tempi di servizio: per i centri LI e MS, deve essere specificato il tempo medio di servizio di un cliente della classe; per i centri MS, questo tempo viene interpretato come il tempo medio richiesto ad uno dei server; per i centri LI, devono essere specificati i tempi di servizio nel caso ci siano 1, 2, 3, …,n clienti presenti in coda. Se ci sono più clienti nel centro rispetto a quanto specificato, si assume che esso rimanga costante all'ultimo specificato. La dipendenza dal carico per i tempi di servizio deve rimanere proporzionale rispetto a tutte le classi. Per un centro FCFS, il tempo di servizio deve essere uguale per tutte le classi che visitano il centro. Se si conosce il service rate invece del service time, allora il service time può essere specificato come 1.0/rate.

Esempi:

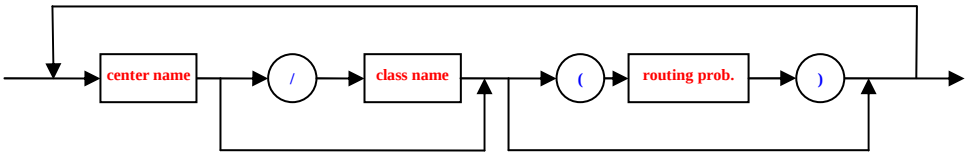
1. ```
   TERMINALS      IS   LI 4.000,
   CPU            PS   LI 0,025,
   DISK           FCFS LI 0.050,
   TAPE           FCFS LI 0.150;
   ```
2. ```
   CPU PS MS 2 BATCH=0.064 TIMESHARE=0.037;
   ```
3. ```
   DISK FCFS LD ALL=(0.0250, 0.0225, 0.0214, 0.0208, 0.0203, 0.0200);
   ```
4. ```
   DISK FCFS LI DEF=0.050 RELOCATE=0.150;
   ```

La sezione di ROUTING:

La sintassi della sezione di routing è la seguente:



dove ogni node list e dato da:



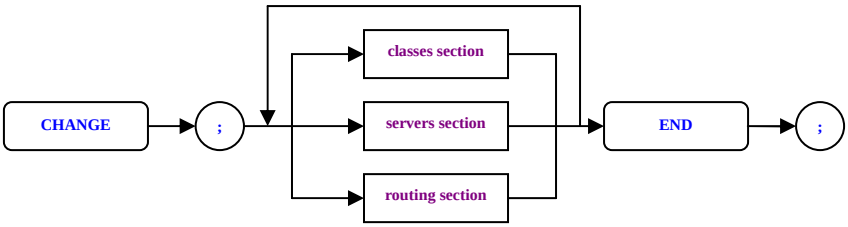
I clienti sono instradati da ogni combinazione di service-center/classe presente sul lato sinistro del simbolo "=>" verso ogni service-center/classe presente sul lato destro in accordo con la probabilità specificata con la coppia service-center/class del lato destro (destinazione). Ogni probabilità deve essere un numero reale compreso tra 0.0 e 1.0. Per ogni classe di clienti, il totale della somma delle probabilità di routing rispetto ai service center visitati deve essere pari a 1.0. Tutte le probabilità non specificate vengono assunte come essere pari a 0.0. Le probabilità possono essere specificate anche come frazione con la sintassi num1/num2 (Es. 20/100). Le probabilità di routing devono essere specificate per ogni combinazione di service-center/classe di un node list o per nessuno di essi. In quest'ultimo caso si assume che la probabilità sia uniforme. Cioè, ogni probabilità di routing viene impostata al valore di default dato dal reciproco del numero di elementi nella lista.

Esempi:

1. `CPU => DRUM (0.7) DISK (0.2) => CPU;`
2. `CPU => CPU (0.1);`
3. `CPU/BATCH => DRUM (0.3) DISK (0.5) TAPE (0.2) => CPU`
4. `MEM1 MEM2 MEM3 MEM4 => MEM1 MEM2 MEM3 MEM4;`

Modifica di un modello:

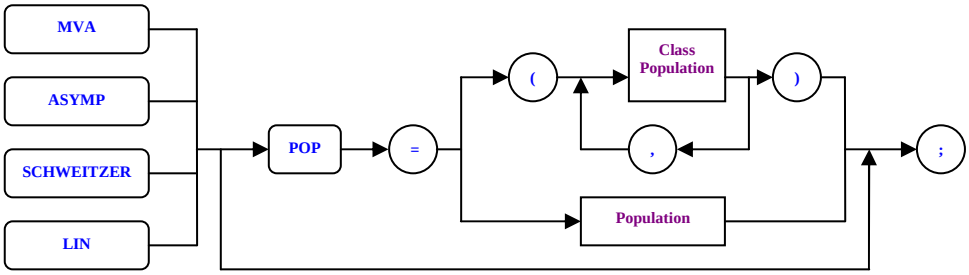
Un modello può contenere anche una sezione di modifica la quale deve avere la seguente sintassi:



L'uso del comando CHANGE, può essere utile per valutare un modello con diversi parametri e valori. La sintassi del comando CHANGE e delle sue diverse sezioni è la stessa del comando NETWORK.

Analisi di un modello a reti di code

Quando una rete di code è stata definita, è possibile procedere alla sua analisi mediante uno o più algoritmi di risoluzione. La sintassi del comando di risoluzione è la seguente:



Come si può notare, gli algoritmi disponibili sono:

- MVA: invoca l'algoritmo Mean Value Analysis esatto.
- ASYMP: invoca l'algoritmo di Bard per reti con grandi popolazioni di clienti (*Large Customer Population*). Si tratta di una versione iterativa dell'MVA che richiede meno tempo di calcolo.
- SCHWEITZER: invoca l'algoritmo di Schweitzer. Anche questo è una versione iterativa dell'MVA che richiede minor tempo computazionale.
- LIN: invoca l'algoritmo Linearizer di Chandy e Neuse. Una versione migliorata dell'MVA che fornisce risultati migliori di ASYMP e SCHWEITZER. Mentre per piccole popolazioni di clienti è meglio usare la soluzione esatta MVA.

I clienti di ogni classe devono essere specificati tra parentesi nello stesso ordine in cui sono state definite le classi nella CLASSES section. Se invece c'è una sola classe di clienti basta specificare la popolazione senza parentesi.

Nel file di comando, possono essere specificate più invocazioni per algoritmi differenti e per ogni invocazione viene calcolato:

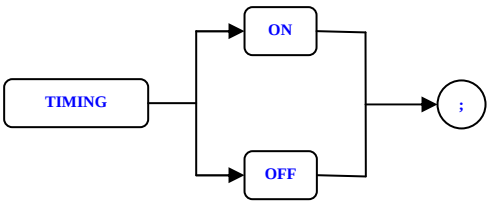
- throughput;
- mean waiting time;
- mean queue length;
- utilization.

Per le reti multiclasse vengono visualizzate le misure riferite sia alle singole classi che all'intero sistema.

Altri comandi

Comando TIMING:

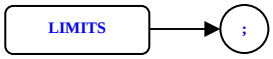
Sintassi:



Il comando TIMING abilita o disabilita la stampa del tempo di CPU richiesto per l'esecuzione di ogni comando.

Comando LIMITS:

Sintassi:



Il comando LIMITS stampa i limiti relativi al numero massimo di nodi, classi, catene, ecc, stabiliti in fase di compilazione.

Ad esempio, l'eseguibile DOS/Windows in nostro possesso il comando LIMITS riporta i seguenti valori:

- max number of customer classes = 11
- max number of chains = 11
- max number of service centers = 22
- max number of load independent centers = 22
- max number of multiple server centers = 3
- max number of load dependent centers = 3
- max servers at any multiple server center = 10
- max number of stages = 242
- max errors reported per line = 10
- max descriptors per routing list = 10
- sing test in linear sys solver = 0.000010

limiti dell'algoritmo MVA:

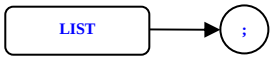
- max population if load dependent centers = 15
- max number of states saved = 400

limiti degli algoritmi approssimati:

- convergence criterion = 0.000100
- max number of iterations = 100

Comando LIST:

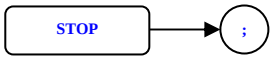
Sintassi:



LIST produce una descrizione, sotto forma di tabella, della rete di code corrente.

Comando STOP:

Sintassi:



Termina l'esecuzione del programma. Deve sempre essere presente, pena il blocco di qualsiasi client che invochi il web service.

6.6 QSolver/1 - OPENQN e CLOSEDQN (Integrati)

Figura 6.13 - Daniel A. Menasce, Virgilio A. F. Almeida, and Larry W. Dowdy

QSolver/1 è un tool per capacity planning abbastanza datato, che funziona su PC con sistema operativo MS-DOS o compatibile. Questo tool è una versione ridotta (supporta un massimo di 3 workloads, 8 device) di un tool commerciale sviluppato da Daniel A. Menasce, Virgilio A. F. Almeida e Larry W. Dowdy del Department of Computer Science della George Mason University – Fairfax VA. Basandosi sui parametri di input che descrivono l'ambiente di calcolo, questo tool calcola le misure di performance corrispondenti a quell'ambiente di calcolo. I progetti di capacity planning spesso contengo la domanda: "cosa succede se?". Ad esempio, consideriamo il caso in un manager IT che vuole sapere quale sarà il tempo di risposta se il numero di terminali connessi ad un sistema di calcolo transazionale è incrementato del 30%? Usando un tool come QSolver/1, risulta più facile rispondere alla domanda "cosa succede se?", cambiando opportunamente i parametri di input.

QSolver/1 lavora con tre entità: devices, workloads e domain. I devices sono usati per specificare i componenti della configurazione del sistema di calcolo, che sono in grado di svolgere un lavoro indipendente. Per esempio, processori, dischi, unità di controllo, unità a nastro e scheda di rete possono tutti essere configurati come devices, mentre altri componenti come memoria e terminali hanno una differente specifica, legata alla definizione delle classi di workloads. I workloads rappresentano tutte le richieste di processo (jobs, transactions, commando, ecc) inviate al sistema dall'utente durante un dato periodo di tempo. Per rappresentare le varie collezioni di componenti eterogenei, QSolver/1 consente al capacity planner di specificare differenti classi di workloads. Ogni classe viene specificata mediante il suo tipo, il quale definisce il suo modo di processo: batch, transaction e terminal. Le classi di workloads Interactive o Terminals rappresentano una classe di processi in linea (*online*) le cui componenti sono generate da un dato numero di terminali o workstation con un dato think time. Questa classe di workloads viene definita da paramenti come: numero di terminali, average think time. Le classi di workloads di tipo Transaction rappresentano una classe di processi online che raggruppano componenti che arrivano nel sistema con un dato intervallo (*rate*). Quindi, questa classe è ben definita dall'arrival rate. Le classi di workloads di tipo Batch, si riferiscono a componenti eseguiti in modalità batch (*offline*), questa classe può quindi essere descritta mediante il numero di jobs attivi nel sistema. QSolver/1 consente la definizione di shared domain e di priorità per la definizione delle classi di workloads. In un modello di performance multiclasse, si possono avere P gruppi di priorità. Diverse classi possono condividere la stessa priorità relativa. La priorità di una classe è un numero nel range 1,…,P e si assume che la priorità più alta sia la 1, mentre essa decresce al crescere del numero di priorià.

L'interfaccia utente di QSolver/1 è di tipo testuale semigrafico e purtroppo non vi è alcun modo di controllare questo programma da linea di comando, per cui risulta essere non integrabile. Tuttavia, nel pacchetto distribuito su internet, che è lo stesso che si trova allegato nel libro degli stessi autori [55], sono presenti altri due sorgenti di programmi scritti in Borland™ Pascal, chiamati CLOSEDQN.PAS e OPENQN.PAS, rispettivamente per la risoluzione di modelli a rete di code chiuse e aperte. Questi programmi, una volta compilati, accettano in input un formalismo per la descrizione della rete di code, e forniscono in output una dettagliata analisi delle performance. La particolarità di questi due tool è che accettano reti di code con server di tipo dipendenti dal carico (*load dependent*). Anche se PMIF 2.0 non consente di specificare server o workunit server di tipo load dependent, questa caratteristica, che potrebbe essere implementata in versioni future di PMIF, insieme con il fatto di avere i sorgenti ed il fatto che i due tool possono essere invocati da linea di comando, li ha resi dei buoni candidati ad essere integrati nel nostro webservice.

Il formalismo per la descrizione del modello a rete di code accettato da CLOSEDQN è abbastanza spartano e non consente di dare un nome ai server o ai workload ed alle classi. Un esempio di file di input per CLOSEDQN è il seguente:

```
 4  3       K RR
 10  20  5  Vector_N
-1  0       Device 1 type (delay):  client workstation
 1  35      Device 2 type (LD): network
    1.000
    0.716
    0.668
    0.648
    0.636
    0.629
    0.623
    0.620
    0.617
    0.614
    0.613
    0.611
    0.610
    0.609
    0.608
    0.607
    0.606
    0.605
    0.605
    0.604
    0.604
    0.603
    0.603
    0.603
    0.602
    0.602
    0.602
    0.601
    0.601
    0.601
    0.601
    0.601
    0.600
    0.600
    0.600
  0  0     Device 3 type (LI)
  0  0     Device 4 type (LI)
>>>> Service Demand Matrix
  40         20         15
  0.00016    0.000417   0.001269
  0.030      0.255      0.615
  0.036      0.306      0.738
```

La prima riga deve contenere due valori interi separati da uno spazio. K rappresenta il numero di devices (al massimo si possono definire 50 device), RR il numero di classi (al massimo si possono definire 20 classi). La seconda linea contiene il vettore popolazione per ogni classe RR e la somma totale della popolazione deve essere inferiore o uguale a 100. Seguono poi un numero di righe pari al numero K di devices. Ogni riga deve contenere due numeri, il primo definisce il tipo di devices, dove:

- 0 = Load Independent Device (LI);
- 1 = Load Dependent Device (LD);
- -1 = Delay Device (DELAY).

Il secondo numero ha senso solo per device LD e rappresenta il fattore di saturazione per la popolazione che visita quel device. Il numero massimo di devices LD ammessi è pari a 5, e per questo tipo di device si deve, di seguito, specificare un numero di service rates (un valore reale per riga) pari al valore della saturazione specificata. Per tutti gli altri tipi di device si deve mettere il valore di saturazione pari a zero.

Segue una riga obbligatoria contenente un commento qualsiasi, commento che può opzionalmente essere posto anche dopo ognuna delle definizioni viste in precedenza, come nell'esempio riportato sopra. Dopo il commento si deve specificare la matrice dei Service Demand. Questa matrice è composta da numeri reali ed ha una dimensione pari a K righe e RR colonne.

L’output ottenibile da CLOSEDQN è del tipo seguente:

```
ClosedQN - (c) Copr. 1994 D. Menasce', V. Almeida, and L. Dowdy.
           All Rights Reserved.
This program comes with the book 'Capacity Planning and
Performance Modelling: from mainframes to client-server systems'
by Menasce, Almeida, and Dowdy published by Prentice Hall.
+++ Iteration:   1   Error:   0.628603
+++ Iteration:   2   Error:   0.260851
+++ Iteration:   3   Error:   0.096008
+++ Iteration:   4   Error:   0.037921
+++ Iteration:   5   Error:   0.015457
+++ Iteration:   6   Error:   0.006421
+++ Iteration:   7   Error:   0.002703
+++ Iteration:   8   Error:   0.001150
+++ Iteration:   9   Error:   0.000494
+++ Iteration:  10   Error:   0.000214
+++ Iteration:  11   Error:   0.000094
>>>> No. of Iterations:  11
Class  1 metrics:
>>>> Device Residence Times:
     Device  1 :  40.000000
     Device  2 :   0.000160
     Device  3 :   0.051456
     Device  4 :   0.071831
>>>> Class  1 response time :  40.123447
>>>> Class  1 throughput....:   0.249231
Class  2 metrics:
>>>> Device Residence Times:
     Device  1 :  20.000000
     Device  2 :   0.000418
     Device  3 :   0.432464
     Device  4 :   0.602337
>>>> Class  2 response time :  21.035219
>>>> Class  2 throughput....:   0.950786
Class  3 metrics:
>>>> Device Residence Times:
     Device  1 :  15.000000
     Device  2 :   0.001271
     Device  3 :   1.019676
     Device  4 :   1.413942
>>>> Class  3 response time :  17.434890
>>>> Class  3 throughput....:   0.286781
>>>> Press Enter
```

Il formalismo per la descrizione del modello a rete di code accettato da OPENQN è abbastanza simile a quello di CLOSEDQN e anche per OPENQN non è possibile dare un nome ai server o ai workload ed alle classi. Un esempio di file di input per OPENQN è il seguente:

```
3  3       K RR
3.65  0.1  0.05  Vector_L (lambda)
1  35      Device 1 type (LD): network
   1.000
   0.720
   0.673
   0.652
   0.641
   0.633
   0.628
   0.625
   0.622
   0.619
   0.617
   0.616
   0.615
   0.613
   0.613
   0.612
   0.611
   0.610
   0.610
   0.609
   0.609
   0.608
   0.608
   0.608
```

```
     0.607
     0.607
     0.607
     0.606
     0.606
     0.606
     0.606
     0.606
     0.605
     0.605
     0.605
  0  0     Device 2 type (LI):  server cpu
  0  0     Device 3 type (LI):  server disk
>>>> Service Demand Matrix
  0.011224 0.003015  0.0006114
  0.030    0.015     0.005
  0.263    0.102     0.030
```

Il significato delle righe è pressappoco identico a quello di CLOSEDQN tranne per la seconda riga che, invece del vettore popolazione, contiene il vettore dei tempi di arrivo per le classi di job. L'output di OPENQN per il formalismo dell'esempio è il seguente:

```
OpenQN - (c) Copr. 1994 D. Menasce', V. Almeida, and L. Dowdy.
             All Rights Reserved.
This program comes with the book 'Capacity Planning and
Performance Modeling: from mainframes to client-server systems'
by Menasce, Almeida, and Dowdy, published by Prentice Hall.
>>>> Class  1 Throughput:   3.650000
>>>> Class  2 Throughput:   0.100000
>>>> Class  3 Throughput:   0.050000
>>>> Utilization of Device  1 :      4.130 %
>>>> Utilization of Device  2 :     11.125 %
>>>> Utilization of Device  3 :     97.165 %
Class  1 metrics:
>>>> Device Residence Times:
     Device  1 :   0.012110
     Device  2 :   0.033755
     Device  3 :   9.276896
>>>> Class  1 Response Time........:   9.322762
>>>> Class  1 Avg. Number in System:  34.028080
Class  2 metrics:
>>>> Device Residence Times:
     Device  1 :   0.003253
     Device  2 :   0.016878
     Device  3 :   3.597884
>>>> Class  2 Response Time........:   3.618014
>>>> Class  2 Avg. Number in System:   0.361801
Class  3 metrics:
>>>> Device Residence Times:
     Device  1 :   0.000660
     Device  2 :   0.005626
     Device  3 :   1.058201
>>>> Class  3 Response Time........:   1.064487
>>>> Class  3 Avg. Number in System:   0.053224
>>>>  Avg. Queue Length of LD device  1 :   0.044561
>>>> Press Enter
```

6.6.1 Modifica ed estensione del formalismo accettato dai due tool

Come per atri tool di cui si disponeva del sorgente, si è voluto modificare il formalismo in modo da renderlo più facilmente interpretabile da un essere umano ed in modo da poter gestire i nomi degli elementi presenti nella rete di code. Un esempio della nuova sintassi per CLOSEDQN modificato è la seguente:

```
# This is a CLOSEDQN Model
CLOSEDQN "The Model's Name"
         # Class Section
         Classes 3
                  # Class Name Population
                  Class  {Compile} 10
                  Class  {Surf   } 20
                  Class  {Compute}  5
         # Server Section
         Nodes 4
                  # Server Name Type [Saturation ( service_rates... )]
                  Server [Client Workstation] DELAY 0
                  Server [DISK] LI 0
                  WorkUnit [DRUM] LI 0
                  Server [Network] LD 35 ( 1.000 0.716 0.668 0.648 0.636 0.629 0.623 0.620 0.617 0.614 0.613 0.611 0.610
                  0.609 0.608 0.607 0.606 0.605 0.605 0.604 0.604 0.603 0.603 0.603 0.602 0.602 0.602 0.601 0.601 0.601
                  0.601 0.601 0.600 0.600 0.600 )
         # Service Demand Matrix
         ServiceDemands
                  # Server_Name               Class_Name = Demand Class_Name = Demand Class_name = Demand
                  [Network]                   {Compile} = 0.00016 {Surf} = 0.000417   {Compute} = 0.001269
                  [Client Workstation]        {Compile} = 40      {Surf} = 20         {Compute} = 15
                  [DISK]                      {Compile} = 0.030   {Surf} = 0.255      {Compute} = 0.615
                  [DRUM]                      {Compile} = 0.036   {Surf} = 0.306      {Compute} = 0.738
END
```

La sintassi è abbastanza evidente quindi non ci soffermeremo su di essa ulteriormente ma è facile accorgersi che essa risulta molto più interpretabile di quella originale.

Anche l'output del programma è stato leggermente modificato ed ora si presenta come segue:

```
/---------------------------------------------------------------\
|ClosedQN - (c) Copr. 1994 D. Menasce', V. Almeida, and L. Dowdy.  |
|           All Rights Reserved.                                  |
|This program comes with the book 'Capacity Planning and          |
|Performance Modeling: from mainframes to client-server systems'   |
|by Menasce, Almeida, and Dowdy published by Prentice Hall.       |
\---------------------------------------------------------------/
Reading input file...
Model Name: "THE MODEL'S NAME"
  N. of Classes = 3
    Class1: "COMPILE" Population: 10
    Class2: "SURF" Population: 20
    Class3: "COMPUTE" Population: 5
    Total Classes Population: 35
  N. of Servers/Nodes = 4
    Server1: "CLIENT WORKSTATION" Type: DELAY Saturation: 0
    Server2: "DISK" Type: LOAD INDEPENDENT Saturation: 0
    Server3: "DRUM" Type: LOAD INDEPENDENT Saturation: 0
    Server4: "NETWORK" Type: LOAD DEPENDENT Saturation: 35 ( 1.00 0.72 0.67 0.65 0.64 0.63 0.62 0.62 0.62 0.61 0.61
0.61 0.61 0.61 0.61 0.61 0.61 0.60 0.60 0.60 0.60 0.60 0.60 0.60 0.60 0.60 0.60 0.60 0.60 0.60 0.60 0.60 0.60 0.60
0.60 )
  Service Demands:
    "NETWORK" "COMPILE" =  0.0001600 "SURF" =  0.0004170 "COMPUTE" =  0.0012690
    "CLIENT WORKSTATION" "COMPILE" =  40.0000000 "SURF" =  20.0000000 "COMPUTE" =  15.0000000
    "DISK" "COMPILE" =  0.0300000 "SURF" =  0.2550000 "COMPUTE" =  0.6150000
    "DRUM" "COMPILE" =  0.0360000 "SURF" =  0.3060000 "COMPUTE" =  0.7380000
End of Model Description
Computing...
Total No. of Iterations:  11 ( Error <=   0.000094 )
Results...
Class 1: "COMPILE" metrics:
  Device Residence Times:
     Device 1: "CLIENT WORKSTATION"  40.000000
     Device 2: "DISK"   0.051456
     Device 3: "DRUM"   0.071831
     Device 4: "NETWORK"   0.000160
   Class 1: "COMPILE" response time :  40.123447
   Class 1: "COMPILE" throughput....:   0.249231
Class 2: "SURF" metrics:
  Device Residence Times:
     Device 1: "CLIENT WORKSTATION"  20.000000
     Device 2: "DISK"   0.432464
     Device 3: "DRUM"   0.602337
     Device 4: "NETWORK"   0.000418
   Class 2: "SURF" response time :  21.035219
   Class 2: "SURF" throughput....:   0.950786
Class 3: "COMPUTE" metrics:
```

```
  Device Residence Times:
     Device 1: "CLIENT WORKSTATION"  15.000000
     Device 2: "DISK"   1.019676
     Device 3: "DRUM"   1.413942
     Device 4: "NETWORK"   0.001271
   Class 3: "COMPUTE" response time :  17.434890
   Class 3: "COMPUTE" throughput....:   0.286781
End of Network Analysis
```

La versione modificata del formalismo per OPENQN è invece la seguente:

```
# This is an OPENQN Model
OPENQN "Central Server Model"
         # Class Section
         Classes 3
                  # Class Name Arrival_Rate
                  Class "WEB Page Request" 3.65
                  Class "Compute"          0.1
                  Class "Disk Access"      0.05
         # Server Section
         Servers 3
                  # Server Name Type [Saturation '(' service_rates... ')']
                  Server [Network] LD 35 (  1.000 0.720 0.673 0.652 0.641 0.633 0.628 0.625 0.622 0.619
                  0.617 0.616 0.615 0.613 0.613 0.612 0.611 0.610 0.610 0.609 0.609 0.608 0.608 0.608
                  0.607 0.607 0.607 0.606 0.606 0.606 0.606 0.606 0.605 0.605 0.605 )
                  Server [CPU]     LI 0
                  Server [DISK]    LI 0
         # Service Demand Matrix
         ServiceDemands
                  # Server_Name       Class_Name = Demand  Class_Name = Demand  Class_name = Demand
                  [Network] "Web Page Request" = 0.011224 "Compute" = 0.003015  "Disk Access" = 0.0006114
                  [CPU]     "Web Page Request" = 0.030    "Compute" = 0.015     "Disk Access" = 0.005
                  [DISK]    "Web Page Request" = 0.263    "Compute" = 0.102     "Disk Access" = 0.030
END
```

L’output della versione modificata di OPENQN è il seguente:

```
/-----------------------------------------------------------------\
|OpenQN - (c) Copr. 1994 D. Menasce', V. Almeida, and L. Dowdy.    |
|           All Rights Reserved.                                  |
|This program comes with the book 'Capacity Planning and          |
|Performance Modeling: from mainframes to client-server systems'  |
|by Menasce, Almeida, and Dowdy published by Prentice Hall.       |
\-----------------------------------------------------------------/
Model Name: "CENTRAL SERVER MODEL"
  N. of Classes = 3
    Class1: "WEB PAGE REQUEST" Lambda: 3.65000
    Class2: "COMPUTE" Lambda: 0.10000
    Class3: "DISK ACCESS" Lambda: 0.05000
  N. of Servers = 3
    Server1: "NETWORK" Type: LOAD DEPENDENT Saturation: 35 ( 1.00 0.72 0.67 0.65 0.64 0.63 0.63 0.63 0.62 0.62 0.62
0.62 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.61 0.60 0.60 0.60
0.60 )
    Server2: "CPU" Type: LOAD INDEPENDENT Saturation: 0
    Server3: "DISK" Type: LOAD INDEPENDENT Saturation: 0
  Service Demands:
    "NETWORK" "WEB PAGE REQUEST" =  0.0112240 "COMPUTE" =  0.0030150 "DISK ACCESS" =  0.0006114
    "CPU" "WEB PAGE REQUEST" =  0.0300000 "COMPUTE" =  0.0150000 "DISK ACCESS" =  0.0050000
    "DISK" "WEB PAGE REQUEST" =  0.2630000 "COMPUTE" =  0.1020000 "DISK ACCESS" =  0.0300000
End of Model Description
Results...
Throughput:
  Class 1: "WEB PAGE REQUEST" =   3.650000
  Class 2: "COMPUTE" =   0.100000
  Class 3: "DISK ACCESS" =   0.050000
Utilization:
  Device 1: "NETWORK" =      4.130%
  Device 2: "CPU" =     11.125%
  Device 3: "DISK" =     97.165%
Class 1: "WEB PAGE REQUEST" metrics:
  Device Residence Times:
    Device 1: "NETWORK" =   0.012110
    Device 2: "CPU" =   0.033755
    Device 3: "DISK" =   9.276896
  Response Time........:   9.322762
  Avg. Number in System:  34.028080
Class 2: "COMPUTE" metrics:
  Device Residence Times:
    Device 1: "NETWORK" =   0.003253
    Device 2: "CPU" =   0.016878
    Device 3: "DISK" =   3.597884
  Response Time........:   3.618014
  Avg. Number in System:   0.361801


Class 3: "DISK ACCESS" metrics:
  Device Residence Times:
    Device 1: "NETWORK" =   0.000660
    Device 2: "CPU" =   0.005626
```

```
    Device 3: "DISK" =   1.058201
  Response Time........:   1.064487
  Avg. Number in System:   0.053224
Avg. Queue Length of Load Dependent Device:
  Device 1: "NETWORK" =   0.044561
End of Network Analysis
```

I due tool sono invocabili dalla riga di comando mediante la semplice sintassi:

- CLOSEDQN <nomefile.dat>
- OPENQN <nomefile.dat>

6.7 MQNA 1 (Integrato) e MQNA 2 (Integrabile)

MQNA è l'acronimo di Markovian Queueing Networks Analyser, ed è un tool software per modellare ed ottenere la soluzione stazionaria di una larga classe di Reti di Code, creato da Leonardo Brenner, Paulo Fernandes e Alfonso Sales della Facoltà di Informatica della Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre - Brasile .

MQNA dispone di un formalismo per la descrizione e la risoluzione di modelli a reti di code aperte e chiuse in forma prodotto, usando gli algoritmi classici della teoria delle code. Ciò che rende interessante questo tool è la sua capacità di tradurre un modello a reti di code in un modello Markoviano, usando il formalismo SPN (*Stochastic Petri Nets*) o SAN (*Stochastic Automata Networks*). Questo formalismo può successivamente essere importato in tools tipo PEPS (*Performance Evaluation of Parallel Systems*) o SMART (*Stochastic Model checking Analyzer for Reliability and Timing*), che possono appunto risolvere modelli SAN e SPN rispettivamente.

L'obbiettivo che MQNA si pone è quello di fornire, in un singolo ambiente, la possibilità di modellare e risolvere una vasta classe di modelli a reti di code. Le classi di modelli a rete di code trattabili con MQNA comprendono alcuni modelli classici (forma-prodotto) e alcuni modelli a capacità finita. Per queste classi di modelli, chiamate rispettivamente PFQN e FCQN, MQNA è in grado di calcolare la soluzione allo stato stazionario (per le PFQN), o di generare la sua rappresentazione Markoviana sotto forma di formalismo SAN o SPN (per le FCQN).

MQNA esiste in due versioni una, più vecchia e limitata, chiamata MQNA1 compilata per gli ambienti DOS, Windows e Linux. La seconda, più completa, chiamata MQNA2 si trova solo compilata per ambiente Linux. Le versioni DOS e Linux sfruttano un'interfaccia testuale/semigrafica, basata su menu, mentre la versione Windows è una vera e propria applicazione dotata di wizard per la creazione e la risoluzione di modelli a reti di code.

Dato che le versioni "a linea di comando" richiedono comunque un'interazione con l'utente del tipo domanda e risposta, non risultano essere adatte all'integrazione con il nostro webservice; tanto meno lo è la versione windows.

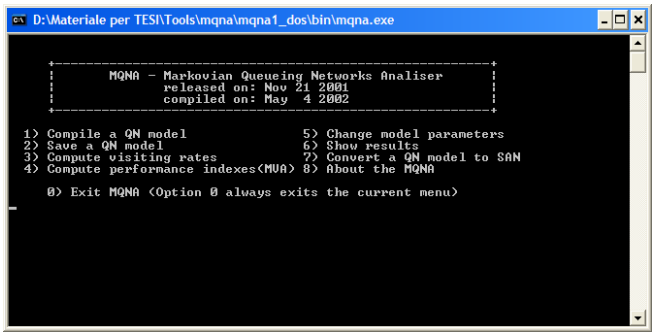


Figura 6.14 - MQNA1 per DOS Menu Principale

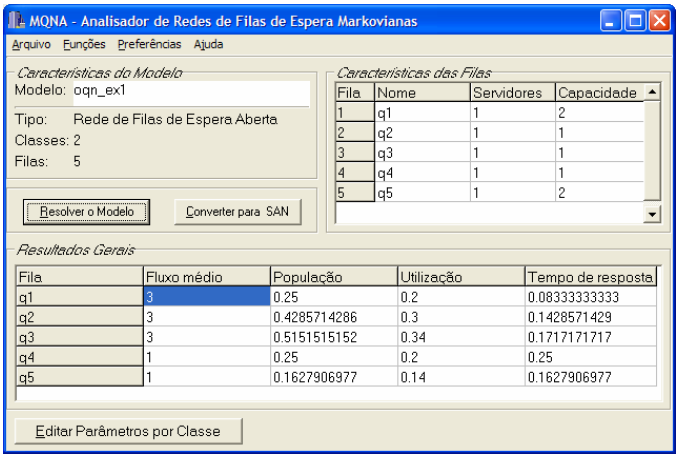


Figura 6.15 - MQNA1 per Windows Finestra Principale

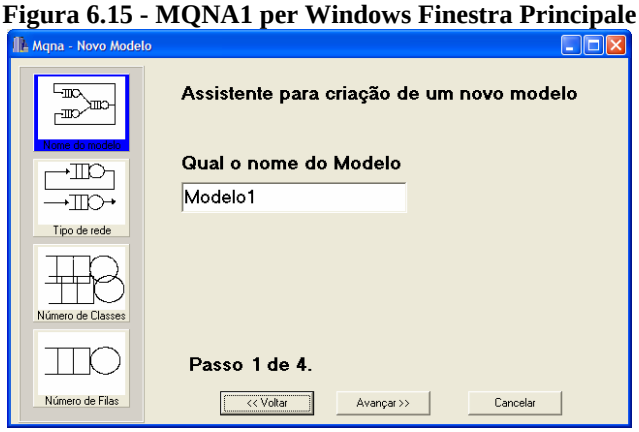


Figura 6.16 - MQNA1 per Windows Wizard per la creazione di un nuovo modello

Contattando uno degli autori, ci sono stati forniti i sorgenti della versione MQNA1, la quale è stata adattata e ricompilata per poter essere invocata da linea di comando (esecuzione in modalità batch) eliminando l'interazione con l'utente. Purtroppo non ci è stata fornita la versione MQNA2 che supportava reti di code in forma prodotto miste multiclasse, mentre MQNA1 supporta solo reti di code in forma prodotto chiuse o aperte a classe singola o multipla.
Nel nostro webservice abbiamo quindi previsto la traduzione di modelli PMIF 2.0 nel formalismo di MQNA1, ed anche la sua risoluzione mediante il tool modificato, mentre MQNA2 è stato integrato solo in parte per quanto riguarda la traduzione da PMIF 2.0 a formalismo MQNA2.
Non ci soffermeremo molto sulla sintassi dei due formalismi, ma ci limiteremo a darne due esempi, questo perché la sintassi risulta essere abbastanza banale ed ovvia.

Esempio di modello MQNA1 - Rete Chiusa con due classi di clienti e tre centri di servizio:

```
cqn rede(2,3)
declarations {
  class classe1;
  class classe2;
  queue q1;
  queue q2;
  queue q3;
}

network {
  population class classe1 = 2;
  population class classe2 = 2;

  queue q1 {
    servers = 1;
    capacity = 5;
    class classe1 {
```

```
      service time = 0.2;
      rot q2: 0.5;
      rot q3: 0.5;
    }
    class classe2 {
      service time = 0.2;
      rot q2: 0.5;
      rot q3: 0.5;
    }

  }
  queue q2 {
    servers = 1;
    capacity = 2;
    class classe1 {
      service time = 0.4;
      rot q1: 1;
    }
    class classe2 {
      service time = 0.4;
      rot q1: 1;
    }
  }
  queue q3 {
    servers = 1;
    capacity = 1;
    class classe1 {
      service time = 1;
      rot q1: 1;
    }
    class classe2 {
      service time = 1;
      rot q1: 1;
    }
  }
}
```

Notare l'uso delle parole riservate:

- "cqn" per indicare che si tratta di un modello chiuso;
- "population class" per specificare la popolazione della classe chiusa;
- "capacity" per dimensionare la capacità della coda.

Nella documentazione di MQNA1 si specifica che quando una coda diventa satura, i clienti smettono semplicemente di arrivare, ovvero rimangono nel centro di servizio precedente fin che non si libera un posto in coda.

Esempio di modello MQNA1 - Rete Aperta con due classi di clienti e tre centri di servizio:

```
oqn redeo3(2,4)
declarations{
 class c1;
 class c2;
 queue q1;
 queue q2;
 queue q3;
 queue q4;
}
network{
 queue q1 {
  servers = 1;
  capacity = 2;
  class c1 {
   service time = 0.1;
   arrival rate = 3;
   rot q2:0.8;
   rot q3:0.2;
  }
```

```
 class c2 {
  service time = 0.1;
  arrival rate = 0;
  rot q2:0.4;
  rot q3:0.6;
 }
}
queue q2 {
 servers = 1;
 capacity = 2;
 class c1{
  service time = 0.1;
  arrival rate = 1;
  rot q4:0.5;
 }
 class c2{
  service time = 0.1;
  arrival rate = 2;
  rot q4:0.3;
 }
}
queue q3 {
 servers = 1;
 capacity = 1;
 class c1 {
  service time = 0.1;
  arrival rate = 0;
  rot q1:0.3;
  rot q4:0.3;
 }
class c2 {
  service time = 0.1;
  arrival rate = 2;
  rot q1:0.1;
 }
}
queue q4 {
 servers = 2;
 capacity = 1;
 class c1 {
  service time = 0.1;
  arrival rate = 0;
 }
 class c2 {
  service time = 1;
  arrival rate = 0;
 }
}
}
```

Anche per le reti aperte c’è da notare l’uso di alcune parole riservate:

- “oqn” per indicare che si tratta di un modello aperto;
- “arrival rate” per specificare il tasso di arrivo dei clienti dall’esterno nella rete;
- “capacity” per dimensionare la capacità della coda.

Esempio di modello MQNA2 - Rete Mista con due classi di clienti e tre centri di servizio:

```
mqn rede(2,3)
declarations {
  class classe1;
  class classe2;
  queue q1;
  queue q2;
  queue q3;
}
network {
  population class classe1 = 2;
  queue q1 {
    priority (1,2);
    servers = 1;
    capacity = 10;
    class classe1 {
      service time = 1.0;
      routing q2(blocking): 0.5;
      routing q3(blocking): 0.5;
    }
    class classe2 {
      service time = 2.0;
      arrival rate = 1.0;
      routing q2(blocking): 0.8;
      routing q3(blocking): 0.2;
    }
  }
  queue q2 {
    priority (1,2);
    servers = 1;
    capacity = 8;
    class classe1 {
      service time = 0.3;
      routing q1(blocking): 1;
    }
    class classe2 {
      service time = 0.4;
      routing q1(blocking): 0.2;
    }
  }
  queue q3 {
    priority (1,2);
    servers = 1;
    capacity = 5;
    class classe1 {
      service time = 1;
      routing q1(blocking): 1;
    }
    class classe2 {
      service time = 2;
      routing q1(loss): 0.5;
    }
  }
}
```

Anche se, purtroppo, non possiamo integrare MQNA2 vediamo alcune caratteristiche e differenze del suo formalismo rispetto a quello di MQNA1:

- “mqn” per indicare che si tratta di un modello misto;
- “arrival rate” usato nelle classi aperte, per specificare il tasso di arrivo dei clienti dall’esterno nella rete;
- “population class” per specificare la popolazione delle classi chiuse;
- “priority” per stabilire una priorità tra i clienti delle diverse classi, siano esse aperte o chiuse;
- “capacity” per dimensionare la capacità della coda;
- “routing” al posto di “rot” usato in MQNA1, per definire le probabilità di transizione;
- “blocking” o “loss” associato alla probabilità di transizione per definire il comportamento del cliente che trova la coda piena. Nel caso di blocking il cliente non transita ma resta bloccato nel centro di servizio precedente fin quando non si libera un posto. Mentre nel caso di loss, il cliente viene perso ed esce dalla rete.

6.8 QNAP2™ (Integrabile):


SIMULOG®

QNAP2 (Queueing Network Analysis Package versione 2) è un il risultato del lavoro d'equipe dell'INRIA (Istituto Nazionale di Ricerca in Informatica e Automazione, in Francia) e di BULL. La versione originale di QNAP2 è stata sviluppata in FORTRAN 77 ed è quindi pienamente portatile. L'interazione con il sistema operativo è garantita dall'interfaccia di Input/Output standard del FORTRAN 77. QNAP2 è ora un tool commerciale prodotto dalla Francese Simulog® (http://www.simulog.fr/), con il nome commerciale MODLINE™ che incorpora QNAP2. Questo è il tool utilizzato nell'implementazione del Web service per la risoluzione di modelli a reti di code in [14].
QNAP2 è un ambiente di modellazione che fornisce utilità specifiche per costruire, maneggiare e risolvere modelli a rete di code. QNAP2 comprende una collezione di algoritmi di risoluzione ed un formalismo per la descrizione del modello, il controllo della sua analisi e per la presentazione dei risultati. Questa interfaccia utente si concretizza in un "meta linguaggio di programmazione" (alcuni autori si riferiscono a questa caratteristica come al "livello algoritmico di QNAP2") che serve per descrivere il modello, e specificare come presentare l'output della sua elaborazione.
QNAP2 tratta oggetti semplici come numeri reali o interi, caratteri, ma anche code di attesa. Con l'aiuto di questi oggetti l'utente può definire una struttura di rete di code di attesa descrivendo un insieme di stazioni.
Il **framework di modellazione** (*pattern*) supportato da QNAP2 è una rappresentazione di un sistema come una rete (*network*) di stazioni (*station*) attraverso le quali circolano i clienti (*customers*). Una stazione viene definita come uno o più server che forniscono servizio a una singola coda di input (*input queue*). La stazione rappresenta la risorsa fisica o logica di elaborazione del sistema modellato, e i clienti rappresentano i processi che vengono eseguiti e che competono per queste risorse.
Una stazione si compone dei seguenti elementi essenziali:

- una coda di attesa;
- uno o più serventi collegati alla coda;
- la descrizione algoritmica del servizio fornito;
- un algoritmo di instradamento che permetta di definire la circolazione degli utenti nella rete.

Gli utenti della rete circolano tra le stazioni in cui ricevono un servizio. É possibile caratterizzare gli utenti attribuendo loro classi distinte.
In QNAP2, esiste un tipo particolare di stazioni, le sorgenti (sources), che permettono di generare degli utenti "ex-novo". Ugualmente esiste una coda particolare, OUT, dove devono dirigersi gli utenti che escono dal modello.
Invece di fornire una grande varietà di **meccanismi di modellazione** predefiniti (*memory allocation policies, polling, dynamic queue selection,…*), i quali sono utili per modellare solo specifici casi di studio, QNAP2 fornisce un insieme limitato di primitive, meccanismi e caratteristiche avanzate del linguaggio di definizione dei modelli, che consentono di costruire, combinandoli, qualsiasi meccanismo si voglia modellare. Attraverso la definizione di procedure e macro-processing facilities, i meccanismi definiti dall'utente risulteranno facili da usare cosi come quelli predefiniti.
Quando la descrizione della rete è terminata, è possibile passare alla seconda fase dell' utilizzo di QNAP: la risoluzione.
Il framework di modellazione di QNAP2 rende possibile il supporto ad un largo insieme di tecniche di analisi, sviluppate per modelli di reti di code standard o generalizzate. In realtà, tenendo a mente la necessità di supportare la continua ricerca nel campo della modellazione tramite reti di code, QNAP2 è stato sviluppato in modo tale da fornire la possibilità di sperimentare nuovi metodi e di implementarli al costo di una limitata scrittura di nuovo codice di programmazione.
QNAP2 mette a disposizione dell'utente le più efficienti e ampiamente validate tecniche di risoluzione attualmente conosciute: ***analytical solvers*** (algoritmo di convoluzione, algoritmo di mean value analysis e metodi approssimati per reti in forma prodotto e loro estensioni), ***discrete event***

simulation con possibilità di *run length control* e un potente risolutore di reti **Markoviano** di specifico interesse per l'analisi esatta di modelli di dimensione finita.
I metodi di risoluzione disponibili sono quindi:

- un simulatore ad eventi discreti;
- metodi analitici esatti;
- algoritmi di convoluzione-teorema BCMP;
- una risoluzione Markoviana;
- metodi analitici approssimati tra cui MVA (Mean Value Analysis);
- metodi iterativi;
- un'approssimazione per diffusione;
- metodi euristici.

I risultati numerici sono degli indici standard come tempo di risposta, tasso di occupazione, lunghezza della coda. Queste quantità sono calcolate, attraverso il loro valore medio quando esiste uno stato di equilibrio con dei metodi analitici, attraverso il loro valore medio e un intervallo di confidenza per i metodi di simulazione.
Relativamente all'insieme di risolutori analitici di QNAP2 (esatti o approssimati) un algoritmo di *dispatching* seleziona il risolutore più consono, in termini di efficienza e accuratezza numerica, in base alle specifiche caratteristiche usate nella definizione del modello.
L'accesso ai tre tipi di risolutori (analytical solver, discrete event simulation, Marcoviano) è reso possibile attraverso il linguaggio di programmazione di QNAP2, al fine di raggiungere la piena indipendenza tra la definizione del modello e la sua analisi. In questo modo le intricate implicazioni tecniche relative ad ogni risolutore sono rese trasparenti all'utente. L'attivazione di un risolutore avviene all'interno del livello algoritmico di QNAP2. Questo consente di combinare i diversi risolutori per modellare sistemi ibridi e gerarchici. Più in generale, usando il livello algoritmico di QNAP2, possono essere costruiti nuovi risolutori sfruttando quelli esistenti (per esempio risolutori euristici basati sulla soluzione iterativa di sottoreti in forma prodotto).
QNAP2 può essere usato potenzialmente in tutti i campi di applicazione che riguardano sistemi di flusso discreti e problemi di allocazione delle risorse come ad esempio: reti di comunicazione, dimensionamento di strutture di elaborazione, linee di produzione ed altri.
QNAP2 è un programma di semplice utilizzo che permette di costruire molto velocemente dei modelli con l'aiuto di una formulazione descrittiva. Il suo pregio principale è di offrire una larga gamma di metodi di risoluzione direttamente accessibili. L'utilizzo pratico mostra tuttavia che la simulazione è utilizzata molto spesso. Questa tecnica è costosa in termini di tempo, e deve quindi essere utilizzata con prudenza.
In un rapporto di ricerca [37] dell'INRIA datato Giugno 1984, si legge (pag. 5 par. 2.5) che: "QNAP2 is released free of charge to non-profit research or education institutions". Tuttavia sul sito della Simulog, che ora produce la nuova versione commerciale MODLINE, non vi è traccia della possibilità di scaricare la vecchia versione QNAP2. Numerose richieste di informazioni relative alla vecchia versione di QNAP2 sono cadute nel vuoto e non c'è stata alcuna possibilità di trovare QNAP2 da altre fonti. Quindi nella nostra implementazione del Web service abbiamo sfruttato ed esteso il lavoro di Smith per quanto riguarda la conversione da PMIF 2.0 a QNAP2 ma non è stato possibile implementare effettivamente la risoluzione del modello tradotto non avendo a disposizione né l'eseguibile né i sorgenti di QNAP2.

6.9 MOSEL / MOSEL 2 (Non integrato)

MOSEL e MOSEL 2: MOdeling Specification and Evaluation Language. Sviluppato da Jörg Barner, Khalid Begain, Björn Beutel, Gunther Bolch (lo stesso autore di PEPSY-QNS) e Helmut Herold [64]. Si tratta di un ambiente integrato per la definizione di modelli di performance e reliability (Queueing Network Models, Petri Net Models, Precedence Graph Models, Fault Trees, Markov Models, Stochastic Activity Networks e Stochastic Process Algebras). Il concetto alla base di MOSEL è che esistono molti tool gratuiti per la risoluzione di questo tipo di modelli, e quindi è inutile sviluppare di nuovo altri tool per la loro analisi, mentre risulta essere più conveniente sviluppare un linguaggio comune per la loro definizione ed utilizzare poi il tool più adatto per la sua analisi. La struttura di MOSEL è la seguente:

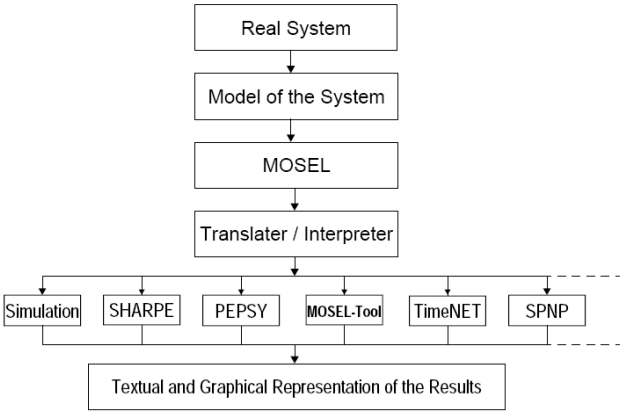


Figura 6.17 - Struttura di MOSEL

Una volta descritto il modello nel linguaggio di MOSEL, esso viene tradotto nel linguaggio nativo del tool più appropriato alla sua risoluzione e poi eseguito ricevendo in output una rappresentazione testuale o grafica dei risultati.

Questo tool non è stato integrato per la complessità del suo linguaggio e per il fatto che, in un certo senso, rappresenta una versione “locale” di ciò che stiamo cercando di fare con il nostro webservice, se anche avessimo integrato MOSEL esso avrebbe comunque richiamato gli altri tool per la risoluzione del modello. Tuttavia questo tool meriterebbe un approfondimento maggiore, e si potrebbe pensare, in futuro, ad una sua integrazione a livello di trasformazione da PMIF 2.0 a MOSEL Language. Inoltre andrebbe indagata la possibilità di utilizzare il tool, che nella documentazione viene chiamato MOSEL-Tool, per la risoluzione di modelli a reti di code.

6.10 Möbius (Non integrabile)

Möbius: Model-Based Environment for Validation of System Reliability, Avalability, Security and Performance. Si tratta di un framework sviluppato presso il Department of Electrical and Computer Engineering and Coordinated Science Laboratory, della University of Illinois Urbana Champaign Illinois USA. Möbius è uno strumento software per modellare il comportamento di sistemi complessi. Il tool deriva dal predecessore UltraSAN, originariamente sviluppato per lo studio dell'affidabilità, availability e performance di computer e sistemi in rete, mediante modellazione con reti SAN (un sottoinsieme delle reti di Petri). L'attuale versione del tool integra più formalismi di modellazione (SAN, Petri Nets, Process Algebras, Fault Tree modelling), che consentono la modellazione di un ampio insieme di sistemi a eventi discreti, dalle reazioni biochimiche agli attacchi di un intruso in un sistema di comunicazione sicuro. Un modello di un sistema complesso è definito in Möbius suddividendo il sistema in submodels, ognuno dei quali può essere descritto utilizzando il formalismo più adatto (SAN, SPN, ecc) .

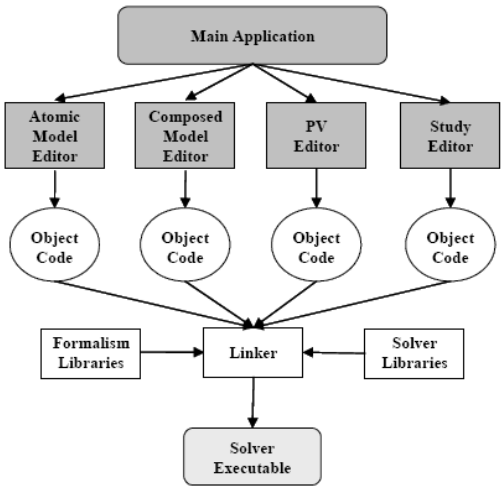


Figura 6.18 - Möbius Architecture

I modelli dei sottosistemi individuati possono essere interconnessi per formare un modello composto rappresentante il sistema complessivo.
Il tool è prettamente grafico e una volta definito il modello, esso viene tradotto in codice C++ che deve essere compilato insieme ad altre librerie del tool stesso. Per questo motivo e per la sua complessità non è stato possibile integrarlo nel webservice.

6.11 Win Modelling Tool V 1.7 (Non Integrabile)

Win Modelling Tool è un package grafico per Microsoft™ Windows© sviluppato dal prof. G. Serazzi, Laboratorio di Valutazione delle Prestazioni del Dipartimento di Elettronica e Informazione del Politecnico di Milano. Il package è costituito da tre tools: WinMVA, WinABA, e WinSIM.

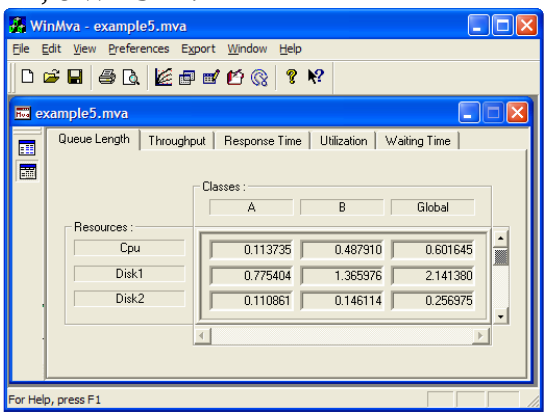


Figura 6.19 - Win Modelling Tool - WinMVA

WinMVA: risolve con l'algoritmo MVA (nella versione stabilizzata) modelli a reti di code aperti, chiusi o misti con carico a classe singola o multiclasse. I server possono essere di puro ritardo (delay) o con coda (queue), con comportamento indipendente (load independent) o dipendente (load dependent) dal carico. Il modello può essere descritto in modo alfanumerico (scegliendo Descrizione alfanumerica del modello) o in modo grafico (scegliendo Descrizione grafica del modello).

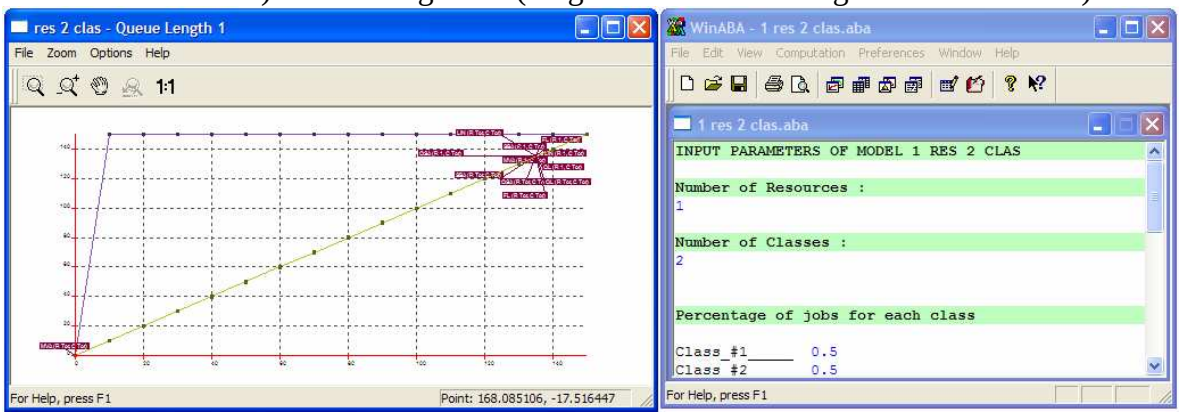


Figura 6.20 - Win Modelling Tool - WinABA

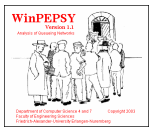
WinABA: calcola i valori limite dei principali indici di prestazione in modelli chiusi multiclasse con vari algoritmi asintotici. Individua i mix delle classi del carico che causano la saturazione di una, due, tre o più risorse. Risolve con metodi approssimati modelli chiusi a reti di code multiclasse.

WinSIM: risolve in simulazione modelli a reti di code, aperti, chiusi o misti, che possono essere descritti sia in modo grafico che in modo alfanumerico. Sono implementate varie distribuzioni per i tempi di servizio e di interarrivo. Il carico può essere a classe singola o multiclasse. Il tempo di risposta di un sistema si ottiene sommando i waiting times (o residence times) delle risorse coinvolte poiché in questa versione non è possibile definire una risorsa di riferimento rispetto alla quale calcolare il tempo di risposta.

È possibile studiare l'andamento degli indici di prestazione al variare della frequenza degli arrivi dei clienti (nelle classi aperte) e del numero dei clienti nel sistema (nelle classi chiuse) entro intervalli di valori definiti dall'utente. I risultati possono essere prodotti in modo tabellare e/o grafico, e salvati come file di testo o Matlab.

Anche questo tool come WinPEPSY non è integrabile in quanto non dispone di un'interfaccia di invocazione a linea di comando.

6.12 WinPEPSY V 1.1 per Windows (Non integrabile)



WinPEPSY per Microsoft™ Windows© Versione 1.1 è un tool grafico sviluppato dal Department of Computer Science, Engineering Sciences Faculty della Friederich-Alexander University Erlagen Nuremberg. Come si può facilmente intuire è l'evoluzione o la trasposizione per l'ambiente Windows di PEPSY-QNS. WinPEPSY è un tool completo per il design grafico e l'analisi di modelli a rete di code. L'interfaccia principale di WinPEPSY consente di disegnare la rete e fissare i parametri di tutti i suoi elementi:

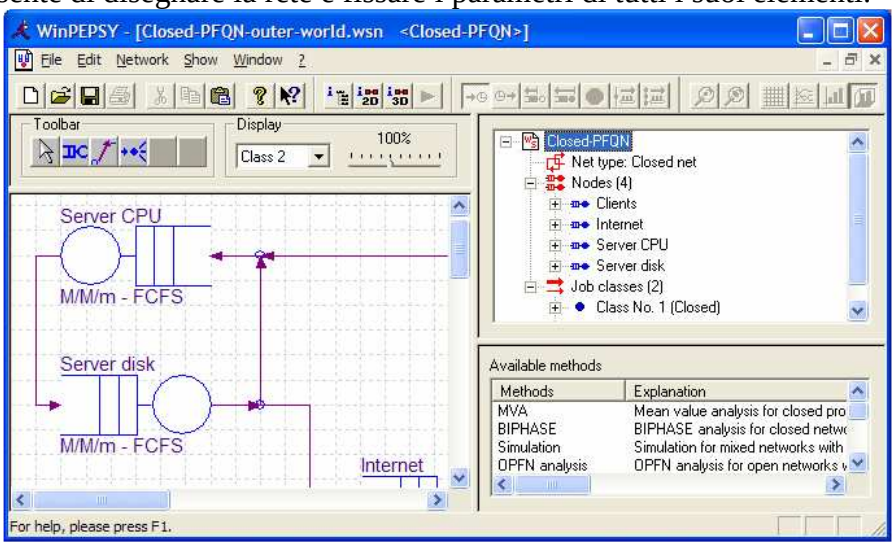


Figura 6.21 - WinPEPSY

Una volta definito il modello, mediante l'editor grafico, si può selezionare il metodo di risoluzione e procedere con l'analisi della rete. WinPEPSY è in grado di creare report grafici per le varie misure effettuate:

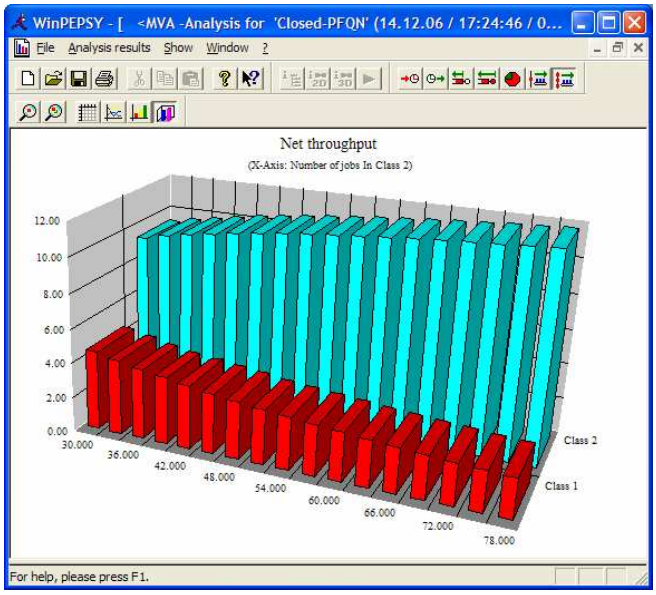


Figura 6.22 - WinPEPSY Net throughput

Tuttavia, WinPEPSY non è integrabile nel nostro web service, in quanto non può essere invocato da linea di comando e inoltre perché utilizza un formato binario proprietario (e non testo) per il salvataggio dei modelli.

6.13 Java Modelling Tools (Non Integrabile)

Questo tool, sviluppato nel Laboratorio di Valutazione delle Prestazioni - Politecnico di Milano. È la versione Java dell'altro tool dello stesso politecnico WinModelling Tool. Anche questo tool purtroppo non è integrabile in quanto non si può invocare da linea di comando. Il package è costituito da sei tools: jSIM, jMODEL, jMVA, jABA, jMCH e jWAT. jSIM. Risolve mediante simulazione modelli a reti di code, aperti, chiusi o misti. Il carico può essere a classe singola o multiclasse. Sono implementate varie strategie di accodamento, di servizio e di routing. Sono disponibili diverse distribuzioni per la generazione stocastica di tempi di servizio e di tempi di interarrivo. Il modello viene descritto in modo testuale, attraverso una interfaccia di tipo wizard. jMODEL risolve un modello a rete di code specificando in modo grafico la topologia della rete. Il modello può essere risolto tramite simulazione oppure, se conforme al teorema BCMP, esportato in modo automatico verso il jMVA per la sua risoluzione mediante algoritmo MVA esatto o approssimato.

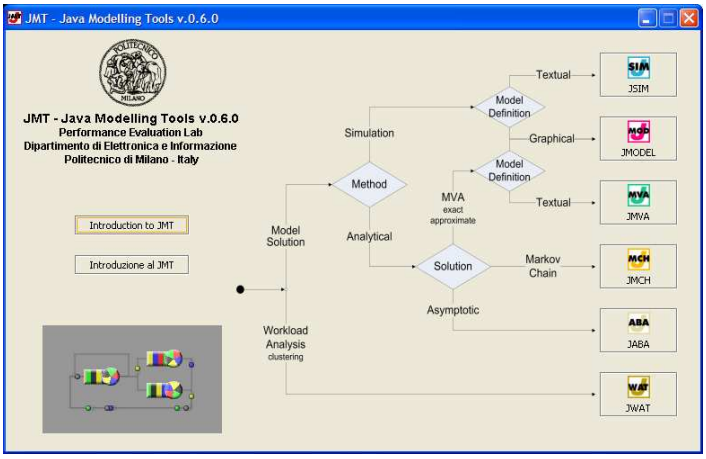


Figura 6.23 - JMT Java Modelling Tools

jMVA: risolve tramite l'algoritmo MVA esatto o approssimato modelli a reti di code aperti, chiusi o misti con carico a classe singola o multiclasse. I servitori possono essere di puro ritardo (delay) o con coda (queue), con comportamento indipendente (load independent) o dipendente (load dependent) dal carico. Il modello viene descritto in modo alfanumerico, attraverso una interfaccia di tipo wizard.

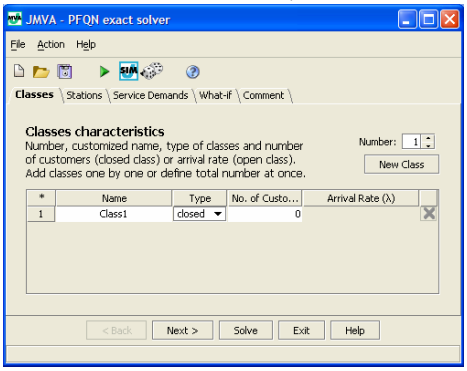


Figura 6.24 - JMT/JMVA PFQN exact solver

jABA: calcola i valori limite dei principali indici di prestazione in modelli chiusi multiclasse con vari algoritmi di analisi asintotica. Permette l'individuazione dei mix delle classi di carico che causano la saturazione di una, due, tre o più risorse. Risolve con metodi approssimati modelli chiusi a reti di code multiclasse. jMCH: risolve in simulazione una stazione con coda, finita (M/M/1/k) o infinita (M/M/1), visualizzandone la catena di Markov corrispondente. E' inoltre possibile variare dinamicamente l'arrival rate e il tempo di servizio del sistema. jWAT: fornisce gli strumenti per l'analisi dei file di log applicando tecniche di clustering.

6.14 Altri tool valutati

Delsi 1.1: Discrete-event simulation system [63]. Si tratta di una libreria per la costruzione di modelli di simulazione in Borland Delphi. Non può essere integrato in quanto un modello è un vero e proprio programma pascal che necessita quindi di essere compilato.

C++SIM 1.5: Object-Oriented Discrete-Event Simulation in C++ [72]. Anche in questo caso siamo in presenza di una libreria di simulazione in C++ che necessita di compilazione.

CSIM: Si tratta di una libreria di routines, da usarsi con i linguaggi di programmazione C e C++, che consente di creare modelli di simulazione ad eventi discreti e process-oriented [56].

GUIDE: A Graphical Interface for Specification of Extended Queueing Network Models [57]. Più che di un tool di risoluzione di modelli a reti di code, si tratta di una interfaccia grafica per il design di modelli a reti di code che poi traduce il progetto in codice eseguibile sfruttando CSIM [56].

DESP-C++: Discrete-Event Simulation Package for C++ [73]. Si tratta di un motore di simulazione ad eventi-discreti per C++. Come le altre librerie necessita di compilazione.

Dependable LQNS: Performability Modelling Tool for Layered Systems [74]. Si tratta di un tool per la modellazione e la valutazione di misure di performance di applicazioni distribuite fault-tolerant, che usano un'architettura separata per la detection delle situazioni di fault. Questo tool supporta l'analisi di Layered Queueing Networks Model il cui studio non rientra negli obiettivi di questa tesi.

MCQueue: Si tratta di un piccolo tool grafico per ambiente Windows, in grado di analizzare modelli Markoviani e a Reti di Code. I tipi di reti di code supportati sono M/G/1, D/G/1, M/M/c, M/D/c, D/M/c, G/M/c, Transient M/M/1, M/M/c/c+N. Purtroppo non può essere invocato da linea di comando e quindi non può essere integrato [75].

MINA: A Tool for MSC-Based Performance Analysis and Simulation of Distribuited Systems [76]. Un tool in parte simulativo ed in parte analitico. Sfrutta un formalismo chiamato MSC (Message Sequence Chart) per la descrizione del modello ed è scritto parte in Microsoft Excel e Parte in Java entrambi difficilmente integrabili.

QSOLVE: An Object-Oriented Software Tool For Teaching Queueing Theory [77]. Tool scritto in Delphi 2.0 per ambiente Windows. Non integrabile in quanto sfrutta l'interfaccia grafica e non ha un formalismo per la descrizione della rete di code ma richiede che essa venga specificata mediante finestre di dialogo.

QNAT: A Graphical Queueing Network Analysis Tool for General Open and Closed Queueing Networks [58]. La versione beta di questo tool viene distribuita gratuitamente su internet ma per la sua esecuzione sfrutta come piattaforma di calcolo Mathematica™ con Mathlink. Integrare questo tool nel webservice vorrebbe dire che ad ogni richiesta di esecuzione di un modello il server che ospita il tool dovrebbe avviare una istanza di Mathematica™ e a patto di avere a disposizione una licenza di questo programma, comunque i tempi di risoluzione sarebbero eccessivi.

QTSPLUS for Excel: Un tool distribuito insieme al libro "Fundamentals of Queueing Theory, Third Edition" [78]. Il tool tuttavia si può scaricare liberamente da internet ma essendo in realtà un foglio di lavoro EXCELL™ 95 o QUATTRO Pro™ risulta impossibile integrarlo nel webservice.

RAQS: Rapid Analysis of Queueing Systems [79]. RAQS è un'applicazione windows che combina diversi modelli di reti di code in un unico ambiente integrato. Essendo un'applicazione totalmente grafica, non c'è modo di integrare RAQS nel nostro progetto.

RESQME: RESearch Queueing package Modelling Environment [59]. Si tratta di un ambiente grafico per la costruzione, soluzione ed analisi di modelli a reti di code. Funzionava su architetture PS/2 con OS/2 e RS/6000 con AIX. Una volta terminata la fase di progetto, il tool traduceva la rete di code in primitive da passare al tool RESQ [60].

RESQ e RESQ2: RESearch Queueing package [60][61]. Si tratta di due tool per l'analisi di sistemi manifatturieri. Analizzando la scarsa documentazione sembra trattarsi di un vero e proprio linguaggio di programmazione che necessità di compilazione. Non è stato possibile integrarlo a causa dell'impossibilità di trovare i sorgenti per windows o linux ed inoltre la sintassi del suo linguaggio sarebbe comunque risultata troppo complicata per attuare una facile conversione da PMIF a RESQ. Tuttavia questo tool meriterebbe un maggiore approfondimento e si potrebbe includere in future versioni del web service.

OMNet++: Si tratta di una libreria di simulazione per sistemi a rete scritta in C++. Dispone anche di un linguaggio di descrizione intermedio chiamato NED Language per la descrizione dei moduli e delle strutture da simulare. Tuttavia alla fine il tool trasforma il progetto in un sorgente C++ da compilare e a cui vanno linkate alcune librerie [80].

SAM: Un Tool per Software Architecture Modelling & Performance Analysis [62]. Si tratta di un insieme di tools che partendo dal workload model e dagli obiettivi di performance aiuta il progettista ad identificare una architettura appropriata per il software intensive system che deve sviluppare. Il tool si basa sulle Layered Queueing Networks (LQN) e i modelli vengono risolti usando tecniche di risoluzione simulative ad eventi discreti (DES). Il tool è sviluppato in parte in Java ed in parte in C++. Il modello "disegnato" con l'interfaccia grafica viene poi tradotto in chiamate alla libreria OMNet++ integrata nel tool.

7 Introduzione ai Web Services

7.1 Architetture locali e distribuite

L'evoluzione delle applicazioni ha portato negli anni a spostare l'attenzione da architetture locali ad architetture distribuite.
In un'applicazione "locale", il programma principale e le sue componenti (altre applicazioni con cui dialoga quella principale, oggetti, librerie, file system, database, ecc…) risiedono tutte sulla stessa macchina fisica. Da un certo punto di vista, in questo tipo di architettura, l'integrazione e il dialogo tra le varie componenti dell'applicazione si può realizzare in maniere veloce, efficiente e senza criticità. Dall'altro tale architettura non ottimizza l'uso delle risorse e richiede la duplicazione di procedure e routine per ogni utente. Inoltre, dal punto di vista dei dati sorge il problema della sincronizzazione dei database tra le postazioni utente, con conseguente proliferazione di dati ridondanti e non aggiornati.
Un architettura distribuita prevede che le diverse componenti di un'applicazione possano risiedere su diverse macchine fisiche messe in comunicazione da un qualche tipo di rete (locale o geografica). Anche se l'applicazione viene complicata dall'overhead necessario alla gestione e sincronizzazione delle comunicazioni tra i client (chi richiede un dato o una sua elaborazione) e i server (chi elabora, aggiorna e mantiene i dati), i vantaggi di questa seconda tipologia di applicazione superano molto spesso gli svantaggi. Primo tra tutti, la possibilità di mettere a disposizione di più utenti una o più applicazioni e database e di poterli usare in modo concorrente, avendo la certezza di lavorare sempre su dati aggiornati in tempo reale.
Le applicazioni distribuite, quindi, offrono dei "servizi" ad uno o più utenti (o genericamente client) in contemporanea. I client accedono all'applicazione mediante un'interfaccia, che può essere un semplice front-end senza logica, oppure una vera e propria applicazione con funzionalità aggiuntive a quella a cui accede per richiedere dei servizi.
Relativamente al "grado" di distribuzione geografica, si possono individuare architetture che assumono nomi particolari, come ad esempio:

- Architettura Client-Server;
- Architettura Tree-Tier / N-Tier;
- Architetture Service Oriented (SOA);

L'architettura distribuita più comune è senza dubbio quella Client-Server. In questo tipo di architettura si individua in genere un fornitore di servizio (Server) e un fruitore del servizio (Client). Il client può essere un semplice visualizzatore dei dati restituiti dal server (come ad esempio un comune browser per pagine HTML), o può essere dotato di un minimo di funzionalità aggiuntive (come nel caso di browser con supporto JavaScript/VBScript).

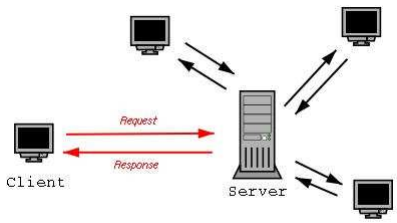


Figura 7.1- Esempio di architettura client-server (CS)

In questo scenario la comunicazione tra client e server è del tipo domanda-risposta. È ovvio che per fornire un adeguato livello di servizio a tutti i client che accedono all'applicazione, il server deve essere dimensionato in modo opportuno per far fronte al carico di lavoro richiesto. Questo comporta un notevole risparmio in termini economici, infatti i client possono essere macchine molto

economiche senza (o con scarse) potenzialità di elaborazione, mentre il server richiederà un investimento maggiore in termini di potenza elaborativa, memoria fisica e spazio disco.
Una evoluzione dell'architettura client-server è l'architettura peer-to-peer in cui ogni macchina funge contemporaneamente sia da client che da server.

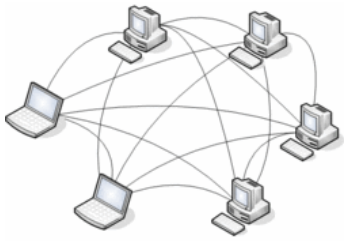


Figura 7.2 - Esempio di architettura peer-to-peer (P2P)

Nelle architetture a più livelli Tree-Tier e N-Tier, il server può delegare ad altre entità parte dell'elaborazione o della gestione dei dati. In origine, l'architettura a più livelli prevedeva che il server fosse scomposto in due parti, una dedicata alla gestione dei dati persistenti e l'altra dedicata alla logica procedurale e alla trasformazione dei dati. Da questo tipo di architettura si sono sviluppate in seguito architetture (dette pattern architetturali) sempre più complesse, con il server suddiviso in un numero di strati logici sempre crescente, ognuno specializzato nella gestione di un particolare aspetto applicativo: gestione dei dati e persistenza, operazioni (o business logic), trasformazione e presentazione dei dati verso client disomogenei (si pensi ad un server web che ha un comportamento e che formatta i dati in modo diverso a seconda del client a cui è destinato il risultato: browser web, cellulari/palmari WAP, TV digitale, ecc…). Ad esempio in ambiente J2EE si implementa il così detto pattern architetturale MVC (model view controller). La parte model, viene gestita da una o più servlet, quella di controller da Java Bean o EJB (Enterprise Java Bean), mentre la parte view viene realizzata tramite JSP (Java Server Pages).
L'evoluzione delle architetture N-Tier è quella delle architetture completamente distribuite, ovvero un'architettura in cui i diversi strati: dati, business e presentazione, non risiedono più sulla stessa macchina fisica o rete locale ma vengono distribuite geograficamente. Questa distribuzione ha creato la necessità di definire delle regole di gioco per i diversi attori in campo. A questo scopo, nel tempo, sono nate architetture complesse da gestire e pesantemente legate al protocollo di comunicazione e alle sottostanti tecnologie utilizzate. Tutte queste tecnologie cercano di fornire delle "specifiche" (per la rappresentazione degli oggetti scambiati), descrivendo il loro ciclo di vita e stabilendo dei "pattern" di comunicazione. Tra le tecnologie più famose e conosciute menzioniamo: CORBA (Common Object Request Broker Architecture), standard multipiattaforma per linguaggi orientati agli oggetti, RMI (Remote Method Invocation), specifico del mondo Java e DCOM (Distribuited COM), tecnologia proprietaria Microsoft.
Nasceva quindi la necessita di stabilire un formato di comunicazione universale che non risentisse della complessità dell'infrastruttura tecnologica sottostante. In parte questo è stato realizzato da RPC (Remote Procedure Call) con uso di XML come formato di rappresentazione delle richieste e delle risposte, senza preoccuparsi del ciclo di vita degli oggetti coinvolti nella transazione.
La rivoluzione si è avuta quando si è pensato di condividere non i componenti ma i servizi che ogni componente era in grado di fornire. Nascono così i primi Web Service.
In questo scenario non ci sono più un unico fornitore ed un fruitore del servizio ma addirittura un fruitore può essere un fornitore di un servizio più complesso.
I servizi esposti eseguono delle singole funzionalità, specifiche ma "atomiche". Si possono pensare come "mattoni" di elaborazione che possono essere combinati per fornire a loro volta "mattoni" più grandi e complessi e, tutti insieme, concorrere all'espletamento di diverse funzionalità.
Avviene quindi uno spostamento del punto di vista, dal ruolo degli attori coinvolti, alla funzione degli attori in gioco. L'architettura dei Web Service non si basa più sull'interazione client/server ma sui servizi che ognuno mette a disposizione degli altri attraverso opportuni "registri" che mantengono le

informazioni sui servizi pubblicati. Questa architettura prende il nome di Service Oriented Architecture (SOA) e la base su cui poggia le fondamenta sono i Web Service e le tecnologie che li implementano.

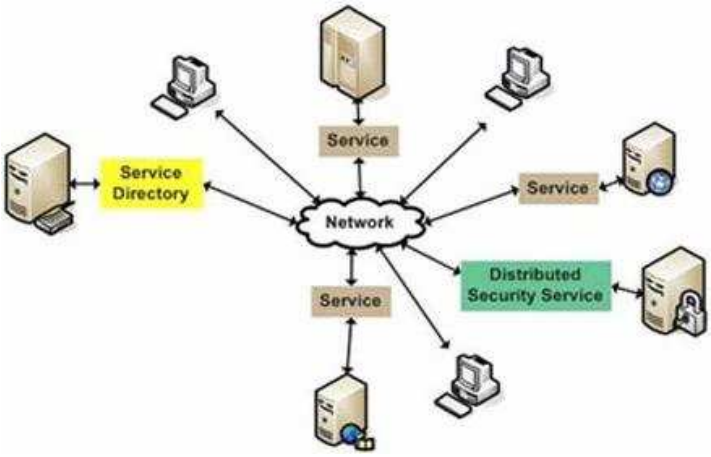


Figura 7.3 - Esempio di Architettura Service Oriented (SOA)

Anche se nel caso più semplice l'architettura SOA si può pensare come un'estensione o rielaborazione dell'architettura client/server, nel caso più generale essa mostra la sua complessità e la necessita di una complessa azione di management.
I Web Service da soli non consentono lo sviluppo di un'architettura SOA ma ne sono i mattoni base. Gli scopi che un'architettura SOA si prefigge sono:

- garantire la riutilizzabilità dei servizi in contesti differenti per scopi differenti;
- svincolarsi dall'uso di una particolare architettura HW/SW, vincolando solo le modalità di comunicazione, e spostando l'attenzione dall'implementazione del servizio verso i dati e il modo di accedervi;
- rendere ogni servizio utile in quanto fornitore di una funzionalità, ma facilmente sostituibile con uno di pari funzionalità eventualmente più economico o efficiente o con caratteristiche aggiuntive;
- garantire l'esecuzione del servizio in modo affidabile.

7.2 Web Services

Un Web Service, in pratica, è un componente software accessibile attraverso i normali protocolli in uso su Internet (HTTP, TCP/IP, SMTP, ecc…). La disponibilità di standard aperti e le crescenti necessita di comunicazione e collaborazione tra persone ed applicazioni hanno creato le condizioni affinché XML e i Web Service diventino la piattaforma per eccellenza per l'integrazione di applicazioni. Le applicazioni verranno costruite usando Web Service multipli, ognuno proveniente da server remoti di aziende diverse, tutti lavoreranno insieme indipendentemente da dove fisicamente sono posizionati nella rete o da come sono stati implementati.
Il modello dei Web Service non si pone in concorrenza con le tradizionali architetture a componenti (CORBA, COM+, EJB) ma le affianca, permettendo di esporre componenti già esistenti verso nuovi client, scritti magari con linguaggi diversi ed operanti su architetture hardware e software diverse.
Il pregio e la forza dei Web Services è quello di utilizzare un set base di protocolli disponibili sulla maggior parte delle piattaforme hardware e software oggi esistenti, permettendo l'interoperabilità tra piattaforme anche molto diverse tra loro mantenendo comunque sempre un alto livello di astrazione e trasparenza.
Esistono varie definizioni di Web Service, e ogni azienda tende a personalizzarne alcuni aspetti. Indipendentemente dalle definizioni, almeno quattro aspetti dei Web Service sono comuni a tutte le implementazioni, i protocolli usati:

- XML: cioè il ben noto eXtensible Markup Language [82] è lo standard usato per rappresentare i dati trasportati ;
- SOAP: acronimo di Simple Object Access Protocol [83], è lo standard usato per definire il formato dei messaggi scambiati. SOAP è in effetti un protocollo che specifica come formattare i dati in formato XML, dati che possono essere messaggi o chiamate di procedure remote. SOAP prevede un formato standard per l'encoding dei dati, che comunque può essere esteso a piacere e definisce come usare i protocolli come HTTP, SMTP e altri per il trasporto dei dati stessi. Ad esempio, utilizzando SOAP con HTTP si mette la richiesta SOAP (un pezzo di codice XML) nel messaggio di richiesta HTTP, e la risposta può essere il solo codice di stato "HTTP 200" che indica che tutto è andato bene o un messaggio SOAP di ritorno più dettagliato contenente i valori di ritorno. Anche in caso di errore si può inviare il solo codice di stato "HTTP 500" (Internal Server Error), o si può inviare un messaggio SOAP con la descrizione dettagliata del codice di errore. SOAP è stato sottoposto nel 2000 al W3C (World Wide Web Consortium) da Microsoft e IBM per essere standardizzato.
- WSDL: ovvero il Web Services Description Language [4] è lo standard usato per descrivere i parametri, i metodi ed i valori di ritorno del web service. Attraverso WSDL è possibile sapere il formato dei messaggi da inviare al Web Service, quali soni i metodi esposti, quali sono i parametri ed i valori di ritorno. È l'equivalente del linguaggio IDL (Interface Description Language) usato da COM e CORBA per definire le interfacce.
- UDDI: Universal Description Discovery and Integration [6] è lo standard promosso dell'omonimo consorzio formato da circa 300 aziende (tra cui: IBM, Microsoft, HP, Intel, Oracle, SUN Microsystems, VeriSign, SAP AG, Ariba, Commerce One, Fujitsu, i2 Tecnology) che ha come scopo quello di favorire lo sviluppo, la scoperta e l'interoperabilità dei servizi Web. UDDI fornisce un database distribuito su cui si possono registrare le aziende ed i servizi da loro esposti. L'interrogazione del database può essere fatta con un comune browser (ad esempio collegandosi ad http://uddi.microsoft.com) o tramite SOAP.

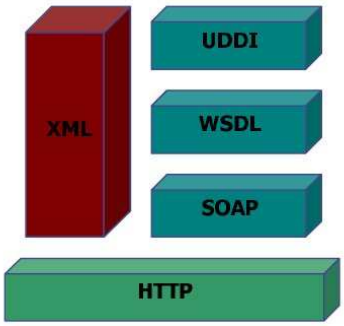


Figura 7.4 - Architettura dei protocolli di un Web Service

Esistono due approcci per lo sviluppo di un Web Service, prendere un componente esistente e trasformarlo in un Web Service oppure scriverne uno in modo nativo partendo da zero.
Per trasformare un componente software già esistente in un Web Service si usano degli appositi Toolkit (generalmente ogni linguaggio e ambiente di sviluppo fornisce il suo), il quale permette di creare il **Wrapper SOAP** per esporlo come Web Service. Alcuni Toolkit generano automaticamente anche il file WSDL e alcuni provvedono anche a registrare automaticamente il componente su un server UDDI (pubblico o provato). Alcuni di questi Toolkit sono: Microsoft SOAP Toolkit 2.0, Microsoft Visual Studio .NET e .NET Framework, Apache SOAP Toolkit e AXIS, IBM Web Services Toolkit, quest'ultimo è in realtà un'estensione del Toolkit di Apache per il supporto dei file WSDL e l'utilizzo di UDDI.
Se bisogna costruire un Web Service da zero, si può scriverlo come se fosse un componente normale e seguire l'approccio precedente, oppure scrivere direttamente un componente con interfaccia SOAP e file WSDL associato.

8 Implementazione del Web Service per Performance Solvers

Come già accennato in precedenza il nostro scopo è quello di sviluppare un Web service in grado di risolvere modelli a reti di code (*QNM*) specificati in formato PMIF 2.0. Qualcosa di simile è già stato realizzato in precedenza da Conie U. Smith e altri nella pubblicazione "A Web Service for Solving Queueing Networks Models using PMIF" [14]. Il lavoro svolto da Smith è stato tuttavia focalizzato sul tool di risoluzione QNAP, noi estenderemo tale lavoro cercando di includere un numero maggiore di tool di risoluzione e nuove funzionalità.
Ciò che si vuole realizzare è un web service, in grado di accettare in ingresso un modello a reti di code in formato PMIF 2.0, e risolverlo fornendo in uscita l'output di uno dei tool integrati nel web service.
Per questo il web service dovrà prevedere un meccanismo per:

1 - specificare un modello (in PMIF 2.0 o nel linguaggio nativo di uno dei tool integrati);
2 - validare sintatticamente il modello (solo per modelli PMIF 2.0);
3 - validare semanticamente il modello (solo per modelli PMIF 2.0);
4 - selezionare uno dei tools integrati;
5 - convertire il modello PMIF 2.0 nel linguaggio del tool selezionato;
6 - eseguire il tool selezionato, passandogli in ingresso il modello convertito;
7 - fornire in uscita l'output dell'esecuzione del tool sul modello.

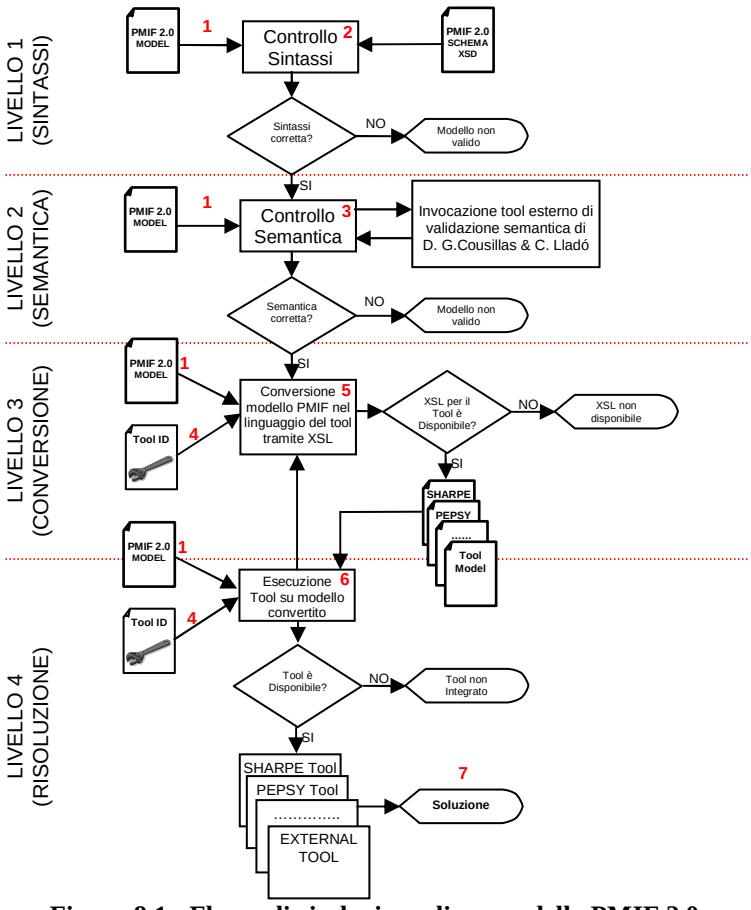


Figura 8.1 - Flusso di risoluzione di un modello PMIF 2.0

Al fine di rendere il web service il più possibile usabile e flessibile, si vuole lasciare all'utente la decisione sulla sequenza delle operazioni da svolgere sul modello PMIF 2.0. Questo vuol dire che l'utente deve poter decidere la sequenza e se eseguire i sette passi (o funzionalità) descritti in precedenza. Questo per consentire una maggiore flessibilità e varietà di impieghi.

Nel flusso di risoluzione del modello (Figura 8.1), l'utente può interagire a differenti livelli di utilizzo con il web service:

Livello 1: Validazione sintattica del modello XML mediante lo schema XSD di PMIF 2.0;
Livello 2: Validazione semantica del modello XML mediante il tool esterno sviluppato da Daniel García Cousillas e C. Lladó [48];
Livello 3: Conversione del modello XML nel linguaggio nativo di uno dei tool integrati;
Livello 4: Risoluzione del modello convertito mediante il suo tool specifico.

Un utente, quindi, potrebbe decidere di usare il web service solo per validare sintatticamente un modello PMIF 2.0, un altro potrebbe voler validare semanticamente il modello e un altro ancora potrebbe aver bisogno solo della funzione di conversione da PMIF ad uno dei linguaggi dei tool integrati. Infine un utente potrebbe usare effettivamente uno dei tool integrati per far risolvere il modello PMIF dopo averlo validato sintatticamente, semanticamente e convertito nel linguaggio proprio del tool.
Rispetto al progetto di C.U. Smith, J. Rosselló e C. M. Lladó [14], si è aggiunto un livello in più (la validazione semantica) e si sono ampliati gli ultimi due livelli, nel senso che ora è possibile scegliere tra più di un tool di risoluzione e tra più formati di conversione. Il primo livello è invece rimasto essenzialmente invariato.

8.1 Scelte progettuali

Per implementare il web service, si è voluto seguire un approccio che consentisse la sua esecuzione sul maggior numero di piattaforme HW e SW oggi esistenti. Inoltre, nella speranza che questo lavoro possa essere utilizzato e ampliato da chiunque volesse contribuire alla sua crescita, si è deciso di utilizzare gli strumenti messi a disposizione dalla comunità Open Source e Freeware.
Come server per il protocollo HTTP si è scelto di utilizzare Apache 2.0, in quanto è il server open-source più utilizzato, efficiente, sicuro ed estensibile oggi esistente. Questa scelta non è vincolante dato che il web service può comunque girare su altre piattaforme server poiché, in realtà, non sfrutta alcuna caratteristica specifica di Apache.
Come linguaggio di programmazione per lo sviluppo del web service si è deciso di utilizzare PHP 5 con l'estensione PEAR::SOAP (*PHP Extension and Application Repository*: un framework e un sistema di distribuzione per componenti PHP riutilizzabili, messi a disposizione sotto forma di package) per il protocollo SOAP.
PHP risulta molto agevole da utilizzare per applicazioni web, si può incorporare all'interno di pagine web ed inoltre è molto diffuso, conosciuto, semplice da imparare e soprattutto gratuito e disponibile per un'ampia gamma di architetture HW e SW. PHP 5 è dotato di una efficace estensione DOM (*Document Object Model*) che sarà usata per la manipolazione dei file XML, e per la trasformazione di file XML mediante fogli di stile XSLT, che come vedremo saranno usati per trasformare i modelli PMIF 2.0 nel linguaggio nativo dei tools.

8.2 Implementazione del Server SOAP

L'implementazione di un web service in PHP è abbastanza semplice. L'estensione PEAR:SOAP mette a disposizione degli oggetti per l'implementazione di un server soap e per la generazione automatica del WSDL e del Discovery Service.
In questo paragrafo descriveremo in dettaglio l'implementazione del server, mentre il codice sorgente completo verrà riportato nel CD allegato.

Prima di tutto si devono includere le librerie dell'estensione PEAR:SOAP mediante le due istruzioni PHP seguenti:

```
require_once('SOAP/Server.php');
require_once('SOAP/Disco.php');
...
```

la prima istruzione include la libreria necessaria all'implementazione del server SOAP, la seconda invece serve per la generazione automatica del file WSDL e DISCO.

Supponiamo di aver creato una classe chiamata QNSolver che implementa tutte le operazioni ed i metodi descritti nel precedente paragrafo, e che vedremo dettagliatamente in seguito.

Inizializziamo una sessione persistente per il colloquio con il client, e sopprimiamo il report degli errori per evitare interferenze con il protocollo SOAP:

```
...
        ...definizione della classe QNSolver...
...
// Inizializzazione della sessione PHP
session_start();
// Verifichiamo se esiste una sessione precedente
// Tentiamo di recuperare la sessione...notare l'uso di "&"
$server =& $_SESSION['server'];
// Se la sessione non esiste già, allora la creiamo
if( !isset( $server ) )
{
        ...creazione del server SOAP ...
}
// Sopprimiamo tutti gli errori per evitare interferenze con la risposta SOAP
error_reporting(0);
...
```

la creazione del server SOAP avviene con le seguenti istruzioni:

```
...
// Creazione di una istanza della classe SOAP_Server
$server = new SOAP_Server();
...
// Creo un'stanza della mia classe QNSolver che implementa il web service vero e proprio
$webservice = new QNSolver();
// L'istruzione seguente crea un collegamento tra il SERVER SOAP e la mia classe
$server->addObjectMap($webservice,'http://schemas.xmlsoap.org/soap/envelope/' );
...
```

A questo punto il server è stato creato ed è pronto a ricevere le richieste del client. Quindi bisogna capire cosa è stato richiesto dal client:

```
// Controlliamo quale richiesta è stata fatta dal client

// E' una richiesta SOAP? ovvero HTTP/POST
if (isset($_SERVER['REQUEST_METHOD']) && $_SERVER['REQUEST_METHOD']=='POST')
{
        // Inviamo la richiesta SOAP al web service
        $server->service($HTTP_RAW_POST_DATA);

} else { // non è una richiesta SOAP? allora deve essere una richiesta HTTP/GET

        // Creiamo un istanza della classe SOAP DISCO Server
        $disco = new SOAP_DISCO_Server($server,'QNSolver');

        // Il client ha richiesto il file WSDL?
        if (isset($_SERVER['QUERY_STRING'])
                && (strcasecmp($_SERVER['QUERY_STRING'],'wsdl') == 0) )
        {
                // Impostiamo il tipo di risposta come file di tipo Testo/XML
                header("Content-type: text/xml");
                // forniamo in output il file WSDL, generato al volo dalla classe $disco
                echo $disco->getWSDL();

        // Il client ha richiesto il file DISCO?
        } elseif (isset($_SERVER['QUERY_STRING'])
                && (strcasecmp($_SERVER['QUERY_STRING'],'disco') == 0) ) {
                // Impostiamo il tipo di risposta come file di tipo Testo/XML
                header("Content-type: text/xml");
                // forniamo in output il file DISCO, generato al volo dalla classe $disco
                echo $disco->getDISCO();

        // Altro tipo di richiesta??
        } else {
                // Impostiamo il tipo di risposta come file di tipo HTML
                header("Content-type: text/html");
                ...stampa un messaggio di descrizione del web service in formato HTML...
        }
}
exit;
```

Il codice del web service PHP con PEAR:SOAP è quindi abbastanza semplice e standard. Quello esposto sopra è uno “scheletro” di web service, che può essere utilizzato come base di partenza per la costruzione di qualsiasi web service. Ciò che cambia, e che implementa il web service vero e proprio, è la classe, che nel nostro caso si chiama QNSolver, che contiene al suo interno tutti i metodi che si vogliono esporre sul web.

8.3 Implementazione della Classe QNSolver

La classe QNSolver rappresenta il cuore del nostro webservice. Questa classe implementa tutti i metodi pubblici invocabili via SOAP ed esportati nell'interfaccia WSDL.
Diamo ora una descrizione generale della classe, mentre il codice sorgente completo verrà riportato nel CD allegato:

```
class QNSolver
{
   // Variabili Pubbliche

      // Necessaria per la generazione automatica del file WSDL
      var $__dispatch_map = array();

   // Metodi Pubblici

      // Costruttore della classe in stile PHP5
      public function __construct()

      // Mostra alcune informazioni relative al webservice
      public function Copyright_()

      // Effettua una validazione sintattica di un modello
      public function ValidateSyntax_($model, $model_type)

      // Effettua una validazione semantica di un modello
      public function ValidateSemantic_($model, $model_type)

      // Converte un modello nel linguaggio di un tool specifico
      public function Transform_($model,$model_type,$tool,$method,$params)

      // Converte un modello nel linguaggio del tool e lo risolve
      public function Solve_($model,$model_type,$tool,$method,$params)

      // Converte un modello in un formato descrittivo più comprensibile
      public function GetModelDescription_($model,$model_type)

      // Restituisce un file XML contenente la lista dei tool integrati
      public function GetToolsList_()

   // Metodi Privati

      // Valida un modello PMIF 2.0 in accordo con il suo schema XML
      private function validate_pmif_from_xml_schema($model)
}
```

Rispetto al webservice progettato da C. U. Smith ed altri in [14], vi sono alcune sostanziali differenze. In un primo momento, si era deciso di implementare gli stessi metodi di [14], ed in particolare il metodo Load, per il caricamento del modello PMIF. Ma, nella fase finale di stesura della classe, il metodo Load è stato eliminato e si è preferito rendere tutti metodi atomici ed indipendenti. Questo per evitare incompatibilità con i client, scritti in linguaggi diversi dal PHP, ed il server, scritto in PHP5.
Il webservice in [14], prevedendo il metodo Load, presupponeva un funzionamento di tipo sincrono tra client e server. Il modello viene inviato al webservice, che lo memorizza in qualche modo, e tutte le successive invocazioni di metodi fanno riferimento al modello inviato al webservice con il metodo Load.

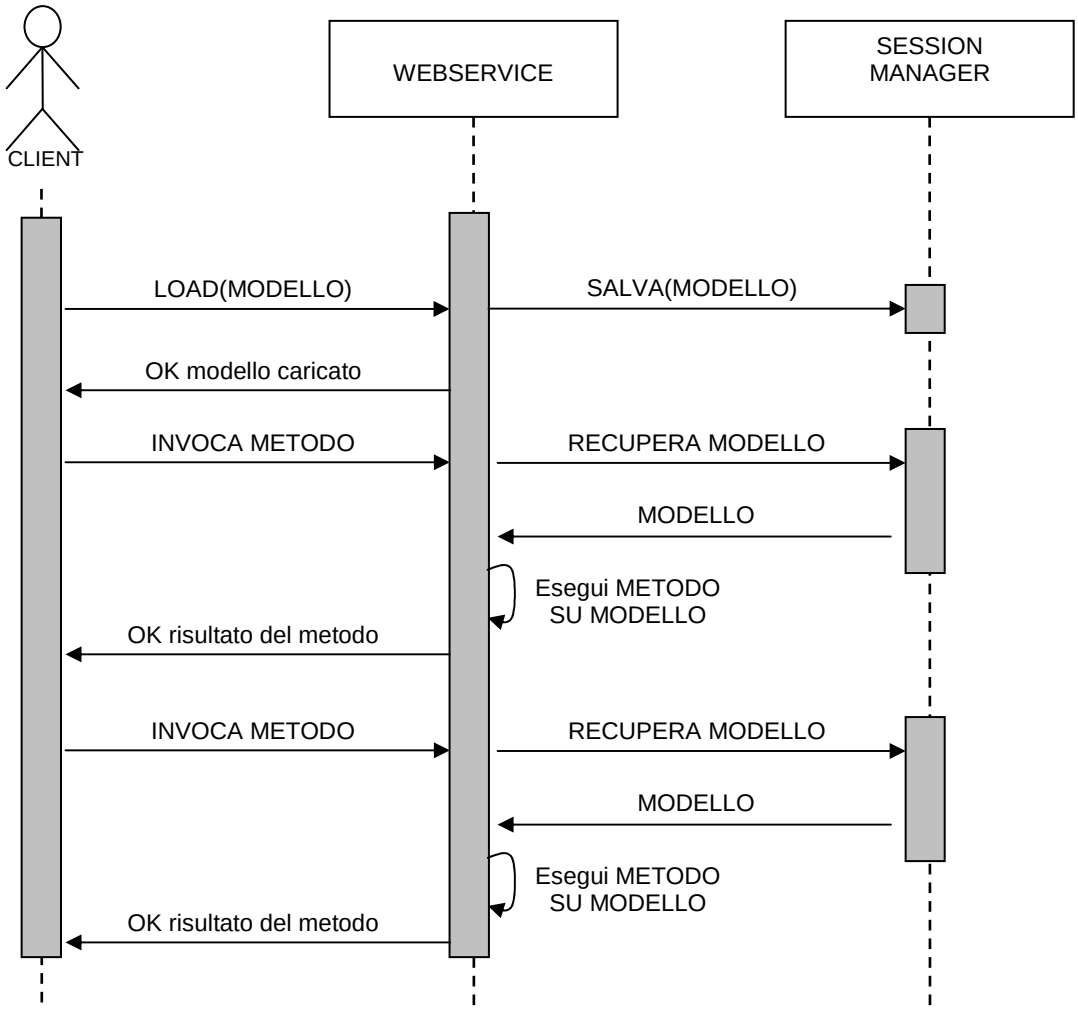


Figura 8.2 - Webservice sincrono con metodo Load

Questa tecnica funziona molto bene con client scritti in PHP, che riconoscono nel header HTTP la sessione PHP. Tuttavia altri linguaggi (come ad esempio ASP e DELPHI) potrebbero avere problemi nel riconoscere la sessione PHP lato server.

Es: Parte di intestazione PHP per il mantenimento della sessione.

```
…
Cookie: PHPSESSID=3dv0p1iblu7j18l5vscetora60
…
```

Questo modello di funzionamento ha il vantaggio di non dover inviare al webservice il modello ad ogni invocazione di un metodo. Ma ha lo svantaggio di un maggior carico di lavoro sul server HTTP e sul webservice stesso che devono mantenere aperto un canale (anche se virtuale) con il client e memorizzare in qualche modo il modello.

Client scritti in altri linguaggi potrebbero non essere in grado di recuperare la sessione e di conseguenza il modello salvato. Per questo, nel nostro webservice, si è deciso di seguire un modello di funzionamento “asincrono”. Tutti i metodi sono atomici ed indipendenti dalle chiamate precedenti effettuate al webservice. Non si deve tenere traccia della sessione, ma si ha lo svantaggio di dover inviare al webservice il modello ad ogni invocazione di un metodo. Questo in caso di modelli pesanti potrebbe diventare oneroso da gestire ma, d’altro canto, si solleva il server ed il webservice dal dover gestire la sessione e memorizzare il modello. Nel caso di un webservice molto utilizzato e con molti

client che vi accedono contemporaneamente, questo potrebbe alla lunga diventare un vantaggio invece di uno svantaggio.

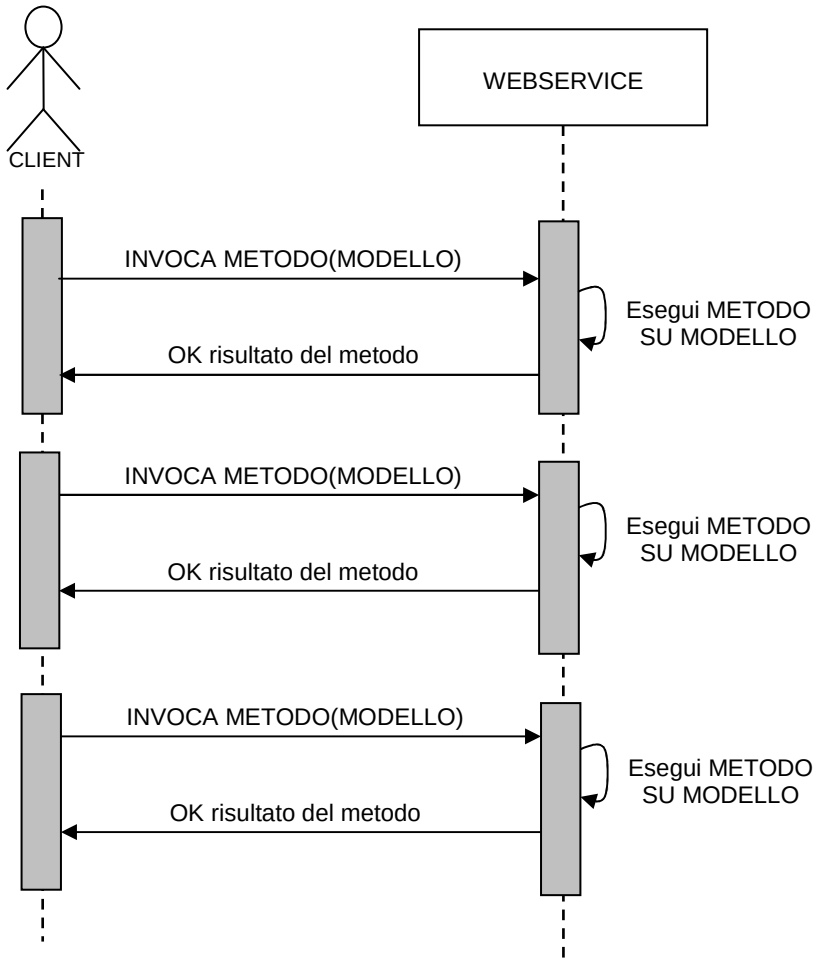


Figura 8.3 - Webservice totalmente asincrono senza metodo Load

Un'altra differenza con il webservice [14] riguarda la presenza del parametro "model_type". Questo parametro è stato previsto per rendere il webservice più generico e flessibile. Infatti senza di esso si sarebbe dovuto vincolare il webservice a risolvere e gestire un solo tipo di modello (ad esempio PMIF 2.0). Tramite il model_type, siamo invece in grado di gestire più tipi di modelli e soprattutto versioni differenti di modelli. Ad esempio se un giorno venisse definito il PMIF 3.0, basterebbe aggiungere al webservice la logica per trattare questo tipo di modello, mantenendo però la gestione del PMIF 2.0.

Nel nostro webservice, tramite il model_type, è stata inserita la possibilità di risolvere modelli scritti nel linguaggio nativo dei tool integrati, senza alcuna conversione, il tutto semplicemente impostando lo stesso valore per i due parametri: "model_type" e "tool".

8.4 Composizione e interazione dei moduli del webservice

Il codice sorgente PHP5 che implementa il webservice si compone di diversi moduli (file), alcuni di programma ed altri di configurazione. In questo paragrafo descriviamo questi moduli e le loro interazioni.
L'intero progetto si compone essenzialmente di due file principali e di alcuni file accessori:

Queueing_Network_Solver_Service.php:
É il modulo principale. Deve essere posizionato all'interno della directory pubblica di Apache, per essere raggiunto via HTTP/SOAP, mediante una richiesta del tipo:

```
http://localhost/Queueing_Network_Solver_Service.php
```

Contiene al suo interno la definizione della classe QNSolver e vengono implementati tutti i suoi metodi, pubblici e privati. Contiene il codice necessario alla creazione del server SOAP, creazione di una istanza di QNSolver, creazione del discovery server per la generazione del file DISCO e WSDL. Contiene inoltre tutto il codice necessario alla gestione della sessione PHP e per il trattamento delle varie richieste SOAP, HTTP/GET, HTTP/POST.
Questo file usa:

- SOAP/Server.php
- SOAP/Disco.php

ed include:

- QNSolver.inc

QNSolver.inc:
É il modulo che contiene le opzioni personalizzabili del webservice. Contiene la definizione dei tool esterni, dei loro file eseguibili e del loro file di conversione XSL.
Deve essere posizionato all'interno della directory pubblica di Apache o comunque in una posizione da cui possa essere richiamato da Queueing_Network_Solver_Service.php.
Al suo interno viene definito uno speciale vettore "**$tools**" dei tool integrati. Questo vettore contiene tutte le informazioni relative ai tools:

- XSL: nome del file XSLT di conversione da PMIF al linguaggio del tool;
- XSL_PARAM: array dei parametri richiesti dal file XSL per la conversione, come ad esempio numero di job o metodo di risoluzione;
- EXE: percorso del file eseguibile ed eventuali parametri da passare sulla riga di comando;
- EXE_PARAM: parametri da passare al file eseguibile in fase di esecuzione, come ad esempio numero di job o metodo di risoluzione.

Contiene il codice sorgente di alcune funzioni di utilità di manipolazione ed estrazione dei dati presenti nel vettore dei tool integrati, nonché delle funzioni per la formattazione dell'output del web service, che possono essere personalizzate dall'utente nel caso ad esempio si voglia un output diverso da XML.

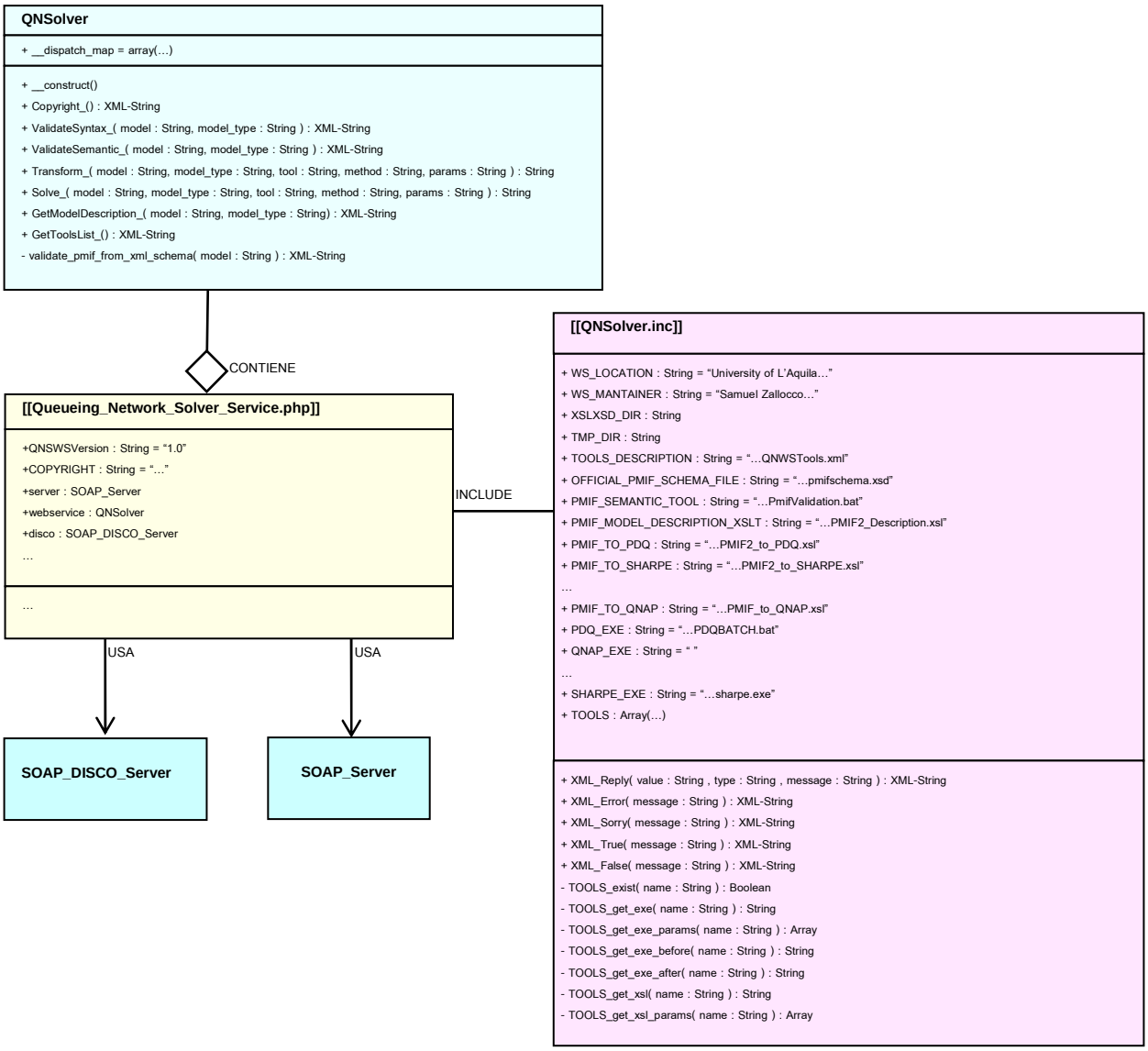


Figura 8.4 - Diagramma UML delle Classi del Webservice PHP5

QNSWSTools.xml:

É il file di descrizione, in formato XML, dei tool integrati nel sistema. Questo file deve essere preparato dal gestore del webservice in base ai tool disponibili sul server che ospita il webservice. Viene inviato, così com'è, al client quando si invoca il metodo GetToolsList_ . Il client lo può usare per capire quali tools sono integrati, quali metodi di risoluzione supportano e di quali parametri necessitano.

pmifschema.xsd:

È la definizione in formato XSD (*XML Schema Definition*) del linguaggio PMIF 2.0. Per una descrizione dettagliata di questo file si veda il paragrafo 5.2.

PMIF2_Description.xsl:

È il file di conversione XSL (*XML Stylesheet Language*) utilizzato per la conversione di un sorgente PMIF 2.0 in un formato descrittivo *human-readable* in lingua inglese. Questo file viene utilizzato quando si invoca il metodo GetModelDescription_ . Per i dettagli si veda il CD allegato.

PMIF2_to TOOL.XSL

Fanno inoltre parte del progetto una serie di file XSL, chiamati:

PMIF2_to_<nometool>.xsl

Il webservice deve contenerne uno per ogni tool integrato. Questi file servono per la conversione da PMIF 2.0 al linguaggio sorgente del tool integrato. Per una descrizione dettagliata di questi file si veda il CD allegato.

Attualmente sono stati sviluppati i seguenti file di conversione per i relativi tool:

Nome Tool	Nome File di Conversione
PDQ	PMIF2_to_PDQ.xsl
QNAP	PMIF2_to_QNAP.xsl
SHARPE	PMIF2_to_SHARPE_PFQN.xsl
OPENQN	PMIF2_to_OPENQN.xsl
CLOSEDQN	PMIF2_to_CLOSEDQN.xsl
MVACCKSW	PMIF2_to_MVACCKSW.xsl
PMVA	PMIF2_to_PMVA.xsl
PEPSY	PMIF2_to_PEPSY.xsl
MQNA1	PMIF2_to_MQNA1.xsl
MQNA2	PMIF2_to_MQNA2.xsl

8.5 Post-Deployment del web service

Il web service è stato sviluppato in ambiente Windows XP, con Apache 2.0 e PHP 5. In seguito si assumerà che il web service sia in funzione sul computer locale (localhost) e che risponda all'indirizzo URL:

```
http://localhost/Queueing_Network_Solver_Service.php
```

Lo scenario classico di invocazione di un web service si può schematizzare genericamente come segue:

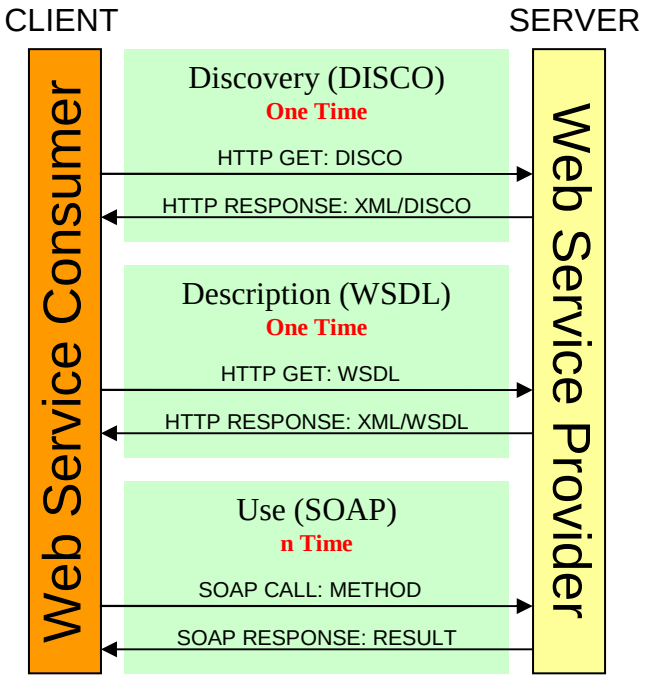


Figura 8.5 - Fasi dell'invocazione di un web service

8.5.1 Fase di Discovery del web service

La prima fase, detta di discovery, è superflua se si conosce già l'URL del descrittore WSDL del servizio. La fase di discovery entra in gioco nel caso si decida di pubblicare il web service in un server UDDI. Questa peculiarità, anche se implementata nel web service, non verrà utilizzata.

Il web service risponde ad una richiesta di discovery formulata, anche con un comunissimo browser (Internet Explorer o Firefox), con la seguente sintassi:

```
http://localhost/Queueing_Network_Solver_Service.php?disco
```

fornendo come risposta il seguente file XML:

```
<?xml version="1.0" ?>
<disco:discovery xmlns:disco="http://schemas.xmlsoap.org/disco/"
        xmlns:scl="http://schemas.xmlsoap.org/disco/scl/">
        <scl:contractRef ref="http://localhost/Queueing_Network_Solver_Service.php?wsdl" />
</disco:discovery>
```

Come si vede, il file XML di discovery contiene un riferimento al descrittore WSDL del web service con l'URL esatto da invocare.

L’estensione PEAR:SOAP di PHP 5 si occupa di generare questo file, in risposta ad una richiesta di discovery, in modo automatico.

8.5.2 Fase di Description del web service tramite file WSDL

Nella seconda fase, detta di description, il client chiama il web service tramite protocollo HTTP e chiede il file WSDL. Il file WSDL contiene la descrizione del web service con i metodi pubblici e la sintassi di invocazione degli stessi. Il file WSDL del nostro web service, invocabile tramite una richiesta HTTP con un qualsiasi browser al seguente URL:

```
http://localhost/Queueing_Network_Solver_Service.php?wsdl
```

è riportato nell’Appendice B:.

Il file contiene una descrizione dettagliata del servizio fornito dal web service, dei messaggi, delle funzioni e dei parametri per una sua corretta invocazione.
La descrizione dettagliata del significato degli elementi presenti in un file WSDL esula dallo scopo di questa tesi e per chi volesse approfondire lo studio di WSDL e UDDI segnaliamo i riferimenti bibliografici [4] e [6].

In molti ambienti di sviluppo i file WSDL e DISCO vengono generati durante la fase di sviluppo del web service e entrano a far parte del progetto come elementi statici. In altri ancora, i file WSDL e DISCO devono essere scritti a mano dal progettista. In PHP, l’estensione PEAR::SOAP, come nel caso del DISCO, fornisce un meccanismo per descrivere i metodi della classe che devono essere esportati e resi visibili nel file WSDL. In fase di creazione dell’istanza della classe che implementa il web service, PEAR::SOAP si occupa di creare “al volo” il file WSDL. Questo offre un non indifferente vantaggio rispetto ad altre soluzioni, primo tra tutti quello di non dover riscrivere manualmente il descrittore WSDL ogni volta che si effettua una minima modifica della classe che implementa il web service.

8.5.3 Uso del web service, descrizione dei metodi pubblici

A questo punto, il client, interpretando il file WSDL, è in grado di sapere quali funzionalità sono messe a disposizione dal web service e quali parametri inviare per effettuare una richiesta.
Queste funzionalità corrispondono a dei metodi o funzioni che riportiamo di seguito con una dettagliata descrizione dei parametri di input e dell'output fornito.

Metodo: Copyright_
Input : nessuno
Output: copyright_tips:String
Descrizione: Visualizza alcune informazioni relative al web service, ed alla sua installazione. Come ad esempio la versione del web service, il sistema operativo su cui sta girando, la versione di PHP utilizzata e il mantainer del web service. Es:

```
Queueing Network Solver Webservice (QNS-WS)
WS Version : 1.0
Guest OS   : WINNT
PHP Version: 5.1.1
Location   : University of L'Aquila - Science Faculty
Contact    : Samuel Zallocco <samuel.zallocco@univaq.it>
Created by Samuel Zallocco.
(CC) Some right reserved.
```

Metodo: GetToolsList_
Input : nessuno
Output: available_tools_list:String
Descrizione: Restituisce la lista, in formato XML, dei tools integrati con: i parametri necessari alla loro esecuzione, le operazioni supportate ed i metodi di risoluzione disponibili. Questo file può essere usato, dal client, per fornire all'utilizzatore del web service un menu di scelta con i tools ed i metodi disponibili per la risoluzione del modello ed, eventualmente, richiedere i parametri necessari per l'esecuzione del tool, come ad esempio il numero di Job o l'algoritmo di risoluzione.
Il file XML ha il seguente formato generale:

```
<?xml version="1.0"?>
<TOOLSLIST>
  <TOOL NAME="ID/nome univoco del tool" VERSION="versione"
        DESCRIPTION="descrizione" AUTHOR="autore">
    <OPERAZIONE>
      <METHOD NAME="NOMEMETODO1" DESCRIPTION="Descrizione Metodo 1" >
        <PARAMS NAME="Nome parametro" TYPE="tipo del parametro"/>
        ...
      </METHOD>
      ...
      <METHOD NAME="NOMEMETODO1" DESCRIPTION="Descrizione Metodo 1" >
        <PARAMS NAME="Nome parametro" TYPE="tipo del parametro"/>
        ...
      </METHOD>
    </OPERAZIONE>
    ...
    <OPERAZIONE>
      <METHOD NAME="NOMEMETODO1" DESCRIPTION="Descrizione Metodo n" />
      ...
      <METHOD NAME="NOMEMETODOn" DESCRIPTION="Descrizione Metodo n" />
    </OPERAZIONE>
    ...
  </TOOL>
  ...
</TOOLSLIST>
```

Nel caso il file di definizione dei tools esterni non sia stato configurato viene visualizzato un messaggio d'errore.

Metodo: ValidateSyntax_
Input : model:String, model_type:String
Output: validated:String
Descrizione: Riceve in input un modello ed il formato del modello e ne effettua una validazione sintattica. Il parametro model, di tipo stringa, deve contenere un modello a reti di code. Per ora i modelli supportati sono solo quelli in formato PMIF 2.0, ma in futuro si potrebbe espandere questa funzionalità ad altri tipi di modelli. Quindi per ora, il parametro model_type può assumere solo il valore "PMIF 2.0".
Restituisce in output il risultato della validazione in formato XML. La risposta può avere uno dei seguenti formati:

- Sintassi errata:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      It seems to be Not Valid,
      according to the official
      PMIF 2.0 Schema.
   </Message>
</QNSolverReply>
```

- Sintassi corretta:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="TRUE" type="Boolean"/>
   <Message>
      It seems to be Valid,
      according to the official
      PMIF 2.0 Schema.
   </Message>
</QNSolverReply>
```

- Modello vuoto:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      ERROR! You have submitted
      an empty model or an empty model type.
   </Message>
</QNSolverReply>
```

- Tipo modello non supportato. Es. SHARPE:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      SORRY! SHARPE models validation not available.
   </Message>
</QNSolverReply>
```

Metodo: **ValidateSemantic_**
Input : **model:String, model_type:String**
Output: **validated:String**
Descrizione: Riceve in input un modello ed il formato del modello e ne effettua una validazione semantica. Per ora i modelli validati sono solo quelli in formato PMIF 2.0, ma in futuro si potrebbe espandere questa funzionalità ad altri tipi di modelli.
Restituisce in output il risultato della validazione semantica in formato XML. La risposta può avere uno dei seguenti formati:

- Tool di validazione semantica non integrato:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      SORRY! Semantic validation tool not available.
   </Message>
</QNSolverReply>
```

- Modello vuoto:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      ERROR! You have submitted
      an empty model or an empty model type.
   </Message>
</QNSolverReply>
```

- Tipo modello non supportato. Es. SHARPE:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      SORRY! SHARPE model semantic validation not available.
   </Message>
</QNSolverReply>
```

- Validazione semantica andata a buon fine. Es.:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="TRUE" type="Boolean"/>
   <Message>
      RESULTS: The PMIF Model is semantically correct.
      Details:
      ...
      1.<?xml version="1.0"?>
      2.<QueueingNetworkModel ... >
            ..viene riportato l'intero modello..
      19.</QueueingNetworkModel>
      RESULTS: The PMIF Model is semantically correct.
   </Message>
</QNSolverReply>
```

- Validazione semantica non andata a buon fine. Es.:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
       The pmif file contains repetitions or invalid Server/WorkUnitServer with quantity 0
       and associated to a ServiceRequest

       RESULTS: 2 error(s).

       Details:
       1.<?xml version="1.0"?>
       2.<QueueingNetworkModel Name="ClosedQueueExample" ... >
       3.   <Node>
       4.     <Server Name="CPU" Quantity="1" SchedulingPolicy="PS"/>
       ^^ #1. Line 4.
              ERROR-07a: The ThinkDevice CPU (4) associated with the ClosedWorkload(s) ACCESS (13)
              must have schedulingPolicy IS (infinite servers)
       5.     <WorkUnitServer Name="DISK1" Quantity="1" SchedulingPolicy="FCFS"
               ServiceTime="0.12" TimeUnits="Sec"/>
       6.     <WorkUnitServer Name="DISK2" Quantity="1" SchedulingPolicy="FCFS"
               ServiceTime="0.015" TimeUnits="Sec"/>
       7.   </Node>
       8.   <Arc FromNode="CPU" ToNode="DISK1" Description="Arc From CPU To DISK1"/>
       9.   <Arc FromNode="DISK1" ToNode="CPU" Description="Arc From DISK1 To CPU"/>
       10.  <Arc FromNode="CPU" ToNode="DISK2" Description="Arc From CPU To DISK2"/>
       11.  <Arc FromNode="DISK2" ToNode="CPU" Description="Arc From DISK2 To CPU"/>
       12.  <Workload>
       13.    <ClosedWorkload WorkloadName="ACCESS" NumberOfJobs="12" ThinkTime="1.56"
               TimeUnits="Sec" ThinkDevice="CPU">
       14.      <Transit To="DISK2" Probability="0.75"/>
       15.      <Transit To="DISK1" Probability="0.25"/>
       16.    </ClosedWorkload>
       17.  </Workload>
       18.  <ServiceRequest>
       19.    <WorkUnitServiceRequest WorkloadName="ACCESS" ServerID="DISK1"
               NumberOfVisits="34">
       20.      <Transit To="CPU" Probability="1"/>
       21.    </WorkUnitServiceRequest>
       22.    <WorkUnitServiceRequest WorkloadName="ACCESS" ServerID="DISK2"
               NumberOfVisits="39">
       23.      <Transit To="CPU" Probability="1"/>
       24.    </WorkUnitServiceRequest>
       25.    <DemandServiceRequest WorkloadName="ACCESS" ServerID="CPU"
               NumberOfVisits="34" ServiceDemand="2.34" TimeUnits="Sec">
       ^^ #2. Line 25.
              ERROR-07b: The ThinkDevice CPU (4) of the ClosedWorkload(s) ACCESS (13) cannot be
              associate to a DemandServiceRequest (25)
       26.      <Transit To="DISK1" Probability="0.25"/>
       27.      <Transit To="DISK2" Probability="0.75"/>
       28.    </DemandServiceRequest>
       29.  </ServiceRequest>
       30.</QueueingNetworkModel>

       RESULTS: 2 error(s).

   </Message>
</QNSolverReply>
```

L’output contenuto nel tag <Message> è quello del tool di validazione semantica esterno. Il tool fornisce una dettagliata descrizione degli errori e della loro posizione all’interno del file PMIF. Per una descrizione approfondita del tool di validazione semantica e sul significato degli errori riportati, si veda [48].

Metodo: **Transform_**
Input : **model:String, model_type:String,**
tool:String, method:String, params:String
Output: **translated_model:String**
Descrizione: Riceve in input un modello ed il formato del modello di origine, il nome del tool di destinazione, un metodo di risoluzione ed eventuali parametri aggiuntivi. Il metodo e i parametri sono opzionali e dipendono dal tool di destinazione. Alcuni tool hanno sintassi differenti in base al metodo di risoluzione richiesto, ed altri hanno bisogno di specificare il numero di job o clienti con cui il modello deve essere risolto. Questo discorso verrà comunque approfondito in seguito. Le possibili risposte del web service sono:

- Modello, tipo o tool vuoto:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      ERROR! You have submitted an empty model/type or empty tool.
   </Message>
</QNSolverReply>
```

- Tool non integrato o non disponibile. Es. QNAP:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      SORRY! The PMIF 2.0 to QNAP conversion was not yet available.
   </Message>
</QNSolverReply>
```

- Codice sorgente del tool richiesto. Ad esempio, nel caso si richieda la conversione nel linguaggio del tool PDQ, passando il seguente modello PMIF 2.0:

```
<?xml version="1.0"?>
<QueueingNetworkModel Name="Rete a Due Fasi”>
  <Node>
    <WorkUnitServer Name="FASE2" Quantity="1" SchedulingPolicy="FCFS" ServiceTime="15"/>
    <Server Name="FASE1" Quantity="1" SchedulingPolicy="IS"/>
  </Node>
  <Arc FromNode="FASE1" ToNode="FASE2" Description="Arc From FASE1 To FASE2"/>
  <Arc FromNode="FASE2" ToNode="FASE1" Description="Arc From FASE2 To FASE1"/>
  <Workload>
    <ClosedWorkload WorkloadName="CARICO" NumberOfJobs="12" ThinkTime="1.23"
                                          TimeUnits="Sec" ThinkDevice="FASE1">
      <Transit To="FASE2" Probability="1"/>
    </ClosedWorkload>
  </Workload>
  <ServiceRequest>
    <WorkUnitServiceRequest WorkloadName="CARICO" ServerID="FASE2" NumberOfVisits="1">
      <Transit To="FASE1" Probability="1"/>
    </WorkUnitServiceRequest>
  </ServiceRequest>
</QueueingNetworkModel>
```

Si ottiene in output il seguente codice PDQ:

```
INIT "Rete_a_Due_Fasi"
CREATENODE "FASE1" CEN IS
VAR $FASE2_ServiceTime = 15
CREATENODE "FASE2" CEN FCFS
CREATECLOSED "CARICO" TERM 12 1.23
SETVISITS "FASE2" "CARICO" 1 $FASE2_ServiceTime
SOLVE EXACT
REPORT
EXIT
```

Metodo: Solve_
Input : model:String, model_type:String,
tool:String, method:String, params:String
Output: solution:String
Descrizione: Riceve in input un modello ed il formato del modello di origine, il nome del tool di destinazione, un metodo di risoluzione ed eventuali parametri aggiuntivi. Il metodo e i parametri, come nel metodo Transform_, sono opzionali e dipendono dal tool di destinazione. Le possibili risposte di questo metodo sono le seguenti:

- Tool non integrato o non disponibile. Es. QNAP:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      SORRY! The QNAP tool is not integrated on this webservice.
   </Message>
</QNSolverReply>
```

- Modello o tipo vuoto:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      ERROR! You have submitted an empty model or an empty model type.
   </Message>
</QNSolverReply>
```

- Eseguibile del tool non integrato o non disponibile. Es. QNAP:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      SORRY! QNAP executable not yet integrated on this webservice.
   </Message>
</QNSolverReply>
```

Questo messaggio può apparire nel caso in cui il web service disponga della possibilità di convertire un modello PMIF 2.0 nel linguaggio del tool selezionato, ma non abbia a disposizione l'eseguibile per risolvere il modello.

- Si è cercato di risolvere un modello con il tool errato. Es. tool QNAP e modello PDQ:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      SORRY! The tool QNAP can not handle a model of type: PDQ.
   </Message>
</QNSolverReply>
```

- Se l’esecuzione del modello è andata a buon fine allora viene visualizzato il risultato dell’esecuzione del tool. Ad esempio per un modello PDQ l’output sarà qualcosa del tipo:

```
                  ****************************************
                  ****** Pretty Damn Quick REPORT *******
                  ****************************************
                  ***  of : Sat Nov 04 17:53:14 2006  ***
                  ***  for: Rete_a_Due_Fasi           ***
                  ***  Ver: PDQ Analyzer v3.0 111904  ***
                  ****************************************
                  ****************************************

                  ****************************************
                  ******    PDQ Model INPUTS      *******
                  ****************************************

Node Sched Resource   Workload   Class     Visits    Service     Demand
---- ----- --------   --------   -----     ------    -------     ------
CEN  ISRV FASE1      CARICO     TERML     0.0000     0.0000     0.0000
CEN  FCFS FASE2      CARICO     TERML     1.0000    15.0000    15.0000

Queueing Circuit Totals:

       Clients:    12.00     for 1st TERM job class
       -----------------
       Generators: 12.00     Total System Clients

       Streams:      1
       Nodes:        2

WORKLOAD Parameters

Client       Number       Demand   Thinktime
----         ------       ------   ---------
CARICO        12.00       15.0000     1.23

                ****************************************
                ******   PDQ Model OUTPUTS      *******
                ****************************************

Solution Method: EXACT

                ******   SYSTEM Performance     *******

Metric                     Value     Unit
-----------------          -----     ----
Workload: "CARICO"
Mean Throughput           0.0667     Job/Sec
Response Time           178.7700     Sec
Mean Concurrency         11.9180     Job
Stretch Factor           11.9180

Bounds Analysis:
Max Throughput            0.0667     Job/Sec
Min Response             15.0000     Sec
Max Demand               15.0000     Sec
Tot Demand               15.0000     Sec
Think time                1.2300     Sec
Optimal Clients           1.0820     Clients

               ******   RESOURCE Performance   *******

Metric          Resource     Work              Value   Unit
---------       ------       ----              -----   ----
Throughput      FASE2        CARICO           0.0667   Visits/Sec
Utilization     FASE2        CARICO         100.0000   Percent
Queue Length    FASE2        CARICO          11.9180   Job
Residence Time  FASE2        CARICO         178.7700   Sec
Waiting Time    FASE2        CARICO         163.7700   Sec
```

Il formato di output per la soluzione del modello è quella tipica del tool utilizzato, ed è quindi differente per ogni tool integrato. Per una descrizione dettagliata del significato dei valori riportati si deve quindi far riferimento alla documentazione dello specifico tool.

Metodo: **GetModelDescription_**
Input : **model:String, model_type:String**
Output: **solution:String**
Descrizione: Riceve in input un modello ed il formato del modello e restituisce una descrizione "testuale" di un modello. Per ora l'unico formato di input supportato è PMIF 2.0. I possibili output sono solo due:

- Modello, tipo vuoto o sconosciuto:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      SORRY! Model/Type empty or unknown Type.
   </Message>
</QNSolverReply>
```

- Descrizione del modello:

```
 ----------------------------
| PMIF 2.0 MODEL DESCRIPTION |
 ----------------------------
GENERAL INFORMATION:
--------------------
  - Model Name.......: Rete a Due Fasi
  - Model Description: Esempio di sistema a due stadi
  - Model Date/Time..: 2006-11-04T18:11:14
  - Model Type.......: Single-Chain Closed Queueing Network Model
NODE DESCRIPTION:
-----------------
The model contain 2 nodes.
- "FASE1" is a Server node, with 1 processing units and scheduling policy "IS";
  It seems to be an M/M/1-IS queue in the Kendall's notation.
- "FASE2" is a Work Unit Server node, with 1 processing units, scheduling policy "FCFS" and
service time 15Sec;
It seems to be an M/M/1-FCFS queue in the Kendall's notation.

Note that a PMIF 2.0 model does not allow the specification of the customers interarrival
times distribution and service times distribution, so the Kendall's classification can be
wrong.

The nodes are connected as follows:
  FASE1 --> FASE2
  FASE2 --> FASE1

WORKLOAD DESCRIPTION:
---------------------
The model contain 1 workloads:
- "CARICO" is a Closed Workload with 12 Jobs and a think time of 1.23 Sec for "FASE1" as
think device;
This workload transit to node "FASE2" with probability 1 (100%)

SERVICE REQUESTS DESCRIPTION:
-----------------------------
The model contain the following Work Unit Service Request:
  - From the "CARICO" Workload to the "FASE2" node:
     Number of Visits = 1
     Service Time     = 15Sec
     Service Demand   = 15Sec
     This service request transit to node "FASE1" with probability 1 (100%)
```

8.5.4 Uso del web service, descrizione dei metodi privati

Metodo: `__construct()`
Input :
Output: Istanza della classe QNSolver
Descrizione: É il costruttore della classe, e non deve mai essere invocato direttamente dall'utente. Il costruttore prevede al settaggio della variabile di classe **`__dispatch_map`** che serve per la generazione del file WSDL. Per i dettagli vedere il codice sorgente nel CD allegato.

Metodo: `validate_pmif_from_xml_schema`
Input : `model`:`String`
Output: `response`:`String`
Descrizione: Riceve in input un modello in formato PMIF 2.0 e ne effettua una validazione sintattica. Restituisce in output il risultato della validazione in formato XML. La risposta può avere uno dei seguenti formati:

- Sintassi errata:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      It seems to be Not Valid,
      according to the official
      PMIF 2.0 Schema.
   </Message>
</QNSolverReply>
```

- Sintassi corretta:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="TRUE" type="Boolean"/>
   <Message>
      It seems to be Valid,
      according to the official
      PMIF 2.0 Schema.
   </Message>
</QNSolverReply>
```

- Errore DOM:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      ERROR! An error occurred while loading PMIF 2.0 DOM model.
   </Message>
</QNSolverReply>
```

- Errore XML/SCHEMA:

```
<?xml version="1.0" encoding="UTF-8"?>
<QNSolverReply>
   <Reply value="FALSE" type="Boolean"/>
   <Message>
      ERROR! An error occurred while loading PMIF 2.0 Schema.
   </Message>
</QNSolverReply>
```

In effetti questo metodo è l'implementazione, specifica per **`model_type`** = "PMIF 2.0", del metodo pubblico **`ValidateSyntax_`**.

9 Come integrare un nuovo tool

Integrare un nuovo tool nel webservice vuol dire:

- sviluppare il file di conversione XSL da PMIF 2.0 al linguaggio accettato dal tool;
- modificare il file QNSolver.inc inserendo all'interno del vettore $tools i dati necessari al riconoscimento ed all'utilizzo del nuovo tool;
- modificare il file QNSWSTools.xml, inserendo le informazioni necessarie alla descrizione del tool per far si che i client possano sapere come invocare il nuovo tool, di quali metodi di risoluzione dispone e di quali parametri necessita.

Mentre le ultime due azioni potrebbero essere automatizzate mediante uno script che richieda i dati necessari all'utente e si occupi di inserirli nei due file di configurazione, la prima azione deve per forza di cose essere effettuata in modo manuale. Vediamo nel dettaglio questi passi.

9.1 Creazione di un XSL di conversione da PMIF 2.0 a linguaggio del Tool

La creazione di un file XSL di conversione da PMIF 2.0 al linguaggio nativo del tool non è una cosa banale. Per riuscire a scrivere un buon file XSL si deve conoscere esattamente:

- il tool, il suo funzionamento e la sintassi del linguaggio usato;
- la sintassi di PMIF 2.0 ed il significato semantico, in termini di teoria delle code, degli elementi definiti in un modello PMIF 2.0;
- la sintassi di XML;
- la sintassi di XSL/XSLT;
- la sintassi di XQuery.

Inoltre, dato che PMIF potrebbe non definire tutti gli elementi necessari alla conversione nel linguaggio nativo del tool, si devono implementare dei metodi per ricavare i dati mancanti, per armonizzare le unità di misura, o comunque per avvisare l'utente nel caso non sia possibile procedere alla conversione.

Una volta creato il file XSL e configurato il web service per il suo utilizzo, si potrà procedere alla conversione dei modelli PMIF in linguaggio sorgente del tool, passando sia il modello PMIF che l'XSL di conversione al processore XSLT di PHP5 (Figura 9.1).

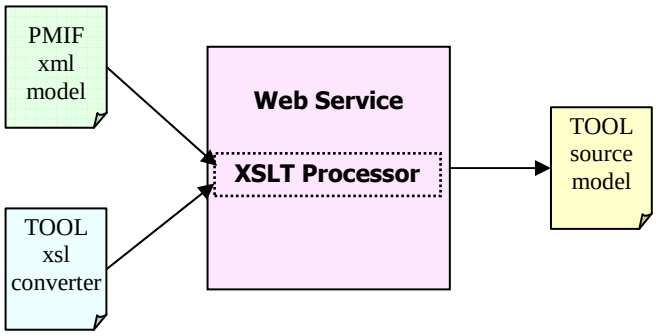


Figura 9.1 - Il processo di conversione con XSLT

Va detto che potrebbe non essere possibile scrivere un file di conversione XSL per determinati tool con sintassi particolari o restrizioni specifiche sulla formattazione del codice. Le motivazioni e le giustificazioni a questo fatto sono discusse da C. M. Llado e C. U. Smith e documentate nel dettaglio in [18]. In questo sfortunato caso si dovrà estendere il webservice, aggiungendo dei metodi privati per la conversione mediante DOM, caricando in memoria tutto il file XML che descrive il modello PMIF e procedere alla conversione internamente al web service (Figura 9.2).

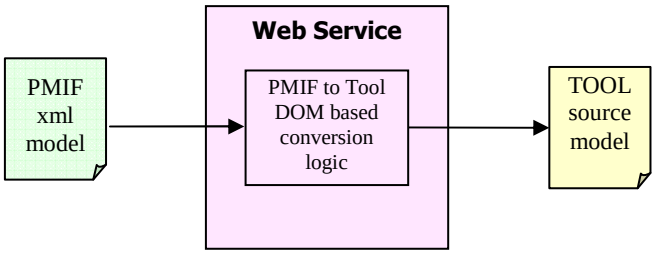


Figura 9.2 - Il processo di conversione senza XSLT

Oppure tramite un wrapper esterno al web service (Figura 9.3):

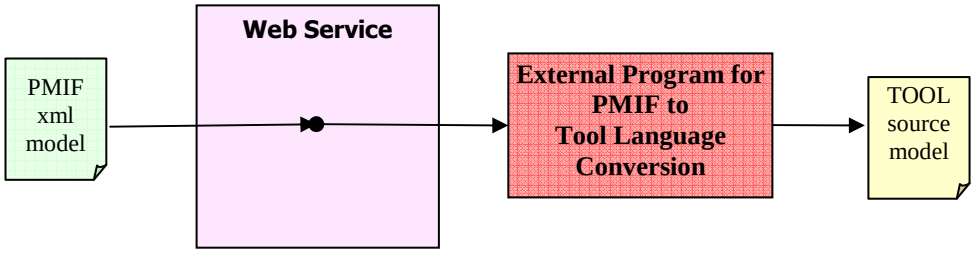


Figura 9.3 - Il processo di conversione tramite wrapper esterno

I casi in cui non è possibile procedere alla conversione con XSL, sono dovuti quindi a cause che possono dipendere da:

- formato del linguaggio del tool: formati binari non testuali, formati tabellari con restrizioni sulla posizione e formattazione del codice;
- carenze di PMIF: impossibilità di specificare le distribuzioni di probabilità relative al processo di arrivo e di servizio dei clienti e, in generale, impossibilità di descrivere la natura di un centro di servizio nella notazione di Kendall;
- dati incompleti nel modello PMIF: PMIF considera opzionale la specifica del numero di visite, mentre questa è richiesta da alcuni tool. Nella documentazione di PMIF [14] si giustifica questa opzionalità con il fatto che il numero di visite è deducibile dalle routing probabilities. Tuttavia, calcolare il numero di visite partendo dalle probabilità di transizione, richiede il calcolo di una sommatoria che presuppone l'uso di variabili inesistenti in XSL.

Nel caso il linguaggio di destinazione non sia particolarmente complicato e richieda solo le informazioni presenti nel modello PMIF 2.0, si può allora tentare di scrivere un file di conversione XSL. Un buon punto di partenza è quello di vedere come sono stati sviluppati gli XSL dei tool già integrati e cercare di capire come adattarli al nuovo tool. Sfortunatamente, al momento di scrivere questa tesi, la casistica era molto limitata, e l'unico file di conversione disponibile era quello da PMIF 2.0 a QNAP, sviluppato da C. U. Smith ed altri in [14]. Si è dovuto quindi per forza di cose prendere spunto da questo file per sviluppare tutti i successivi file di conversione, cercando di adattare ed interpretare le scelte fatte da Smith e Llado nella stesura del file XSL.

Dare una regola generale per scrivere il file XSL di conversione non è possibile. Questo perché il file XSL è strettamente correlato alla sintassi del linguaggio di destinazione. I fattori che intervengono sono troppi e troppo variegati per poter definire uno schema di massima e quindi l'unica cosa che si può fare è vedere un esempio e commentarlo.

Consideriamo, ad esempio, il tool PDQ ed analizziamo il suo file di conversione.

La sintassi generale di un file XSL con output in formato testo, prevede la presenza obbligatoria dei seguenti marcatori xml:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
...
<xsl:output method="text"/>
<xsl:template match="/">
        ...
</xsl:template>
</xsl:stylesheet>
```

Il nostro webservice consente di poter specificare dei parametri, da passare al file XSL per poter fornire, ad esempio, il metodo di risoluzione da utilizzare o il numero di job su cui basare le misure. La sintassi generale è la seguente:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:param name="sulution_method">EXACT</xsl:param>
<xsl:param name="number_of_jobs">100</xsl:param>
<xsl:output method="text"/>
<xsl:template match="/">
        ...
</xsl:template>
</xsl:stylesheet>
```

Attualmente gli unici due parametri supportati ed implementati nel webservice sono appunto “solution_method” e “number_of_jobs”. La sintassi di XSL prevede che i marcatori <xsl:param> debbano per forza di cose essere posti subito al di sotto del marcatore principale <xsl:stylesheet>, pena il non riconoscimento dei parametri. Come si può notare dall’esempio, vengono specificati dei valori di default: EXACT per solution_method e 100 per number_of_jobs. Questa è una buona regola da seguire, in quanto il client potrebbe dimenticare di fornire al webservice il parametro richiesto. In questo modo non si generano errori di conversione e soluzione in quanto il processore XSL se non trova un parametro applica il valore di default.

A questo punto, avendo specificato come formato di output “text” mediante il tag:

```
<xsl:output method="text"/>
```

tutto ciò che metteremo all’interno del marcatore:

```
<xsl:template match="/"> … </xsl:template>
```

verrà fornito in output come risultato.

Può essere comodo definire alcune variabili globali per memorizzare alcuni dati particolari, come ad esempio il nome del modello. Nei file XSL, abbiamo definito sempre le seguenti variabili:

```
...
<xsl:template match="/">
<xsl:variable name="ModelNameASIS" select="QueueingNetworkModel/@Name"/>
<xsl:variable name="ModelName" select="translate(normalize-space($ModelNameASIS),' ','_')"/>
<xsl:variable name="ModelDescription" select="QueueingNetworkModel/@Description"/>
...
```

Queste variabili contengono il nome del modello così com’è riportato nel file PMIF (ModelNameASIS), il nome del modello “normalizzato” da usare come identificatore per il modello nel linguaggio del tool (ModelName) e la descrizione del modello (ModelDescription).

Queste informazioni, quando possibile, vengono messe nel codice sorgente generato, nella parte iniziale sotto forma di commento:

```
...
<xsl:template match="/">
<xsl:variable name="ModelNameASIS" select="QueueingNetworkModel/@Name"/>
<xsl:variable name="ModelName" select="translate(normalize-space($ModelNameASIS),' ','_')"/>
<xsl:variable name="ModelDescription" select="QueueingNetworkModel/@Description"/>
// This is a PDQ Model:
//  - Name: <xsl:value-of select="$ModelName"/><xsl:if test="$ModelName != $ModelNameASIS"> AKA
<xsl:value-of select="$ModelNameASIS"/></xsl:if>
//  - Description: <xsl:value-of select="$ModelDescription"/>
```

```
//  - Date-Time: <xsl:value-of select="QueueingNetworkModel/@Date-Time"/>
// Generated from the original PMIF 2.0 Model using PMIF2_to_PDQ.xsl
// that was part of the Queueing Network Solver Webservice Project
...
```

In questo caso il linguaggio di PDQ da noi introdotto, prevede il doppio back-slash "//" come marcatore di commento.

Un altro blocco di codice che risulta conveniente mettere nel file XSL è quello per riconoscere il tipo di modello che si sta trattando. La casistica prevede che si possano modellare in PMIF 2.0 i seguenti cinque tipi di reti di code:

- Single Chain Open Queueing Network Models
- Single Chain Close Queueing Network Models
- Multiple Chain Open Queueing Network Models
- Multiple Chain Closed Queueing Network Models
- Multiple Chain Mixed Queueing Network Models

Per determinare a quale tipo appartiene un modello, si può usare il seguente codice XSL:

```
<xsl:if test="count(QueueingNetworkModel/Workload/*)=1">
<xsl:if test="count(QueueingNetworkModel/Workload/OpenWorkload)=1">
//
// Model Type: Single-Chain Open Queueing Network Model
//
</xsl:if>
<xsl:if test="count(QueueingNetworkModel/Workload/ClosedWorkload)=1">
//
// Model Type: Single-Chain Closed Queueing Network Model
//
</xsl:if>
</xsl:if>


<xsl:if test="count(QueueingNetworkModel/Workload/*)>1">
<xsl:if test="(count(QueueingNetworkModel/Workload/OpenWorkload)>=1) and
(count(QueueingNetworkModel/Workload/ClosedWorkload)=0)">
//
// Model Type: Multiple-Chain Open Queueing Network Model
//
</xsl:if>
<xsl:if test="(count(QueueingNetworkModel/Workload/OpenWorkload)=0) and
(count(QueueingNetworkModel/Workload/ClosedWorkload)>=1)">
//
// Model Type: Multiple-Chain Closed Queueing Network Model
//
</xsl:if>
<xsl:if test="(count(QueueingNetworkModel/Workload/OpenWorkload)>=1) and
(count(QueueingNetworkModel/Workload/ClosedWorkload)>=1)">
//
// Model Type: Multiple-Chain Mixed Closed/Open Queueing Network Model
//
</xsl:if>
</xsl:if>
```

Questo frammento di codice non fa altro che contare quanti workload vengono definiti nel file PMIF, se ne viene definito uno solo allora siamo nel caso di single-chain, se ne vengono definiti più di uno siamo nel caso multiple-chain. Successivamente si contano quanti di questi sono definiti come OpenWorkload e quanti come ClosedWorkload per dedurre il tipo di rete modellata.
Se il tool da integrare supporta solo uno dei tipi di rete possibili, allora al posto del commento si può mettere il codice necessario alla conversione, oppure un messaggio d'errore che avvisi l'utente dell'impossibilità di proseguire con la conversione.

Nel caso di PDQ, il codice di conversione inizia con:

```
INIT "<xsl:value-of select="$ModelName"/><xsl:if test="string-length($ModelName)=0">UNNAMED</xsl:if>"
```

Questo frammento di codice genera il codice PDQ per inizializzare il modello, assegnandogli un nome. Notare il fatto che viene utilizzata la clausola <xsl:if…> per verificare che effettivamente sia stato fornito un nome al modello, in caso contrario viene dato il nome di default “UNNAMED”.

Successivamente si analizzano tutti i nodi di tipo server e workunitserver, e per ognuno di essi viene generato il relativo codice PDQ del tipo: `CREATENODE “nome” CEN schedulingpolicy`:

```
// Network Nodes Description
...
<xsl:if test="count(QueueingNetworkModel/Node/Server)!=0">

// SERVER
<xsl:for-each select="QueueingNetworkModel/Node/Server">
CREATENODE "<xsl:value-of select="@Name"/>" CEN <xsl:value-of select="@SchedulingPolicy"/> // Number
of Processing Units = <xsl:value-of select="@Quantity"/>
</xsl:for-each>
</xsl:if>
<xsl:if test="count(QueueingNetworkModel/Node/WorkUnitServer)!=0">

// WORKUNITSERVER
<xsl:for-each select="QueueingNetworkModel/Node/WorkUnitServer">
VAR $<xsl:value-of select="@Name"/>_ServiceTime = <xsl:value-of select="@ServiceTime"/>
CREATENODE "<xsl:value-of select="@Name"/>" CEN <xsl:value-of select="@SchedulingPolicy"/> // Number
of Processing Units = <xsl:value-of select="@Quantity"/>
</xsl:for-each>
</xsl:if>
...
```

In PDQ non esiste la distinzione tra nodi di tipo SERVER e nodi di tipo WORKUNITSERVER, quindi le dichiarazioni saranno quasi identiche.

Segue la generazione del codice relativo ai carichi di lavoro (Workloads):

```
// Workloads Description
<xsl:if test="count(QueueingNetworkModel/Workload/OpenWorkload)!=0">
// Open Workloads in PMIF 2.0 are TRANSACTION Workloads in PQD
<xsl:for-each select="QueueingNetworkModel/Workload/OpenWorkload">
CREATEOPEN "<xsl:value-of select="@WorkloadName"/>" <xsl:value-of select="@ArrivalRate"/>
</xsl:for-each>
</xsl:if>
<xsl:if test="count(QueueingNetworkModel/Workload/ClosedWorkload)!=0">
// Closed Workloads in PMIF 2.0 are TERMINAL (if NumberOfJobs &gt; 0) or BATCH (if NumberOfJobs = 0)
Workloads in PQD
<xsl:for-each select="QueueingNetworkModel/Workload/ClosedWorkload">
CREATECLOSED "<xsl:value-of select="@WorkloadName"/>" <xsl:if test="@ThinkTime =
0">BATCH</xsl:if><xsl:if test="@ThinkTime > 0">TERM</xsl:if><xsl:text> </xsl:text><xsl:value-of
select="@NumberOfJobs"/><xsl:text> </xsl:text><xsl:value-of select="@ThinkTime"/>
</xsl:for-each>
</xsl:if>
```

La generazione del codice relativo alle richieste di servizio:

```
// Service Requests Description
<xsl:if test="count(QueueingNetworkModel/ServiceRequest/WorkUnitServiceRequest)!=0">

// Work Unit Service Request
<xsl:for-each select="QueueingNetworkModel/ServiceRequest/WorkUnitServiceRequest">
SETVISITS "<xsl:value-of select="@ServerID"/>" "<xsl:value-of select="@WorkloadName"/>" <xsl:value-of
select="@NumberOfVisits"/><xsl:text> </xsl:text>$<xsl:value-of
select="@ServerID"/>_ServiceTime</xsl:for-each>
</xsl:if>
<xsl:if test="count(QueueingNetworkModel/ServiceRequest/DemandServiceRequest)!=0">

// Demand Service Request
<xsl:for-each select="QueueingNetworkModel/ServiceRequest/DemandServiceRequest">
SETVISITS "<xsl:value-of select="@ServerID"/>" "<xsl:value-of select="@WorkloadName"/>" <xsl:value-of
select="@NumberOfVisits"/><xsl:text> </xsl:text><xsl:value-of select="@ServiceDemand div
@NumberOfVisits"/>
</xsl:for-each>
</xsl:if>
<xsl:if test="count(QueueingNetworkModel/ServiceRequest/TimeServiceRequest)!=0">

// Time Service Request
<xsl:for-each select="QueueingNetworkModel/ServiceRequest/TimeServiceRequest">
SETVISITS "<xsl:value-of select="@ServerID"/>" "<xsl:value-of select="@WorkloadName"/>" <xsl:value-of
select="@NumberOfVisits"/><xsl:text> </xsl:text><xsl:value-of select="@ServiceTime"/>
</xsl:for-each>
</xsl:if>
```

Ed infine il codice per la risoluzione del modello ed il report dei risultati:

```
// Solve the model using one of EXACT, APPROX or CANON methods
SOLVE <xsl:value-of select="$sulution_method"/>
REPORT

EXIT
</xsl:template>
</xsl:stylesheet>
```

Notare che nel generare l'istruzione SOLVE si utilizza il parametro solution_method che era stato passato all'XSL.

Come si può vedere il codice XSL è molto poco leggibile, ma non si può fare diversamente, in quanto i marcatori XML di XSL lavorano "in linea" con il testo da fornire in output e quindi non è possibile usare una formattazione migliore perché non si otterrebbe il risultato voluto.

Scendere ulteriormente nel dettaglio sarebbe inutile, in quanto come già detto non è possibile dare una regola generale per scrivere un file di conversione XSL, in quanto esso è strettamente legato alla sintassi del linguaggio del tool.

9.2 Aggiunta di un nuovo tool nel file QNSolver.inc

Il file QNSolver.inc, è strutturato in diverse sezioni:

- blocco di definizione di costanti globali
- blocco di definizione del vettore $TOOLS
- blocco di definizione delle funzioni di output XML
- blocco di definizione delle funzioni di manipolazione del vettore $TOOLS

Per integrare un nuovo tool si deve agire sulla seconda e se necessario sulla prima sezione, mentre non è necessario toccare le altre due.

Il vettore $TOOLS è un vettore "associativo" PHP 5 con la seguente sintassi:

```
vettore_tools     ←    $TOOLS = array ( tools_list );
tools_list        ←    "tool_id" => array( tool_def ) [ , tools_list ]
tool_def          ←    xsl , xsl_param , exe , exe_param
xsl               ←    "XSL" => { global_const | "sys_path_to_xsl" }
xsl_param         ←    "XSL_PARAM" => array( xsl_param_list )
xsl_param_list    ←    "xsl_param_name" => "variable_name" [ , xsl_param_list ]
xsl_param_name    ←    solution_method | number_of_jobs
exe               ←    "EXE" => { global_const | "sys_path_to_exe" }
exe_param         ←    "EXE_PARAM" => array( exe_param_list )
exe_param_list    ←    "exe_param_name" => "variable_name" [ , exe_param_list ]
exe_param_name    ←    AFTHER | BEFORE
variable_name     ←    method | params
tool_id           ←    valid_char[valid_char]
valid_char        ←    chars | numbers
chars             ←    A|B|C|D|E|F|G|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
numbers           ←    0|1|2|3|4|5|6|7|8|9
global_const      ←    Must be a valid PHP 5 constant ID defined somewhere in QNSolver.inc
sys_path_to_exe   ←    Must be a string representing a valid system path to the tool executable
```

Facciamo un esempio per chiarire meglio il concetto espresso, riportando la prima e la seconda sezione del file QNSolver.inc usata nella versione per MS Windows™ del webservice:

```
// Default folder for all the xsl and xsd files
// For Windows path, remember to use the \\ escape sequence for path "\" specification
define("XSLXSD_DIR","D:\\Programmi\\HTTP\\Apache Group\\Apache2\\htdocs\\");
// Tools translation Style Sheet name and location
define("PMIF_TO_PDQ",XSLXSD_DIR."PMIF2_to_PDQ.xsl");
define("PMIF_TO_QNAP_XSLT",XSLXSD_DIR."PMIF2_to_QNAP.xsl");
define("PMIF_TO_SHARPE_PFQN",XSLXSD_DIR."PMIF2_to_SHARPE_PFQN.xsl");
define("PMIF_TO_OPENQN",XSLXSD_DIR."PMIF2_to_OPENQN.xsl");
define("PMIF_TO_CLOSEDQN",XSLXSD_DIR."PMIF2_to_CLOSEDQN.xsl");
define("PMIF_TO_MVACCKSW",XSLXSD_DIR."PMIF2_to_MVACCKSW.xsl");
define("PMIF_TO_PMVA",XSLXSD_DIR."PMIF2_to_PMVA.xsl");
define("PMIF_TO_PEPSY", XSLXSD_DIR."PMIF2_to_PEPSY.xsl");
// Tools executable location and parameters
// Remember to leave a white space as the last char of the string
// and to include all the needed cmd line switch or input redirection
define("PDQ_EXE","C:\\PDQ\\PDQBATCH.bat ");
define("QNAP_EXE","");
define("SHARPE_EXE","C:\\SHARPE\\sharpe.exe ");
define("OPENQN_EXE","C:\\QSolver-1\\OPENQN.exe ");
define("CLOSEDQN_EXE","C:\\QSolver-1\\CLOSEDQN.exe ");
define("MVACCKSW_EXE","C:\\mva_cck_sw\\mva.exe ");
define("PMVA_EXE","C:\\PMVAWin\\pmva.exe < ");
define("PEPSY_EXE","");
```

```
$TOOLS = array(
            "PDQ"      => array("XSL" => PMIF_TO_PDQ,
                                "XSL_PARAM" => array("solution_method" => "method"),
                                "EXE" => PDQ_EXE,
                                "EXE_PARAM" => array()
                               ),
            "QNAP"     => array("XSL" => PMIF_TO_QNAP_XSLT,
                                "XSL_PARAM" => array(),
                                "EXE" => QNAP_EXE,
                                "EXE_PARAM" => array()
                               ),
            "SHARPE"   => array("XSL" => PMIF_TO_SHARPE_PFQN,
                                "XSL_PARAM" => array(),
                                "EXE" => SHARPE_EXE,
                                "EXE_PARAM" => array()
                               ),
            "OPENQN"   => array("XSL" => PMIF_TO_OPENQN,
                                "XSL_PARAM" => array(),
                                "EXE" => OPENQN_EXE,
                                "EXE_PARAM" => array()
                               ),
            "CLOSEDQN" => array("XSL" => PMIF_TO_CLOSEDQN,
                                "XSL_PARAM" => array(),
                                "EXE" => CLOSEDQN_EXE,
                                "EXE_PARAM" => array()
                               ),
            "MVACCKSW" => array("XSL" => PMIF_TO_MVACCKSW,
                                "XSL_PARAM" => array(),
                                "EXE" => MVACCKSW_EXE,
                                "EXE_PARAM" => array("AFTER" => "method")
                               ),
            "PMVA"     => array("XSL" => PMIF_TO_PMVA,
                                "XSL_PARAM" => array("solution_method" => "method"),
                                "EXE" => PMVA_EXE,
                                "EXE_PARAM" => array()
                               ),
            "PEPSY"    => array("XSL" => PMIF_TO_PEPSY,
                                "XSL_PARAM" => array(),
                                "EXE" => PEPSY_EXE,
                                "EXE_PARAM" => array("solution_method" => "params")
                               )
        );
```

Il significato degli elementi “XSL” e “EXE” è abbastanza chiaro, il primo è il path completo al file di conversione XSL del tool, il secondo è il path completo al file eseguibile del tool. Per quanto riguarda il significato di “XSL_PARAM” e “EXE_PARAM”, spendiamo invece qualche parola in più.

Il vettore XSL_PARAM è un vettore associativo, che associa il parametro definito nel file di conversione XSL (vedi paragrafo 9.1) con una delle due variabili (*method* o *params*) passate ai metodi del webservice (vedi paragrafo 8.5.3). Il valore attuale di queste variabili viene passato al file XSL in fase di conversione e utilizzato per generare il codice nel linguaggio sorgente del tool.

Il vettore EXE_PARAM è un vettore associativo, che può definire due sole “chiavi”: AFTER e BEFORE. Alcuni tool, come ad esempio PEPSY e MVACCKSW, non consentono di specificare nel codice sorgente il metodo di risoluzione, ma vogliono che il metodo sia specificato sulla linea di comando del file eseguibile. Il parametro AFTER consente di specificare una delle due variabili “method” o “params”, passate in fase di invocazione di uno dei metodi del webservice, e di inserire il valore attuale di questa variabile sulla linea di comando[†] dopo (*after* in inglese) il nome del file sorgente. Mentre il parametro BEFORE consente di inserire il valore attuale della variabile prima (*before* in inglese) del nome del file.

Es.:

```
path\eseguibile.exe <contenuto di before> <nome del file> <contenuto di after>
```

[†] Questo meccanismo ci consente di poter integrare quei tool che necessitano di specificare parametri variabili sulla linea di comando, parametri che non sarebbero altrimenti specificabili. Nel nostro web service, non sono stati implementati meccanismi di sicurezza per proteggere il web service da usi impropri. Il poter passare parametri variabili sulla linea di comando senza ulteriori controlli apre una falla di sicurezza che deve essere attentamente tenuta in considerazione nel caso si decidesse di pubblicare il servizio su internet. Come estensione futura di questa tesi si dovrebbero prevedere dei controlli aggiuntivi sui parametri e gli input forniti dall’utilizzatore del web service.

9.3 Aggiunta di un nuovo tool nel file QNSWSTools.xml

Il file QNSWSTools.xml è, come si intuisce dall'estensione, un file XML. Questo file contiene tutte le informazioni necessarie all'uso dei tool integrati da parte dei client. Il suo utilizzo da parte dei client non è obbligatorio, nel senso che se il client conosce i tools integrati e sa di quali metodi dispone e di quali parametri necessità, può anche fare a meno di scaricarlo dal webservice.
Tuttavia, nell'ottica in cui è stata sviluppata questa tesi, ovvero quella di creare un webservice distribuito multi-tools, un eventuale client generico, e non specializzato per un singolo fornitore di webservice, deve avere la possibilità di sapere quali tools mette a disposizione un fornitore di webservice, di quali metodi risolutivi dispone e di quali parametri necessita. Lo scopo di QNSWSTools.xml è proprio questo, fornire all'utilizzatore lato client tutte le informazioni di cui necessità per utilizzare correttamente un tool integrato nel webservice.

Diamo la sintassi di questo semplice file in formato DTD:

```
<!ELEMENT TOOLSLIST (TOOL+) >
<!ELEMENT TOOL (SAVE? ,SOLVE? ,SOLVEASIS?) >
   <!ATTLIST TOOL NAME CDATA #REQUIRED  >
   <!ATTLIST TOOL VERSION CDATA #IMPLIED  >
   <!ATTLIST TOOL DESCRIPTION CDATA #REQUIRED  >
   <!ATTLIST TOOL AUTHOR CDATA #REQUIRED  >
<!ELEMENT SAVE (METHOD* | PARAMS*) >
<!ELEMENT SOLVE (METHOD* | PARAMS*) >
<!ELEMENT SOLVEASIS (METHOD* | PARAMS*) >
<!ELEMENT METHOD (PARAMS*) >
   <!ATTLIST METHOD NAME CDATA #REQUIRED  >
   <!ATTLIST METHOD DESCRIPTION CDATA #REQUIRED  >
<!ELEMENT PARAMS EMPTY >
   <!ATTLIST PARAMS NAME CDATA #REQUIRED  >
   <!ATTLIST PARAMS TYPE CDATA #REQUIRED  >
```

L'elemento principale del file XML è TOOLSLIST, che deve avere al suo interno almeno un elemento TOOL definito.

```
<?xml version="1.0" encoding="UTF-8" ?>
<TOOLSLIST>
      <TOOL ... >
        ...
      </TOOL ... >
      ...
</TOOLSLIST>
```

Ogni elemento TOOL deve avere gli attributi il NAME, DESCRIPTION e AUTHOR e può specificare l'attributo VERSION.

```
<TOOL NAME="PDQ" VERSION="3.0" DESCRIPTION="Pretty Damn Quick Solver"
                                          AUTHOR="Prof. Neil J. Gunther" >
      ...
</TOOL>
```

L'elemento TOOL può contenere gli elementi SAVE, SOLVE, SOLVEASIS. SAVE corrisponde al metodo **`Transform_()`** del webservice, mentre SOLVE e SOLVEASIS corrispondono rispettivamente al metodo **`Solve_()`** con i parametri: **`model_type ≠ tool`** e al metodo **`Solve_()`** con i parametri: **`model_type = tool`**. Quest'ultimo serve per consentire al webservice la risoluzione di modelli in formati diversi da PMIF 2.0, ovvero espressi nel linguaggio

nativo dei tools. Per ora, non sono previste altre operazioni oltre a SAVE, SOLVE e SOLVEASIS in quanto gli altri metodi del webservice (Copyright_ , ValidateSyntax_ , ValidateSemantic_ , GetModelDescription_ e GetToolsList_) sono sempre disponibili e non dipendono dai tool integrati, e quindi un client ha tutte le informazioni necessarie per la loro invocazione. Es.:

```
<SAVE>
      <METHOD NAME="EXACT" DESCRIPTION="Exact Solution" />
      <METHOD NAME="APPROX" DESCRIPTION="Approximate Solution" />
      <METHOD NAME="CANON" DESCRIPTION="Canonical Solution" />
</SAVE>
<SOLVE>
      <METHOD NAME="EXACT" DESCRIPTION="Exact Solution" />
      <METHOD NAME="APPROX" DESCRIPTION="Approximate Solution" />
      <METHOD NAME="CANON" DESCRIPTION="Canonical Solution" />
</SOLVE>
<SOLVEASIS/>
```

Se il tool non ha bisogno di specificare dei metodi ma necessita invece di qualche parametro, allora invece di usare METHOD si deve usare PARAMS. Es.:

```
<SAVE>
      <PARAMS NAME="Number of Jobs" TYPE="INTEGER" />
</SAVE>
<SOLVE>
      <PARAMS NAME="Number of Jobs" TYPE="INTEGER" />
</SOLVE>
<SOLVEASIS/>
```

Infine se si deve specificare un metodo di risoluzione e anche dei parametri si deve annidare il tag PARAMS dentro METHOD. Es.:

```
<SAVE>
      <METHOD NAME="MVA" DESCRIPTION="Exact MVA Solution" >
            <PARAMS NAME="Number of Jobs" TYPE="INTEGER" />
      </METHOD>
      <METHOD NAME="ASYMP" DESCRIPTION="Bard's algorithm for .." >
            <PARAMS NAME="Number of Jobs" TYPE="INTEGER" />
      </METHOD>
      <METHOD NAME="SCHWEITZER" DESCRIPTION="Schweitzer's algorithm" >
            <PARAMS NAME="Number of Jobs" TYPE="INTEGER" />
      </METHOD>
      <METHOD NAME="LIN" DESCRIPTION="Linearizer algorithm .." >
            <PARAMS NAME="Number of Jobs" TYPE="INTEGER" />
      </METHOD>
</SAVE>
<SOLVE>
      <METHOD NAME="MVA" DESCRIPTION="Exact MVA Solution" >
            <PARAMS NAME="Number of Jobs" TYPE="INTEGER" />
      </METHOD>
      <METHOD NAME="ASYMP" DESCRIPTION="Bard's algorithm for .." >
            <PARAMS NAME="Number of Jobs" TYPE="INTEGER" />
      </METHOD>
      <METHOD NAME="SCHWEITZER" DESCRIPTION="Schweitzer's algorithm" >
            <PARAMS NAME="Number of Jobs" TYPE="INTEGER" />
      </METHOD>
      <METHOD NAME="LIN" DESCRIPTION="Linearizer algorithm .." >
            <PARAMS NAME="Number of Jobs" TYPE="INTEGER" />
      </METHOD>
</SOLVE>
<SOLVEASIS/>
```

10 Implementazione di CLIENT semplici per l'uso del webservice

Per testare il webservice e per dimostrare la generalità della sua implementazione, sono stati sviluppati tre diversi client in tre linguaggi differenti:

- Client a linea di comando scritto in PHP 5;
- Client ipertestuale scritto in JavaScript con tecnologia AJAX;
- Client grafico scritto in Borland® Delphi™ 2005 for Microsoft® Windows™ Architect Ed.

Il primo client, molto semplice, riceve in input dalla linea di comando il nome di un file in formato PMIF 2.0, il nome del tool con cui si vuole risolvere il modello ed eventuali parametri come il metodo di risoluzione ed il numero di job. Esegue una validazione sintattica, e se il modello è sintatticamente valido viene invocato il metodo di risoluzione richiesto.
Il secondo client, scritto in HTML più JavaScript, utilizza la tecnologia AJAX per effettuare le chiamate al server, utilizzando una libreria JavaScript per l'implementazione delle chiamata SOAP.
Il terzo client verrà invece trattato nel capitolo successivo.

10.1 PHP 5 Client a linea di comando

La realizzazione di un client per l'uso del webservice, utilizzando PHP 5 + PEAR::SOAP, è abbastanza banale. Prima di tutto si deve importare l'estensione "Client" di PEAR::SOAP mediante la solita istruzione:

```
...
require_once 'SOAP/Client.php';
...
```

Il passo successivo consiste nel creare un'istanza dell'oggetto SOAP_WSDL, passando come parametro l'URL del file WSDL del webservice:

```
...
$wsdl_url = 'http://localhost/QueueingNetwork_Solver_Service.php?wsdl';
$wsdl = new SOAP_WSDL($wsdl_url);
...
```

Ora si può chiedere all'oggetto appena creato di generare il codice di "proxy" necessario all'invocazione dei metodi della classe remota:

```
...
$QNSolver = $wsdl->getProxy();
...
```

La variabile $QNSolver, appena creata tramite getProxy(), è un'istanza della classe QNSolver remota ed avrà quindi gli stessi metodi. Infatti le seguenti invocazioni sono perfettamente lecite e funzionanti:

```
...
$risposta = $QNSolver->ValidateSyntax_($myModel,'PMIF 2.0');
...
$risposta = $QNSolver->ValidateSemantic_($myModel,'PMIF 2.0');
...
$risposta = $QNSolver->Transform_($myModel,'PMIF 2.0','PDQ','EXACT' ,'');
...
$risposta = $QNSolver->Solve_($myModel,'PMIF 2.0','PDQ','EXACT' ,'');
...
```

Il client viene invocato dalla linea di comando del sistema operativo, con la seguente sintassi:

```
php QNClient.php <option> <model> <model_type> <tool> <method> <params>
```

Dove:

- **php**: è l'interprete PHP. Se il client sta girando in ambiente Linux e si danno al file QNClient.php i diritti di esecuzione, allora può essere omesso. In quest'ultimo caso il client prevede che l'interprete PHP sia installato nella directory: /usr/local/bin/php. In caso contrario dovrà essere cambiata la prima riga del file QNClient.php per puntare al path corretto.
- **QNClient.php**: è il sorgente del client in PHP il cui listato completo è riportato nel CD allegato. Se viene invocato senza alcun altro parametro aggiuntivo, viene visualizzata una breve descrizione su come utilizzare il client.
- **<option>**: può essere uno dei seguenti:
  - **COPYRIGHT**: richiede al client di invocare il metodo **Copyright_()** del webservice e ne visualizza il risultato a video. È ammessa anche la forma abbreviata **CPY**.
  - **TOOLS**: richiede al client di invocare il metodo **GetToolsList_()** del webservice e ne visualizza il risultato a video. È ammessa anche la forma abbreviata **TLS**.
  - **VALIDATESYNTAX**: consente di effettuare la validazione sintattica di un modello PMIF 2.0. Questa opzione chiede al client di invocare il metodo **ValidateSyntax_(model,model_type)**. È ammessa anche la forma abbreviata **VSY**.
  - **VALIDATESEMANTIC**: consente di effettuare la validazione semantica di un modello PMIF 2.0. Questa opzione richiede al client di invocare il metodo **ValidateSemantic(model,model_type)**. È ammessa anche la forma contratta **VSE**.
  - **TRANSFORM**: chiede al client di effettuare una conversione di un modello da PMIF 2.0 al linguaggio di uno dei tools integrati. Il client traduce questa richiesta in una chiamate al metodo **Transform_(model,model_type,tool,method,params)**. È ammessa anche la forma abbreviata **TRA**.
  - **SOLVE**: chiede al client di invocare il metodo di risoluzione per il modello specificato. Il client traduce questa richiesta nell'invocazione del metodo **Solve_(model,model_type,tool,method,params)**. È ammessa anche la forma abbreviata **SOL**.
- **<model>**: è il nome del file, completo di path, che contiene il modello. Il file deve essere comprensivo di path e deve rispettare la sintassi del sottostante sistema operativo.
- **<model_type>**: generalmente è "PMIF 2.0" ma nel caso dell'opzione SOLVE può anche coincidere con l'ID di qualche tool.
- **<tool>**: è un'ID che identifica uno dei tool integrati nel webservice.
- **<method>**: è uno dei metodi supportati dal tool richiesto.
- **<params>**: sono i parametri opzionali richiesti dal tool.

Nel caso il webservice non sia in esecuzione sulla stessa macchina su cui sta girando il client, allora si dovrà modificare l'url del WSDL all'interno del sorgente del client.

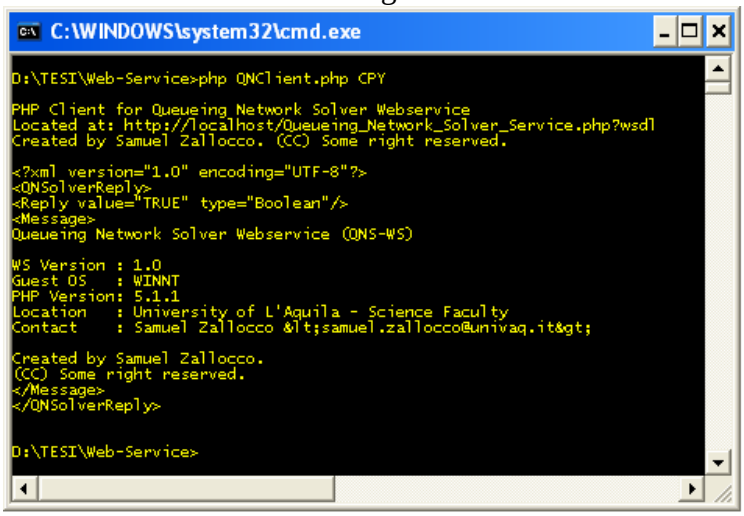


Figura 10.1 - Esempio di invocazione del metodo Copyright_( ) mediante client PHP

Il client a linea di comando visualizza il suo output a video. Per il salvataggio dei risultati, come nella più classica tradizione Unix e DOS, si può ricorrere reindirizzare l'output verso un file.

La sintassi è più o meno la stessa sia per DOS che per Unix:

- Esempio DOS:

```
C:\>
C:\> php QNClient.php SOLVE pmif_model_example.xml PMIF SHARPE > output.txt
C:\>
```

- Esempio UNIX:

```
root$
root$ php QNClient.php SOLVE pmif_model_example.xml PMIF SHARPE > output.txt
root$
```

In realtà in ambiente UNIX non è necessario specificare l'interprete PHP, perché, dando i diritti di esecuzione al file client, esso viene ricavato dalla prima riga del file QNClient.php: "`#!/usr/local/bin/php`". Quindi la sintassi di invocazione diventa:

- Esempio UNIX:

```
root$
root$ QNClient.php SOLVE pmif_model_example.xml PMIF SHARPE > output.txt
root$
```

Per salvare un modello convertito si segue la stessa procedura:

- Esempio DOS:

```
C:\>
C:\> php QNClient.php TRANSFORM pmif_model_example.xml PMIF SHARPE > sharpe_model.pfqn
C:\>
```

- Esempio UNIX:

```
root$
root$ QNClient.php TRANSFORM pmif_model_example.xml PMIF SHARPE > sharpe_model.pfqn
root$
```

10.2 JavaScript/AJAX Client su pagina web

10.2.1 Breve introduzione ad AJAX

AJAX (*Asyncronous Javascript And Xml*) è una tecnologia che sta suscitando grande interesse. Il termine AJAX venne utilizzato per la prima volta da Jesse Garrett il 18 Febbraio 2005, come titolo di un post all'interno del suo blog. Non si tratta di una nuova tecnologia né di un'invenzione rivoluzionaria ma bensì di un insieme di tecnologie esistenti da molto tempo a cui nessuno aveva mai pensato di dare un nome.
Queste tecnologie sono:

- HTML (o meglio XHTML) e CSS per la parte visuale;
- DOM (Document Object Model) manipolato attraverso JavaScript per mostrare dinamicamente le informazioni ed interagire con esse;
- L'oggetto XMLHttpRequest per l'interscambio e la manipolazione dei dati in modo asincrono tra il browser dell'utente ed il server web remoto.

Le applicazioni che utilizzano la tecnologia AJAX richiedono browser che supportino tutte le tecnologie dell'elenco esposto in precedenza. Oggi le ultime versioni dei più diffusi browser come: Mozilla Firefox, Internet Explorer, Opera, Konqueror e Safari hanno (più o meno) implementato tutte le tecnologie necessarie per il funzionamento di applicazioni AJAX, anche se persistono differenze relativamente all'oggetto XMLHttpRequest.
In una applicazione web tradizionale, l'interazione utente-browser-server avviene per mezzo di link e form HTML. L'utente deve immettere dei dati in un form, il browser invia i dati al server che li elabora e rispedisce al browser una nuova pagina con il risultato dell'elaborazione. Lo svantaggio di questo approccio è che ad ogni richiesta vengono rispediti al client tutti gli elementi che compongono la pagina e non solo i risultati dell'elaborazione. Questa ritrasmissione continua di parti comuni e statiche delle pagine web ovviamente consuma banda e richiede tempo.
Diversamente, in un'applicazione AJAX, il browser dopo aver caricato una sola volta tutte le parti comuni e statiche di una pagina/applicazione, si limita ad inviare ed a richiedere al server solo i dati da elaborare e le risposte da visualizzare. In questo modo il totale dei dati scambiati tra client e server scende drasticamente, e di conseguenza le applicazioni web risultano essere più veloci.

Nello sviluppo del client AJAX per il nostro webservice utilizziamo una versione modificata della libreria[†]: "***JavaScript SOAP Client library V 2.1 – 2006.09.08***" e del relativo esempio d'uso.
Solo a titolo d'esempio diamo una breve descrizione dell'implementazione in Javascript della chiamata al metodo Copyright_(), mentre si rimanda al sorgente completo del client, presente nel CD allegato, per maggiori dettagli.

Evidenziamo in rosso le parti di codice interessanti. Prima di tutto si deve includere la libreria soapclient.js:

```
<html>
<head>
<title>JavaScript SOAP Client for Queueing Network Solver Webservice</title>
...
<script type="text/javascript" src="soapclient.js"></script>
...
```

Si imposta l'url del webservice:

```
...
```

[†] La libreria utilizzata è stata sviluppata da Matteo Casati – http://www.guru4.net/. Le modifiche apportate riguardano essenzialmente un aggiornamento della versione del protocollo SOAP e del formato di scambio dei messaggi.

```
<script type="text/javascript">
      var url = "http://localhost/Queueing_Network_Solver_Service.php";
...
```

Ora, per ogni metodo che si vuole invocare si devono definire due funzioni, una per l'invocazione del metodo e l'altra per la gestione del risultato:

```
...
// Copyright_ Demo
function Copyright_()
{
      var pl = new SOAPClientParameters();
      SOAPClient.invoke(url, "Copyright_", pl, true, Copyright_callBack);
}

function Copyright_callBack(r)
{
      var d = document.getElementById("copyright_demo");
      d.innerHTML = r;
}
...
</script>
</head>
...
```

La funzione "Copyright_()" definisce una variabile "pl" (parameters list) che contiene gli eventuali parametri da passare al metodo del webservice. Nel nostro caso stiamo invocando il metodo Copyright_() che non necessita di parametri quindi non aggiungiamo alcun parametro. Se invece avessimo dovuto specificare qualche parametro allora avremmo dovuto aggiungere le seguenti righe di codice:

```
...
      var pl = new SOAPClientParameters();
      ...
      pl.add("model", model);
      pl.add("model_type", model_type);
      ...
...
```

L'istruzione successiva:

```
      SOAPClient.invoke(url, "Copyright_", pl, true, Copyright_callBack);
```

richiama il metodo invoke dell'oggetto SOAPClient definito in "soapclient.js", passandogli come parametri: l'url del webservice, il nome del metodo da invocare così come definito nel file WSDL, l'oggetto "pl" (di tipo SOAPClientParameters) contente la lista di parametri, "true" che sta ad indicare che vogliamo un'invocazione di tipo "sincrono" (ovvero che ci aspettiamo di ricevere subito il risultato) ed infine la funzione di *Call Back* da invocare e a cui passare il risultato.

La funzione Copyright_callback viene invocata automaticamente dal metodo invoke dell'oggetto SOAPClient, passandogli come parametro il risultato "r" dell'esecuzione del metodo del webservice.
Questa funzione non fa altro che ricercare all'interno dell'albero DOM del documento HTML la division in cui mettere il risultato ottenuto e, tramite l'attributo InnerHTML, assegnargli il risultato per essere automaticamente visualizzato dal browser.

Ora, nel body, specifichiamo il codice HTML necessario alla gestione dell'interfaccia:

```
...
<body>
...
<form id="frmDemo" name="frmDemo" action="" method="post">
<h2>DEMO 1: "Copyright_()"</h2>
<p>Call the Copyiright_ method that return some tips about webservice and his
mantainer:</p>
<input type="button" value="Call Copyright_()" onclick="Copyright_();" />
<a href="javascript:toggle('copyright_demo');">Show / Hide result</a></p>
<div id="copyright_demo" class="s"></div>
...
</form>
...
</body>
</html>
```

Le cose interessanti da notare sono il controllo “input” di tipo “button” che, tramite la keyword “onclick”, richiama la funzione JavaScript Copyright_() definita sopra, e la division con id “copyright_demo”, inizialmente vuota, che tramite la funzione di Call Back conterrà il risultato dell’invocazione.

10.2.2 Descrizione del Client AJAX

Per accedere al client AJAX si deve utilizzare un browser web (preferibilmente Internet Explorer 6.x o superiore in quanto è quello che ha dato meno problemi). Il client, pur essendo scritto in AJAX, non deve per forza di cose essere posizionato all'interno di un server web, ma può essere tranquillamente eseguito in locale, abilitando IE all'esecuzione del contenuto attivo.

I file che compongono il client AJAX sono i seguenti:

- ajaxclient.html: è il codice html di interfaccia verso l'utente;
- soapclient.js: è la libreria per l'implementazione del protocollo SOAP.

Questi due file devono obbligatoriamente essere presenti nella stessa directory.

Come per il client a linea di comando, si presuppone che il webservice sia in esecuzione sulla stessa macchina del client, ma in caso contrario si dovrà cambiare l'url del WSDL all'interno del file ajaxclient.html alla riga 18:

```
var url = "http://localhost/Queueing_Network_Solver_Service.php";
```

Una volta eseguito il client, ad esempio in Internet Explorer, verrà visualizzata una comune pagina web dalla quale è possibile accedere alle diverse funzioni del webservice.

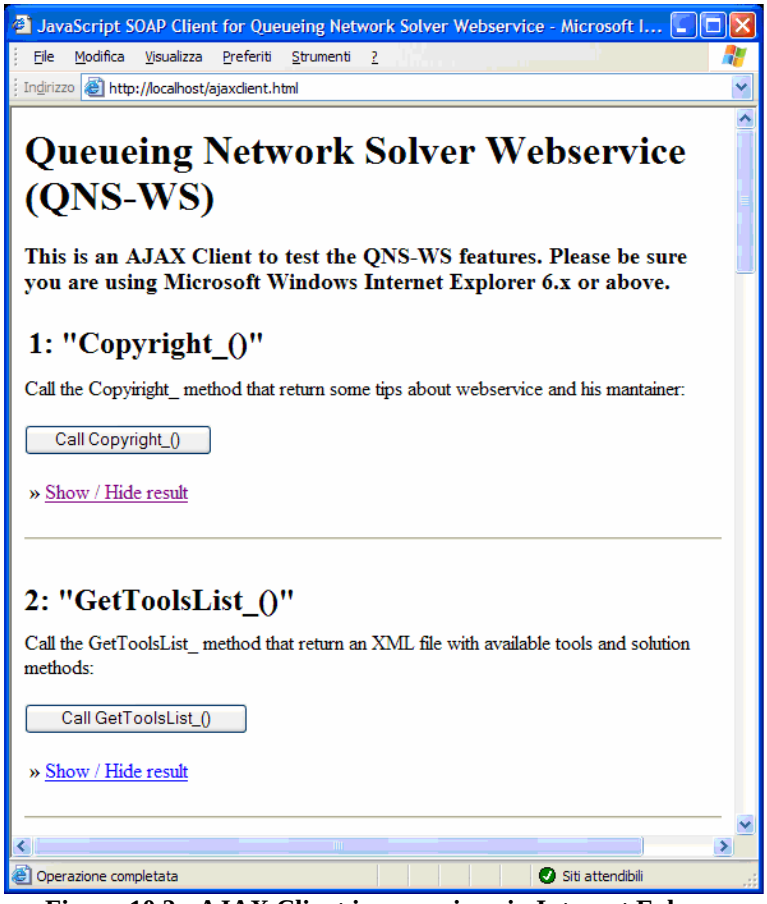


Figura 10.2 - AJAX Client in esecuzione in Internet Exlorer

Si hanno in totale 7 sezioni che coprono tutte le funzioni del webservice:

1: “Copyright_()”:
Come si vede dalla figura, questa sezione è composta da un solo tasto. Cliccando su di esso verrà visualizzata la risposta del webservice nella “division” presente sotto la scritta “Show/Hide result”. Cliccando su questa scritta la “division” viene nascosta se visibile o visualizzata se nascosta.

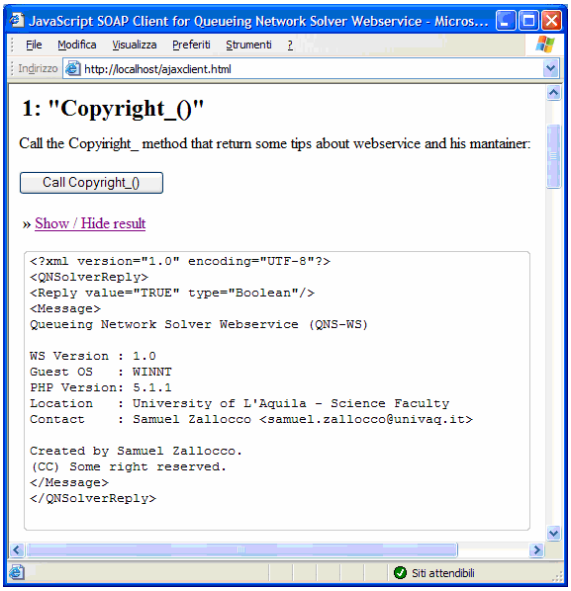


Figura 10.3 - AJAX Client: invocazione di Copyright_

2: “GetToolsList_()”:
Anche questa sezione è composta da un solo tasto, cliccato il quale appare nella division sottostante il risultato dell’invocazione del metodo GetToolsList_().

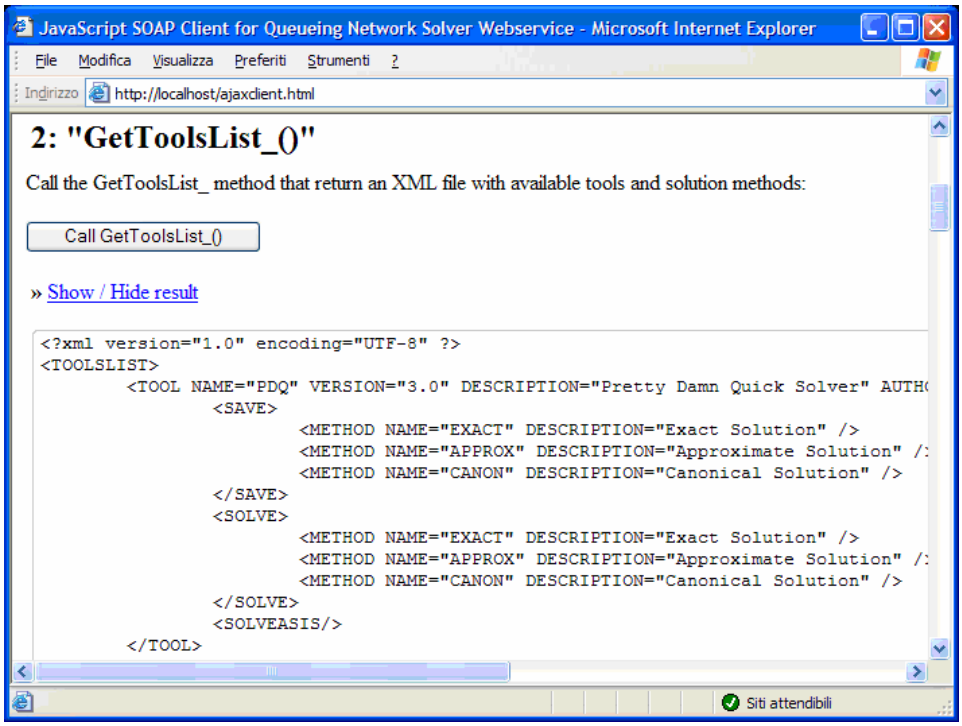


Figura 10.4 - AJAX Client: invocazione di GetToolsList_

3: “ValidateSyntax_(model, model_type)”:
Questa sezione prevede che vengano specificati due parametri model e model_type. Il primo deve essere un file PMIF 2.0 ed il secondo per ora può essere solo “PMIF 2.0”. I due parametri vengono specificati mediante un form e l’uso di un oggetto HTML “textarea” e dell’oggetto “select”. A fianco della textarea è presente un link “Example”, ciccandolo viene riportato nella textarea un file di esempio in formato PMIF 2.0. Una volta specificati i due parametri, cliccando sul tasto “Call…” viene invocato il metodo ValidateSyntax_ del webservice.

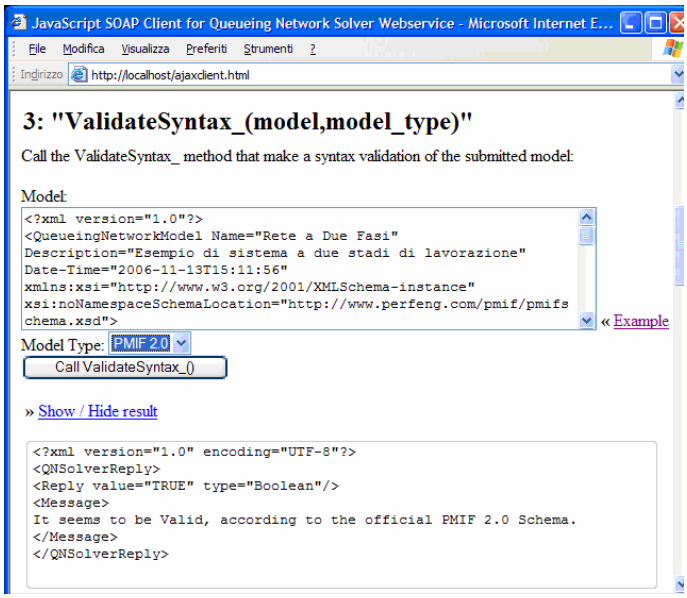


Figura 10.5 - AJAX Client: invocazione di ValidateSyntax_

4: “ValidateSemantic_(model, model_type)”:
Anche questa sezione, come la precedente, prevede di dover specificare il modello ed il tipo del modello sempre mediante l’uso di due oggetti “textarea” e “select”. Solo che questa volta, cliccando sul tasto “Call…”, viene invocato il metodo ValidateSemantic_ del webservice.

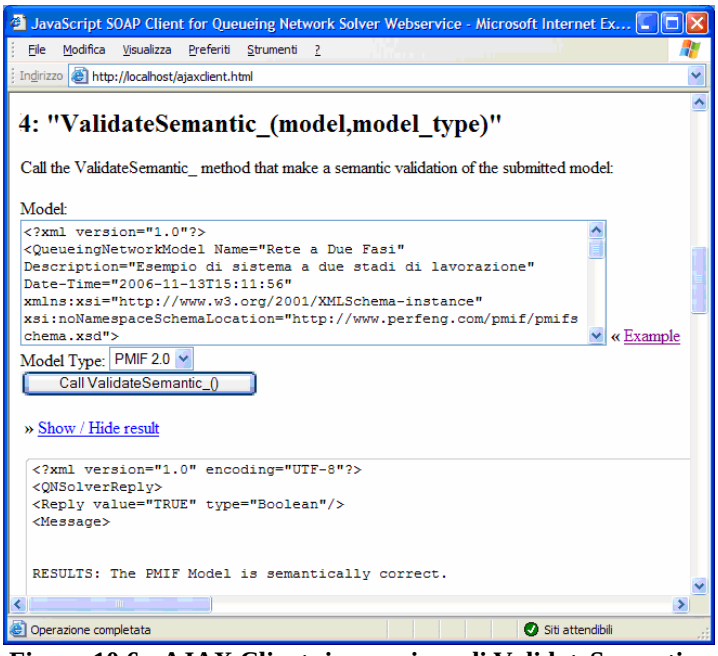


Figura 10.6 - AJAX Client: invocazione di ValidateSemantic_

5: “GetModelDescription_(model, model_type)”:
Questa sezione richiede che vengano specificati un modello ed il tipo (obbligatoriamente PMIF 2.0), e cliccando sul tasto “Call…” viene invocato il metodo GetModelDescription_ che come sappiamo visualizza una descrizione testuale in lingua inglese della rete di code modellata dal file PMIF 2.0.

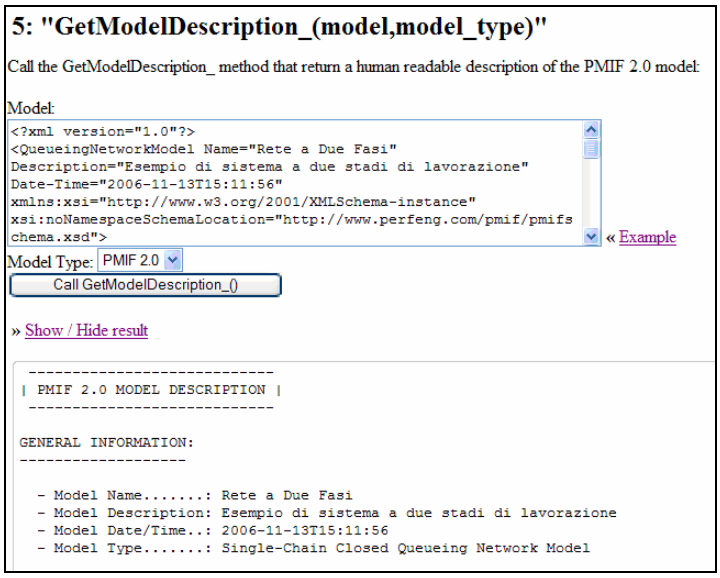


Figura 10.7 - AJAX Client: invocazione di GetModelDescription_

6: “Transform_(model, model_type, tool, method, params)”:
Questa sezione mostra l’uso del metodo Transform_ del webservice. Oltre al modello ed al tipo, prevede la possibilità di specificare il tool di destinazione, il metodo di risoluzione da applicare ed eventuali parametri aggiuntivi richiesti dal tool. Per aiutare l’utente, quando si seleziona un tool il contenuto di Method e Parameters cambia in accordo con la selezione effettuata:

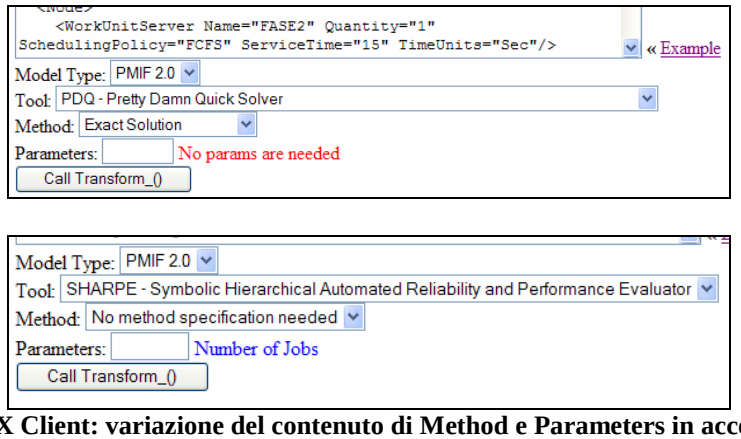


Figura 10.8 - AJAX Client: variazione del contenuto di Method e Parameters in accordo alla selezione

Cliccando sul solito tasto “Call …” viene invocato il metodo Transform_ del web service, e nella division sottostante viene visualizzato il sorgente nel linguaggio del tool selezionato.
Trattandosi di una pagina web, per il salvataggio del modello si deve selezionare il testo contenuto nel textbox di output, ed incollarlo in un text editor esterno per poi salvarlo nel filesystem locale.

7: “Solve_(model, model_type, tool, method, params)”:
Questa è l’ultima sezione presente nel client AJAX e consente di risolvere un modello a rete di code specificando gli stessi parametri del metodo precedente.

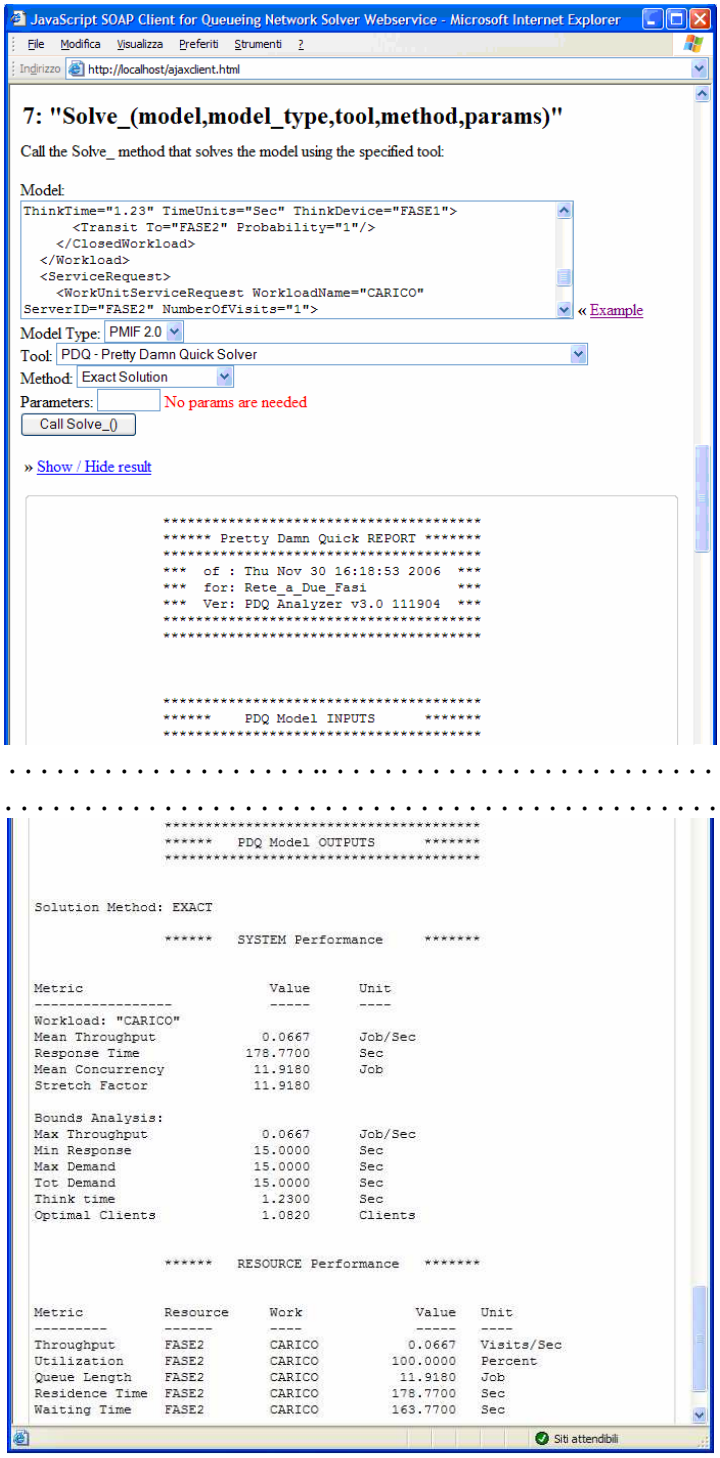


Figura 10.9 - AJAX Client: invocazione di Solve_

Come per il metodo *Transform_*, anche in questo caso, per salvare il risultato ottenuto si deve selezionare il testo presente nella textbox di output, ed incollarlo in un editor di testo esterno per procedere poi al suo salvataggio sul file system locale.

11 Applicazione Client per Windows “PMIF Editor”

Questo è sicuramente il più complesso dei tre client. Si tratta di un’applicazione completa in grado di assistere l’utente dalla fase di design grafico della rete di code fino alla sua risoluzione mediante il web service. Dato che lo sviluppo di un client così complesso non rientrava negli scopi originari di questa tesi daremo qui solo una breve descrizione dalla sua struttura e delle sue funzionalità, mentre, data la complessità della sua implementazione, non sarà possibile scendere ulteriormente nei dettagli implementativi.
L’idea, alla base di questa applicazione, era quella di creare un editor visuale per modelli a rete di code che permettesse di definire gli oggetti tipici di un modello PMIF (*Source Node*, *Server*, *Work Unit Server*, *Sink Node*, *Arc*), con la possibilità di definire classi di clienti aperte e chiuse (*Closed Workload*, *Open Workload*) e le relative richieste di servizio (*Work Unit Service Request*, *Demand Service Request* e *Time Service Request*); il tutto finalizzato alla conversione del modello grafico creato in un modello PMIF 2.0, e al suo invio al web service per la conversione, o risoluzione, con uno dei tool integrati.
L’ambiente di sviluppo scelto per questo scopo è stato il Borland® Delphi™ 2005 for Microsoft® Windows™ Architect Edition, un ambiente RAD (Rapid Application Development) basato sull’Object Pascal della Borland, chiamato Delphi, che consente di sviluppare agevolmente applicazioni, anche complesse, mediante l’uso di componenti predefinite (FORM, BUTTON, CANVAS, IMAGE, ecc), e consentendone anche la creazione di nuovi a partire da quelli predefiniti.

Per lo sviluppo di questa applicazione si sono dovuti affrontare numerosi problemi, primo tra tutti quello relativo al formato di memorizzazione del modello grafico. Dato che il nostro scopo era quello di avere un modello modificabile anche dopo la sua creazione, la scelta del formato di salvataggio è ricaduta su di una versione modificata di PMIF 2.0, che con un po’ di presunzione abbiamo chiamato PMIF 3.0. Le aggiunte, rispetto a PMIF 2.0, riguardano i TAG di definizione dei nodi e degli archi, mentre la definizione dei workload e delle service request sono rimaste invariate.

Esempio:

```
<?xml version="1.0"?>
<QueueingNetworkModel Name="ATM Orig PMIF Example" Description="ATM Orig PMIF 2.0 Example  Model" ... PMIF-Version="PMIF 3.0" >
  <Node>
    <SourceNode Name="SOURCENODE" X="113" Y="230"/>
    <Server Name="CPU" Quantity="1" SchedulingPolicy="PS" X="386" Y="230"/>
    <SinkNode Name="SINKNODE" X="698" Y="230"/>
    <WorkUnitServer Name="ATM" Quantity="1" SchedulingPolicy="IS" ServiceTime="1" TimeUnits="Sec" X="386" Y="332"/>
    <WorkUnitServer Name="DISKS" Quantity="1" SchedulingPolicy="FCFS" ServiceTime="0.05" TimeUnits="Sec" X="384" Y="146"/>
  </Node>
  <Arc FromNode="SOURCENODE" ToNode="CPU">
    <Path Segments="2">
      <Point SegID="1" X="176" Y="230"/>
      <Point SegID="2" X="386" Y="230"/>
    </Path>
  </Arc>
  ... ... ...
  <Arc FromNode="CPU" ToNode="DISKS">
    <Path Segments="6">
      <Point SegID="1" X="478" Y="230"/>
      <Point SegID="2" X="526" Y="230"/>
      <Point SegID="3" X="521" Y="189"/>
      <Point SegID="4" X="350" Y="189"/>
      <Point SegID="5" X="350" Y="145"/>
      <Point SegID="6" X="384" Y="146"/>
    </Path>
  </Arc>
  <Workload>
    ... ... ...
  </Workload>
  <ServiceRequest>
    ... ... ...
  </ServiceRequest>
</QueueingNetworkModel>
```

Nell’esempio sono state evidenziate in rosso le aggiunte rispetto ad un file PMIF 2.0. Per i nodi sono stati aggiunti due attributi, *X* ed *Y*, che indicano la posizione del nodo in un sistema di riferimento relativo, dove lo zero è in alto a sinistra (Figura 11.1).

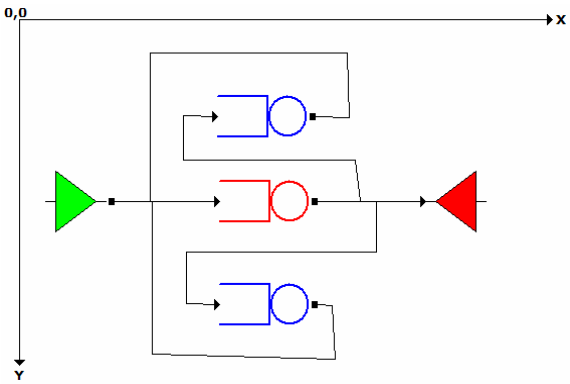


Figura 11.1 - Sistema di riferimento usato in PMIF Editor

Per gli archi si è aggiunto un nuovo TAG contenuto in *Arc*, chiamato *Path*, con un unico attributo *Segments* che contiene il numero totale di punti che costituiscono il path.
All'interno di *Path* ci sono i TAG *Point* che definiscono il cammino dell'arco, e il loro numero deve coincidere con quello definito in *Segments*.
Il TAG *Point* dispone di tre attributi: *SegID* definisce l'ordine in cui devono essere disegnati i segmenti della spezzata che costituisce il cammino o *path*; *X* e *Y* sono invece le coordinate del punto sullo stesso sistema di riferimento definito sopra.

Tutta l'applicazione ruota attorno al componente visuale *TPMIFModel*, da noi sviluppato, per la gestione grafica di un modello a rete di code. Questo componente è stato realizzato estendendo la classe predefinita *TImage* di Delphi (Figura 11.2), con i metodi e gli attributi necessari alla gestione di un modello a rete di code in formato PMIF 2.0.

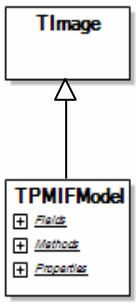


Figura 11.2 - *TPMIFModel* estende e specializza la classe *TImage* di Delphi

Nel gergo di Delphi con il termine *Package* si indica un "contenitore" di componenti visuali e non visuali. Un *Package* Delphi può contenere uno o più *Unit* (un file con estensione .pas), che a loro volta possono contenere la definizione di una o più Classi.

Il concetto di *Package* usato da Delphi non va confuso con il concetto di *Package* usato in UML; infatti Delphi fa coincidere il concetto UML di *Package* con una *Unit*, mentre in UML un *Package* è una collezione di classi che logicamente fanno parte della stessa entità.

Il *Package* Delphi che contiene il nostro componente *TPMIFModel* si chiama *PMIFPackage.bpl*, ed il diagramma delle classi UML, generato da Delphi, è riportato in Figura 11.3.

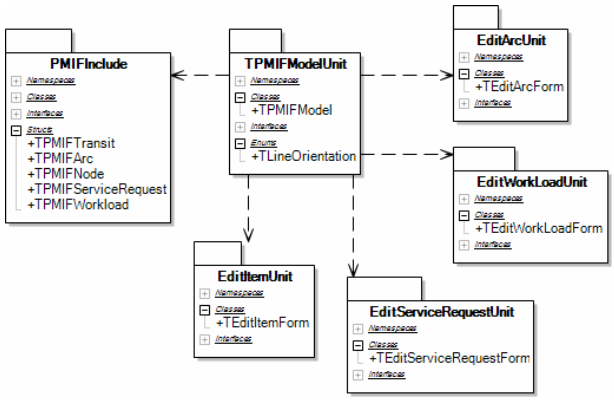


Figura 11.3 - Diagramma UML delle Unit contenute in PMIFPackage.bpl

Come si vede, nella Figura 11.3, la classe principale del componente *TPMIFModel* è contenuta in *TPMIFModelUnit*. Di seguito, descriviamo brevemente il significato delle altre Unit che fanno parte di questo componente:

- *PMIFInclude*: contiene la definizione delle strutture *TPMIFTransit*, *TPMIFArc*, *TPMIFNode*, *TPMIFServiceRequest* e *TPMIFWorkload*, più alcune costanti di uso generale come ad esempio le scheduling policy e le time unit supportate da PMIF;
- *EditItemUnit*: contiene un form (Figura 11.8) per l'editing delle proprietà di un nodo PMIF;
- *EditArcUnit*: contiene un form (Figura 11.9) per l'editing delle proprietà di un arco PMIF;
- *EditWorkLoadUnit*: contiene un form (Figura 11.11) per l'editing (inserimento o modifica) di un workload PMIF;
- *EditServiceRequestUnit*: contiene un form (Figura 11.12) per l'editing (inserimento o modifica) di service request PMIF.

In questa sede non è possibile estendere oltre la descrizione di questo componente. Per chi fosse interessato alla sua implementazione rimandiamo al sorgente contenuto nel CD allegato.

Vediamo ora com'è strutturata l'applicazione principale PMIF Editor (Figura 11.4):

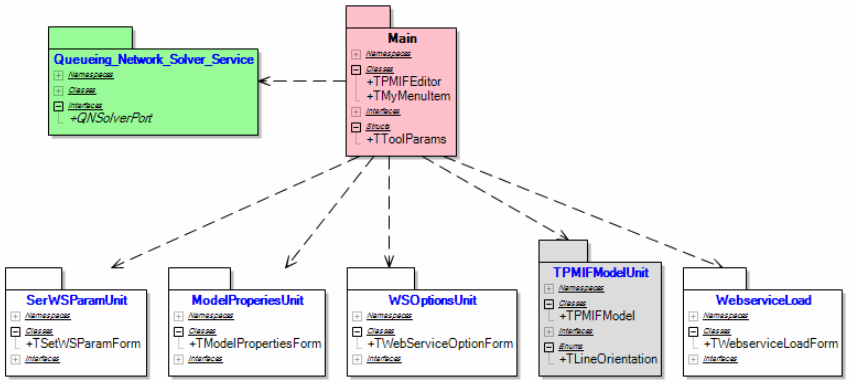


Figura 11.4 - Diagramma UML delle Unit che compongono l'applicazione PMIF Editor

- *Main*: contiene la definizione della classe principale dell'applicazione *TPMIFEditor*, la classe *TMyMenuItem* che discende dalla classe predefinita Delphi *TMenuItem*, ed una struttura *TToolParams* usata da *TMyMenuItem* per la gestione dinamica del contenuto del menu "Model" dell'applicazione;

- *TPMIFModelUnit*: è il componente principale dell'applicazione che abbiamo già descritto sopra (Figura 11.3);
- *WebserviceLoad*: contiene un form (Figura 11.5) visualizzato durante la fase di collegamento al web service. È in questa fase che l'applicazione invoca il metodo "GetToolsList_" per capire quali tool sono integrati nel web service remoto, e costruire di conseguenza il menu "Model" con le voci opportune;
- *SetWSParamUnit*: contiene un form (Figura 11.7) visualizzato, se necessario, durante la fase di conversione, o di risoluzione, di un modello PMIF nel caso il web service necessiti di informazioni aggiuntive (come ad esempio il metodo di risoluzione o il numero di job) non presenti nel modello;
- *ModelPropertiesUnit*: contiene un form (Figura 11.10) visualizzato quando si vogliono modificare alcune informazioni relative al modello (nome e descrizione);
- *WSOptionsUnit*: contiene un form (Figura 11.6) che consente di cambiare l'URL del web service;
- *Queueing_Network_Solver_Service*: è la classe wrapper generata automaticamente da Delphi a partire dal file WSDL di descrizione del web service. È questa la classe che si occupa di tradurre le invocazioni locali, dei metodi del web service, in richieste SOAP da inviare al web service remoto e di recuperare le risposte dello stesso.

Come per il componente *TPMIFModel* non è possibile scendere ad un dettaglio maggiore nella descrizione dell'applicazione, e si rimanda quindi ai sorgenti presenti nel CD allegato.

11.1 Avvio dell'applicazione

L'applicazione, una volta installata, si può eseguire con un doppio click sull'icona 

posta sul desktop.

In fase di avvio viene visualizzata la seguente schermata:

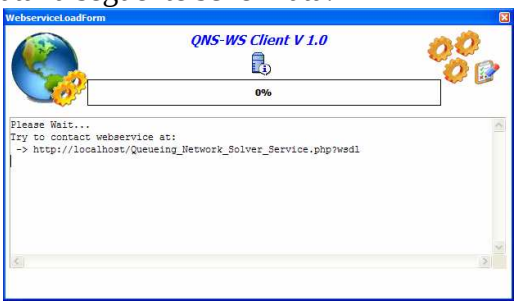


Figura 11.5 - Form visualizzato durante la fase di collegamento al web service

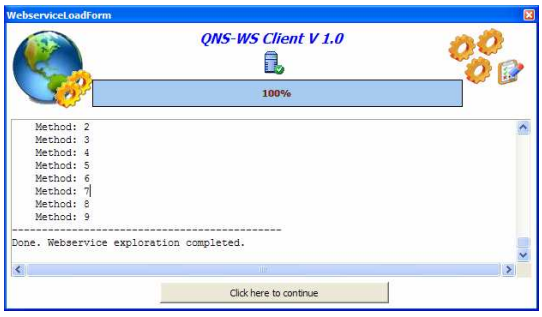
che mostra una progress bar ed una text area, le quali evidenziano il procedere della fase di connessione al web service.
In questa fase il client tenta di contattare il web service all'ultimo URL specificato e di invocare il metodo Copyright_ e GetToolsList_. Il primo viene invocato per testare che il web service sia effettivamente in funzione all'indirizzo specificato, ed il secondo per determinare quali tool sono disponibili ed impostare i menu di invocazione degli stessi in modo corretto.

Un tipico esempio di messaggio di log che appare nella text area in caso di successo è il seguente:

```
Please Wait...
Try to contact web service at:
 -> http://localhost/Queueing_Network_Solver_Service.php?wsdl
-------------------------------------------
Queueing Network Solver Web service (QNS-WS)
WS Version : 1.0
Guest OS   : WINNT
PHP Version: 5.1.1
Location   : University of L'Aquila - Science Faculty
Contact    : Samuel Zallocco <samuel.zallocco@univaq.it>
Created by Samuel Zallocco.
(CC) Some right reserved.
-------------------------------------------
Web service seems to be active...:
Retrieving tools...
Web service integrate 7 tools
-------------------------------------------
Tool ID: PDQ
  Version: 3.0
  Description: Pretty Damn Quick Solver
  Author: Prof. Neil J. Gunther
  Option: SAVE
    Method: EXACT
    Method: APPROX
    Method: CANON
  Option: SOLVE
    Method: EXACT
    Method: APPROX
    Method: CANON
  Option: SOLVE AS IS
...
-------------------------------------------
Done. Web service exploration completed.
```

Come si vede vengono identificati tutti i tool integrati, i metodi di risoluzione di cui dispongono ed i parametri di cui necessitano. In base a queste informazioni verrà costruito il menu "Model".

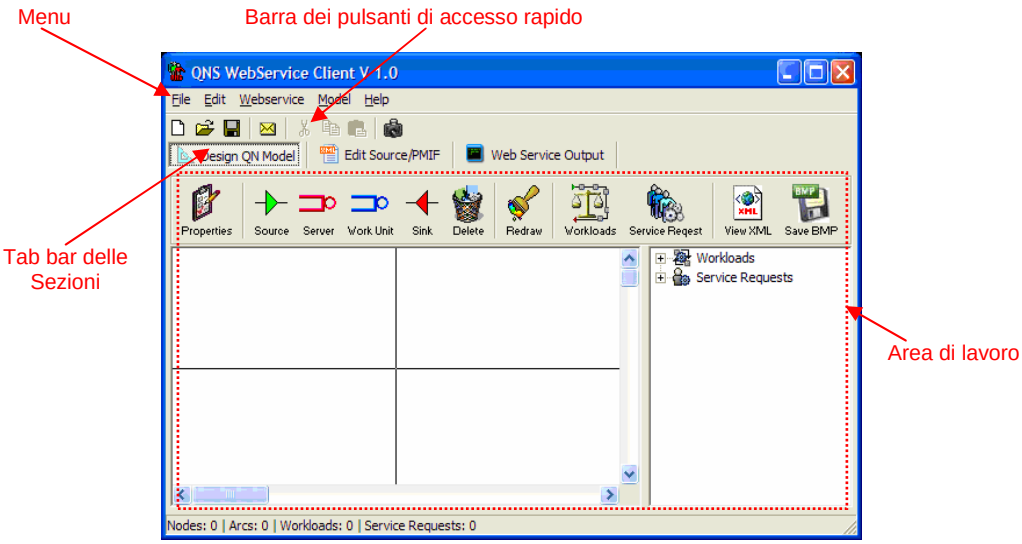
Quando la fase di caricamento dei tool è completata cliccando sul tasto “Click here to continue” si entra nell’applicazione vera e propria.



Nel caso il webservice non sia attivo, nel text box apparirà invece un messaggio del tipo:

```
Please Wait...
Try to contact web service at:
 -> http://localhost/Queueing_Network_Solver_Service.php?wsdl
Error connecting to web service at:
 -> http://localhost/Queueing_Network_Solver_Service.php?wsdl
Please verify web service URL and retry.
Sorry!! The web service return an empty tools list!!
-------------------------------------------
Done. Web service exploration completed.
```

Una volta avviata, l’applicazione si presenta come nella figura seguente:

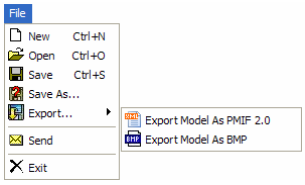


Come si può notare, si tratta di una classica applicazione WIN32, dotata di menu, barra dei pulsanti di selezione rapida per le funzioni di uso comune e di un’area di lavoro. Quest’ultima in realtà si compone di tre sezioni accessibili mediante una tab bar: 

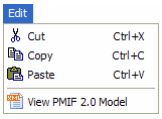
.
Le tre sezioni corrispondono alle diverse fasi di progettazione di una rete di code, la prima (Design QN Model) consente di progettare in modo grafico una rete di code, la seconda (Edit Source/PMIF) consente di vedere il sorgente PMIF 2.0 della rete ed eventualmente di modificarlo, l’ultima sezione (Web Service Output) consente di dialogare con il webservice e di vedere il risultato della risoluzione del modello PMIF.

11.2 Struttura e composizione dei menu dell'applicazione

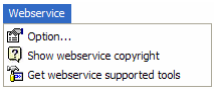
Vediamo le voci e le funzioni di tutti i menu di cui dispone l'applicazione:



- **New**: Azzera il contenuto dell'area di lavoro selezionata. Consentendo di creare un nuovo modello grafico (area di design) o un nuovo sorgente PMIF 2.0 (area di edit), se ci si trova nella sezione Web Service allora viene cancellato il contenuto della text area con le risposte del web service;
- **Open**: Consente di aprire un modello grafico (area di design) o un sorgente PMIF 2.0 (area di edit);
- **Save**: Consente di salvare un modello (grafico o testuale) o le risposte del web service;
- **Save As…**: Consente di salvare un modello (grafico o testuale) e di rinominarlo;
- **Export…**: Consente di salvare un modello grafico in formato PMIF 2.0 o come immagine BMP;
- **Send**: Consente di inviare il modello testuale per posta elettronica;
- **Exit**: Chiude l'applicazione.



- **Cut**: Consente di ritagliare del testo e di salvarlo nella clipboard di windows;
- **Copy**: Copia il testo selezionato e lo posiziona nella clipboard di windows;
- **Paste**: Incolla il testo contenuto nella clipboard di windows nel punto in cui si trova il cursore;
- **View PMIF 2.0 Model**: se ci si trova nella sezione di design visuale, allora il modello viene convertito in PMIF 2.0 e si passa all'area di editing.



- **Option**: Consente di impostare l'URL del web service. Viene anche mantenuto uno storico degli URL inseriti in modo da non doverli rieditare in seguito;

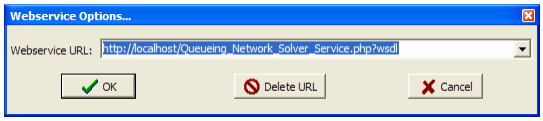
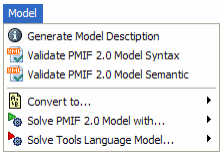


Figura 11.6 - Form per l'impostazione del URL del web service

- **Show webservice copyright**: Viene invocato il metodo Copyright_ del Web Service e viene visualizzata la risposta nella sezione di Output;
- **Get webservice supported tools**: Viene invocato il metodo GetToolsList_ del Web Service e ne viene visualizzata la risposta nella sezione di Output.

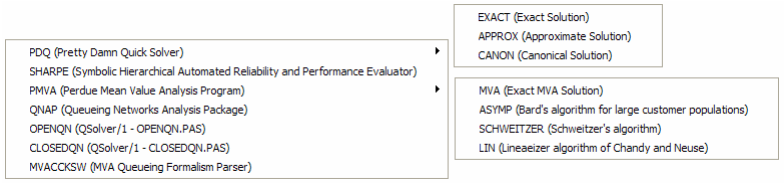


- **Generate Model Description**: Consente di invocare il metodo GetModelDescription_ del Web Service, passando il modello PMIF 2.0 che attualmente si trova nella finestra di editing;
- **Validate PMIF 2.0 Model Syntax**: Consente di invocare il metodo ValidateSyntax_ del Web Service, fornendo in input il modello presente nella finestra di editing;

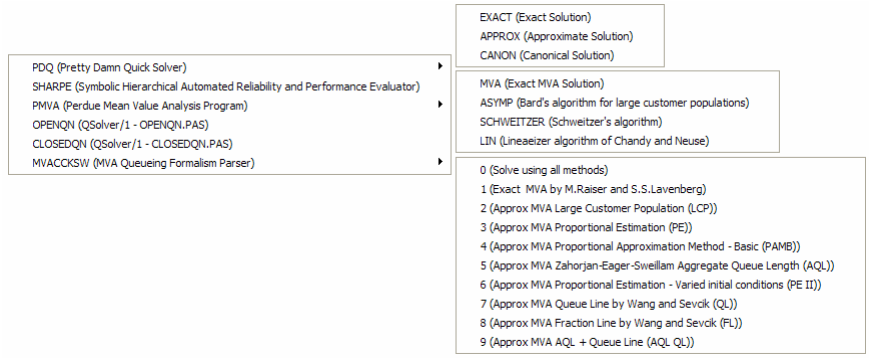
- **Validate PMIF 2.0 Model Semantic**: Consente di invocare il metodo ValidateSemantic_ del Web Service, fornendo in input il modello presente nella finestra di editing;
- **Convert to…**: Consente di invocare il metodo Transform_ del Web Service. Il contenuto del sottomenu associato dipende dai tool supportati dal Web Service remoto e viene costruito in fase di connessione in base alla risposta del metodo GetToolsList_. Al metodo viene passato il file PMIF 2.0 presente nella finestra di editing;
- **Solve PMIF 2.0 Model with…**: Consente di invocare il metodo Solve_ del Web Service. Il contenuto del sottomenu associato dipende dai tool supportati dal Web Service remoto e viene costruito in fase di connessione in base alla risposta del metodo GetToolsList_. Al metodo viene passato il file PMIF 2.0 presente nella finestra di editing;
- **Solve Tools Language Model…**: Consente di invocare il metodo Solve_ del Web Service. Il contenuto del sottomenu associato dipende dai tool supportati dal Web Service remoto e viene costruito in fase di connessione in base alla risposta del metodo GetToolsList_. Al metodo viene passato il file presente nella finestra di editing. A differenza dell'opzione precedente il sorgente che viene passato al metodo deve essere scritto nel linguaggio del tool.

Vediamo un esempio di come potrebbero presentarsi i sottomenu relativi a queste ultime tre voci:

Esempio di sottomenu per "Convert to…":



Esempio di sottomenu per "Solve PMIF 2.0 Model with…" e "Solve Tools Language Model…":



Quando si seleziona dal menu di conversione, o di risoluzione, un tool che necessita di parametri aggiuntivi, si vedrà apparire una finestra di dialogo (Figura 11.7) con la descrizione del parametro richiesto ed un campo per la sua immissione.

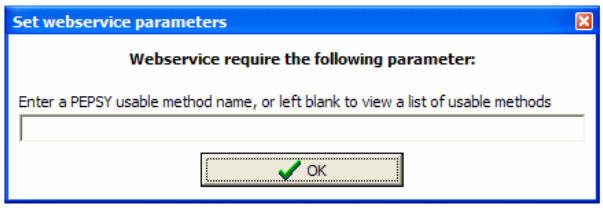
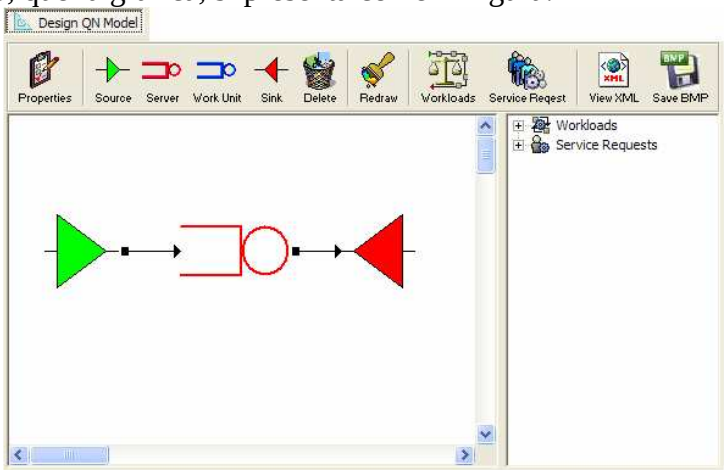


Figura 11.7 - Form di richiesta parametri aggiuntivi

11.3 Design grafico della rete di code

La prima area di lavoro, quella grafica, si presenta come in figura:



Come si vede ci sono tre aree principali: nella parte alta una barra con delle icone, nella parte centrale l'area di disegno della rete di code, e sul lato destro un'area di visualizzazione per le proprietà non grafiche della rete di code. Come si vede le icone sono suddivise per funzionalità. Il loro significato è il seguente:

Consente di specificare il nome e la descrizione del modello.

Consente di definire il nodo di ingresso (Source Node) di una rete di code aperta.

Consente di definire un nodo di tipo Server.

Consente di definire un nodo di tipo WorkUnit Server.

Consente di definire il nodo di uscita (Sink Node) di una rete di code aperta.

Consente di eliminare un nodo qualsiasi della rete di code.

Consente di effettuare un redraw della rete. È utile in fase di definizione degli archi, nel caso si lascino archi incompleti che non vengono correttamente visualizzati.

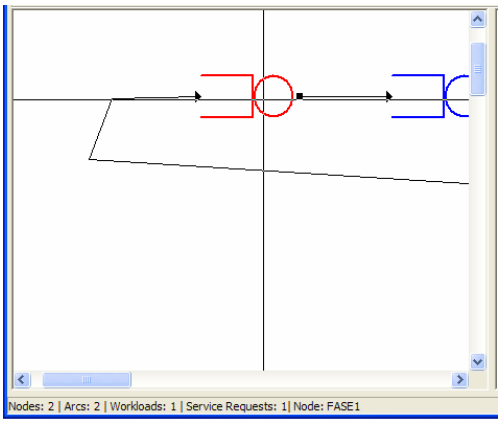
Consente di definire un nuovo workload.

Consente di definire una nuova service request.

Converte la rete di code in formato XML/PMIF e passa automaticamente all'area di editing del sorgente.

Consente di esportare una "foto" della rete di code in formato grafico BMP.

L’area di disegno della rete:



è dotata di due guide, che aiutano a posizionare gli elementi grafici e gli archi di connessione. Tenendo premuto il tasto [CTRL] le guide scompaiono. Per posizionare uno qualsiasi dei quattro tipi di nodi disponibili:

- Source Node,
- Server Node,
- Work Unit Server Node,
- Sink Node,

è sufficiente selezionare il pulsante relativo e fare click con il mouse nel punto esatto in cui si vuole inserirlo. Una volta posizionato, è anche possibile editare le sue caratteristiche facendo doppio click su di esso. Apparirà una finestra di dialogo che in base al tipo di nodo selezionato consentirà di cambiare alcuni parametri del nodo:

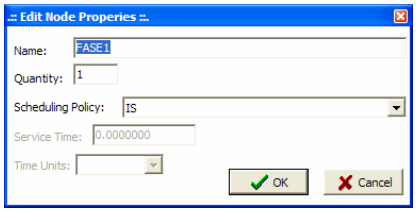


Figura 11.8 - Form per l'editing delle proprietà di un nodo

Una volta piazzati i nodi, è possibile connetterli mediante un arco con l’uso del tasto [CRTL]. Posizionando il mouse sopra un nodo origine e facendo click tenendo premuto il tasto [CTRL] si vedrà apparire un arco; a questo punto, sempre tenendo premuto [CTRL], è possibile facendo click con il mouse stabilire il cammino che l’arco deve compiere e quando si farà click sul nodo destinazione allora apparirà l’arco voluto. Se si lascia il tasto [CTRL] prima di raggiungere il nodo destinazione, allora si dovrà effettuare un refresh per eliminare i residui di arco non valido.
La stessa procedura descritta per l’editing delle proprietà dei nodi vale anche per gli archi. Facendo doppio click su di un arco apparirà una finestra che darà la possibilità di modificare la descrizione dell’arco. Fare attenzione al fatto che se si cambia la descrizione di default, poi nel caso si decidesse di cambiare nome ai nodi questa non verrà più aggiornata in modo automatico.

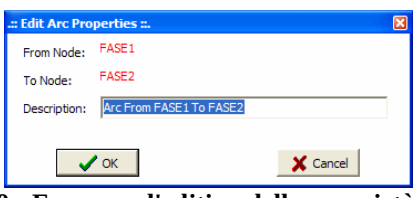


Figura 11.9 - Form per l'editing delle proprietà di un arco

Per cancellare un nodo o un arco, è sufficiente fare click sull'icona Delete, e successivamente fare click sul nodo o sull'arco da cancellare. Nel caso si elimini un nodo, anche tutti gli archi ad esso connessi verranno eliminati.

Per dare un nome e una breve descrizione al modello, è sufficiente premere l'icona "Properties", apparirà la seguente finestra di dialogo:

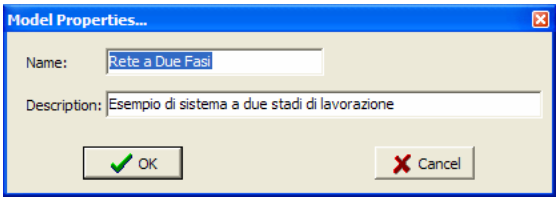
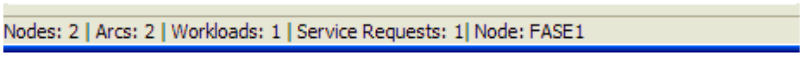


Figura 11.10 - Form per impostare alcune informazione relative al modello PMIF

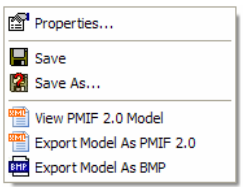
Sulla barra di stato dell'applicazione si può notare come, posizionando il mouse sopra ad un nodo o ad un'arco, vengano visualizzate le informazioni relative al nome:



Il nome del nodo (o la descrizione dell'arco) viene visualizzato anche tramite "*tool tip*", posizionando per qualche secondo il mouse sopra di esso senza cliccare.

La stessa barra di stato contiene dei contatori con il numero di nodi, archi, carichi e richieste di servizio definiti.

Cliccando con il tasto destro del mouse in un punto qualsiasi dell'area di disegno appare il seguente menu:



che consente di accedere velocemente ad alcune funzioni di uso comune, come proprietà del modello, salvataggio, esportazione in PMIF o BMP.

Una volta definita la rete di code con tutti i nodi ed archi necessari, si può procedere alla definizione dei Workload e delle Service Request cliccando sulle icone relative. L'inserimento di Workloads avviene mediante la seguente finestra:

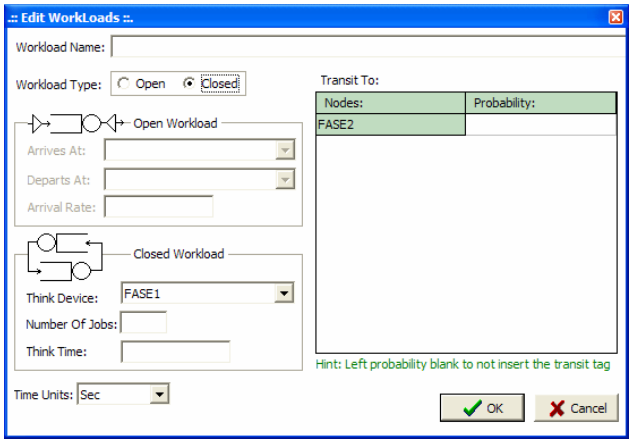
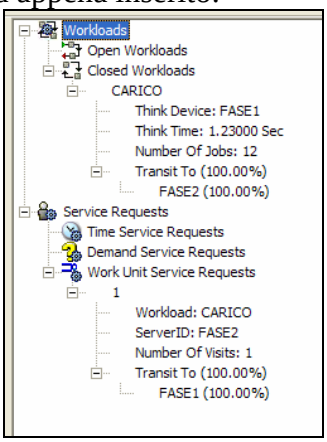
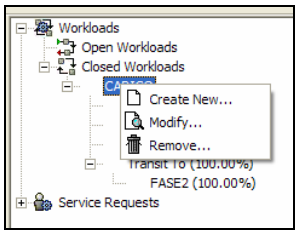


Figura 11.11 - Form per l'editing di Workloads

Come si vede dalla figura, in alto è necessario dare un nome al workload che si sta definendo, poi si può scegliere il tipo di workload: Aperto o Chiuso. In base alla selezione si attiverà o si disattiverà la parte di inserimento relativa ai closed workload o open workload. La scelta dei nodi avviene mediante combo box a discesa. Non è possibile scegliere un nodo qualsiasi ma solo quelli che rispettano determinati vincoli, così come definiti nel documento [48]. Sul lato destro è possibile definire le probabilità di transito (in percentuale mediante il simbolo % o come numero tra 0 e 1) ed in fondo a sinistra l'unità di misura per il Think Time o Arrival Rate. Confermando l'inserimento si vedrà apparire nell'area di lavoro il workload appena inserito:



Per modificare o eliminare un workload, è sufficiente selezionare il nome del workload e premere il tasto destro del mouse. Apparirà un menu contestuale:



con le opzioni di modifica e cancellazione.
La procedura di inserimento delle service request è pressoché identica, ma cambia la finestra di dialogo:

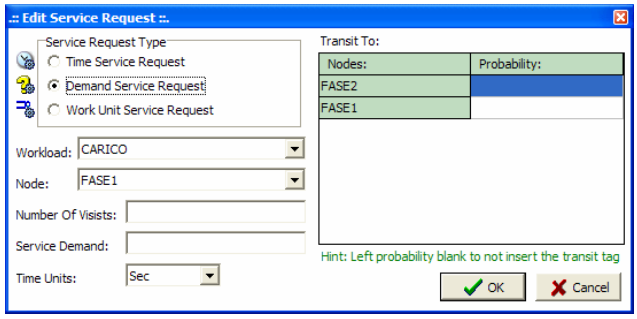
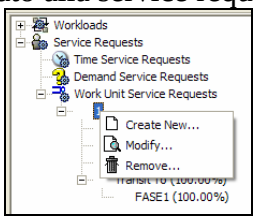


Figura 11.12 - Form per l'editing di una ServiceRequest

La modifica o cancellazione di una service request avviene sempre mediante menu contestuale cliccando sul numero progressivo assegnato alla service request:



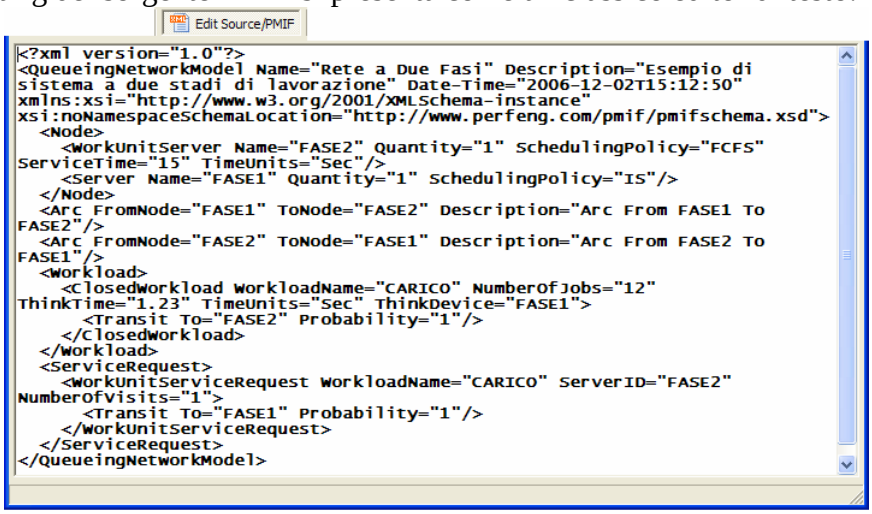
Completata la fase di design grafico della rete di code, mediante l'icona 

è possibile vedere il sorgente PMIF 2.0 e passare automaticamente alla sezione di editing.

Ovviamente il modello si può salvare e ricaricare successivamente mediante le apposite opzioni del menu "File".

11.4 Editing del sorgente XML/PMIF 2.0

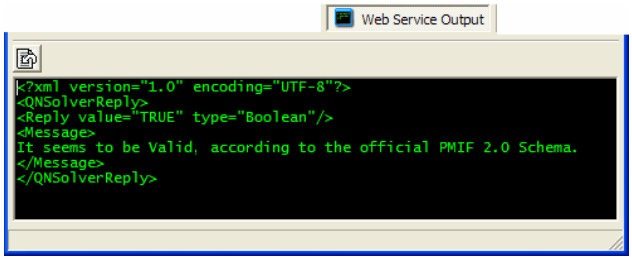
La sezione di editing del sorgente PMIF si presenta come un classico editor di testo:



È possibile scrivere, modificare, cancellare, copiare, tagliare e incollare il testo come si farebbe con Notepad.
Questa sezione è del tutto indipendente dalla precedente, nel senso che se si dispone di un file PMIF 2.0 si può caricarlo mediante il menu FILE, editarlo, salvarlo e risolverlo tramite il webservice, senza il bisogno di passare attraverso al sezione di editing visuale.

11.5 Invocazione del Web Service

Quando la rete di code è stata definita e trasformata in XML/PMIF 2.0 è possibile passare alla sezione di invocazione del web service. Questa sezione, è composta da una text box non editabile che contiene le risposte del web service:



L'unico tasto presente () serve per copiare il contenuto della text box di output nell'editor. Questa funzione può essere utile per modificare il sorgente convertito di uno dei tool per una successiva risoluzione mediante "Solve Tools Language Model…".

Selezionando una delle voci dei menu Webservice e Model si vedrà apparire il risultato dell'invocazione del web method relativo nella finestra di output. Come già visto in precedenza, gli output sono tutti in XML tranne i risultati della risoluzione dei modelli e della conversione da PMIF 2.0 a linguaggio del tool che sono invece in formato testuale.

Per salvare il risultato della conversione o della risoluzione è sufficiente utilizzare le funzioni "Save" o "Save As…" del menu "File".

12 Risultati sperimentali

Vediamo alcuni modelli di reti di code in PMIF 2.0, e come il nostro webservice traduce questi modelli nel linguaggio sorgente dei tool. Ove possibile metteremo anche a confronto gli indici di performance ottenuti con i vari tool.

12.1 Rete di code aperta con 3 centri di servizio e 2 classi di clienti

Il modello che andremo ad analizzare è lo stesso usato in [14]. Una caratteristica di questo modello è di presentare risorse sature e di risultare quindi instabile.

Rappresentazione grafica del modello:

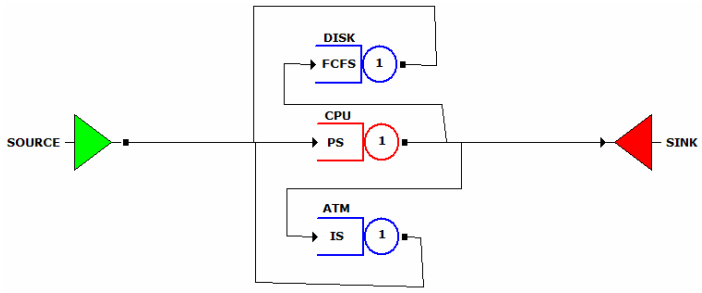


Figura 12.1 - ATM Original Example

Modello PMIF 2.0:

```
<?xml version="1.0"?>
<QueueingNetworkModel Name="ATM Orig PMIF Example" Description="ATM Orig PMIF 2.0 Example Model" Date-Time="2007-03-14T10:03:43"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://www.perfeng.com/pmif/pmifschema.xsd">
  <Node>
    <SourceNode Name="SOURCENODE"/>
    <Server Name="CPU" Quantity="1" SchedulingPolicy="PS"/>
    <SinkNode Name="SINKNODE"/>
    <WorkUnitServer Name="ATM" Quantity="1" SchedulingPolicy="IS" ServiceTime="1" TimeUnits="Sec"/>
    <WorkUnitServer Name="DISKS" Quantity="1" SchedulingPolicy="FCFS" ServiceTime="0.05" TimeUnits="Sec"/>
  </Node>
  <Arc FromNode="SOURCENODE" ToNode="CPU" Description="Arc From SOURCENODE To CPU"/>
  <Arc FromNode="CPU" ToNode="SINKNODE" Description="Arc From CPU To SINKNODE"/>
  <Arc FromNode="ATM" ToNode="CPU" Description="Arc From ATM To CPU"/>
  <Arc FromNode="CPU" ToNode="ATM" Description="Arc From CPU To ATM"/>
  <Arc FromNode="DISKS" ToNode="CPU" Description="Arc From DISKS To CPU"/>
  <Arc FromNode="CPU" ToNode="DISKS" Description="Arc From CPU To DISKS"/>
  <Workload>
    <OpenWorkload WorkloadName="WITHDRAWAL" ArrivalRate="1" TimeUnits="Sec" ArrivesAt="SOURCENODE" DepartsAt="SINKNODE">
      <Transit To="CPU" Probability="1"/>
    </OpenWorkload>
    <OpenWorkload WorkloadName="GET_BALANCE" ArrivalRate="1" TimeUnits="Sec" ArrivesAt="SOURCENODE" DepartsAt="SINKNODE">
      <Transit To="CPU" Probability="1"/>
    </OpenWorkload>
  </Workload>
  <ServiceRequest>
    <DemandServiceRequest WorkloadName="WITHDRAWAL" ServerID="CPU" NumberOfVisits="20" ServiceDemand="0.0063" TimeUnits="Sec">
      <Transit To="SINKNODE" Probability="0.05"/>
      <Transit To="ATM" Probability="0.55"/>
      <Transit To="DISKS" Probability="0.4"/>
    </DemandServiceRequest>
    <WorkUnitServiceRequest WorkloadName="WITHDRAWAL" ServerID="ATM" NumberOfVisits="11">
      <Transit To="CPU" Probability="1"/>
    </WorkUnitServiceRequest>
    <WorkUnitServiceRequest WorkloadName="WITHDRAWAL" ServerID="DISKS" NumberOfVisits="8">
      <Transit To="CPU" Probability="1"/>
    </WorkUnitServiceRequest>
    <DemandServiceRequest WorkloadName="GET_BALANCE" ServerID="CPU" NumberOfVisits="10" ServiceDemand="0.0025" TimeUnits="Sec">
      <Transit To="SINKNODE" Probability="0.1"/>
      <Transit To="ATM" Probability="0.6"/>
      <Transit To="DISKS" Probability="0.3"/>
    </DemandServiceRequest>
    <WorkUnitServiceRequest WorkloadName="GET_BALANCE" ServerID="ATM" NumberOfVisits="6">
      <Transit To="CPU" Probability="1"/>
    </WorkUnitServiceRequest>
    <WorkUnitServiceRequest WorkloadName="GET_BALANCE" ServerID="DISKS" NumberOfVisits="3">
      <Transit To="CPU" Probability="1"/>
    </WorkUnitServiceRequest>
  </ServiceRequest>
</QueueingNetworkModel>
```

12.1.1 Traduzioni possibili per modelli aperti:

- PDQ (Anche risoluzione)
- QNAP (Solo traduzione)
- OPENQN (Anche risoluzione)
- MQNA1 (Anche risoluzione)
- MQNA2 (Solo traduzione)
- PEPSY (Anche risoluzione)

Modello PDQ:

```
// This is a PDQ Model:
//  - Name: ATM_Orig_PMIF_Example AKA ATM Orig PMIF Example
//  - Description: ATM Orig PMIF 2.0 Example  Model
//  - Date-Time: 2007-03-14T10:03:08
// Generated from the original PMIF 2.0 Model using PMIF2_to_PDQ.xsl
// that was part of the Queueing Network Solver Webservice Project
// Copyright (c) by Samuel Zallocco - University of L'Aquila - ITALY - All right reserved
PROMPT OFF  // Do not show the command prompt
MESSAGE OFF // Do not show the PDQ Shell execution messages
ERROR OFF   // Do not report PDQ Shell error messages but only PDQ LIB errors
//
// Model Type: Multiple-Chain Open Queueing Network Model
//
INIT "ATM_Orig_PMIF_Example"
CREATENODE "CPU" CEN PS // Number of Processing Units = 1
VAR $ATM_ServiceTime = 1
CREATENODE "ATM" CEN IS // Number of Processing Units = 1
VAR $DISKS_ServiceTime = 0.05
CREATENODE "DISKS" CEN FCFS // Number of Processing Units = 1
CREATEOPEN "WITHDRAWAL" 1
CREATEOPEN "GET_BALANCE" 1
SETVISITS "ATM" "WITHDRAWAL" 11 $ATM_ServiceTime
SETVISITS "DISKS" "WITHDRAWAL" 8 $DISKS_ServiceTime
SETVISITS "ATM" "GET_BALANCE" 6 $ATM_ServiceTime
SETVISITS "DISKS" "GET_BALANCE" 3 $DISKS_ServiceTime
SETVISITS "CPU" "WITHDRAWAL" 20 0.000315
SETVISITS "CPU" "GET_BALANCE" 10 0.00025
SOLVE CANON
REPORT
EXIT
```

Modello OPENQN:

```
// This is an OPENQN Model:
//  - Name: ATM_Orig_PMIF_Example AKA ATM Orig PMIF Example
//  - Description: ATM Orig PMIF 2.0 Example  Model
//  - Date-Time: 2007-03-14T10:03:08
// Generated from the original PMIF 2.0 Model using PMIF2_to_OPENQN.xsl
// that was part of the Queueing Network Solver Webservice Project
// Copyright (c) by Samuel Zallocco - University of L'Aquila - ITALY - All right reserved
//
// Model Type: Multiple-Chain Open Queueing Network Model
OPENQN "ATM Orig PMIF Example"
         // Class Definition Section
         Classes 2
                  // CLASS {WorkloadName} ArrivalRate
                  Class  {WITHDRAWAL}  1
                  Class  {GET_BALANCE} 1
         // Server Definition Section
         Nodes 3
                  // Server Name Type [Saturation ( service_rates... )]
                  Server             [CPU] LI
                  WorkUnitServer     [ATM] LI
                  WorkUnitServer     [DISKS] LI
         // Service Demand Matrix
         ServiceDemands
                  // Server_Name     Class_Name = Demand          Class_Name = Demand
                  [CPU]              {WITHDRAWAL} = 0.0063        {GET_BALANCE} = 0.0025
                  [ATM]              {WITHDRAWAL} = 11            {GET_BALANCE} = 6
                  [DISKS]            {WITHDRAWAL} = 0.4           {GET_BALANCE} = 0.15
END
```

Modello MQNA1:

```
oqn ATM_Orig_PMIF_Example (2,5)
//_This_is_an_Open_MQNA1_Model
//__-_Name:_ATM_Orig_PMIF_Example
//__-_Description:_ATM_Orig_PMIF_2.0_Example_Model
//__-_Date-Time:_2007-03-14T10:03:08
//_Generated_from_the_original_PMIF_2.0_Model_using_PMIF2_to_MQNA1.xsl
//_that_was_part_of_the_Queueing_Network_Solver_Webservice_Project
//_(CC)_Samuel_Zallocco_-_University_of_L_Aquila_-_ITALY_-_Some_right_reserved
declarations{
  class WITHDRAWAL;
  class GET_BALANCE;
  queue SOURCENODE;
  queue CPU;
  queue ATM;
  queue DISKS;
  queue SINKNODE;
}
network{
  queue SOURCENODE {
    servers = 0;
    capacity = 10000;
    class WITHDRAWAL {
      service time = 0.0;
      arrival rate = 1;
      rot CPU: 1;
    }
    class GET_BALANCE {
      service time = 0.0;
      arrival rate = 1;
      rot CPU: 1;
    }
  }
  queue CPU {
    servers = 1;
    capacity = 10000;
    class WITHDRAWAL {
      service time = 0.000315;
      arrival rate = 0;
      rot SINKNODE: 0.05;
      rot ATM: 0.55;
      rot DISKS: 0.4;
    }
    class GET_BALANCE {
      service time = 0.00025;
      arrival rate = 0;
      rot SINKNODE: 0.1;
      rot ATM: 0.6;
      rot DISKS: 0.3;
    }
  }
  queue ATM {
    servers = 1;
    capacity = 10000;
    class WITHDRAWAL {
      service time = 1;
      arrival rate = 0;
      rot CPU: 1;
    }
    class GET_BALANCE {
      service time = 1;
      arrival rate = 0;
      rot CPU: 1;
    }
  }
  queue DISKS {
    servers = 1;
    capacity = 10000;
    class WITHDRAWAL {
      service time = 0.05;
      arrival rate = 0;
      rot CPU: 1;
    }
    class GET_BALANCE {
      service time = 0.05;
      arrival rate = 0;
      rot CPU: 1;
    }
  }
  queue SINKNODE {
    servers = 0;
    capacity = 10000;
    class WITHDRAWAL {
      service time = 0.0;
      arrival rate = 0;
    }
    class GET_BALANCE {
      service time = 0.0;
      arrival rate = 0;
    }
  }
}
```

Modello MQNA2:

```
oqn ATM_Orig_PMIF_Example (2,5)
//_This_is_an_Open_MQNA2_Model
//__-_Name:_ATM_Orig_PMIF_Example
//__-_Description:_ATM_Orig_PMIF_2.0_Example_Model
//__-_Date-Time:_2007-03-14T10:03:08
//_Generated_from_the_original_PMIF_2.0_Model_using_PMIF2_to_MQNA2.xsl
//_that_was_part_of_the_Queueing_Network_Solver_Webservice_Project
//_(CC)_Samuel_Zallocco_-_University_of_L_Aquila_-_ITALY_-_Some_right_reserved
declarations{
  class WITHDRAWAL;
  class GET_BALANCE;
  queue SOURCENODE;
  queue CPU;
  queue ATM;
  queue DISKS;
  queue SINKNODE;
}
network{
  queue SOURCENODE {
    priority (1,1);
    servers = 0;
    capacity = 10000;
    class WITHDRAWAL {
      service time = 0.0;
      arrival rate = 1;
      routing CPU(loss): 1;
    }
    class GET_BALANCE {
      service time = 0.0;
      arrival rate = 1;
      routing CPU(loss): 1;
    }
  }
  queue CPU {
    priority (1,1,);
    servers = 1;
    capacity = 10000;
    class WITHDRAWAL {
      service time = 0.000315;
      arrival rate = 0;
      routing SINKNODE(loss): 0.05;
      routing ATM(loss): 0.55;
      routing DISKS(loss): 0.4;
    }
    class GET_BALANCE {
      service time = 0.00025;
      arrival rate = 0;
      routing SINKNODE(loss): 0.1;
      routing ATM(loss): 0.6;
      routing DISKS(loss): 0.3;
    }
  }
  queue ATM {
    priority (1,1);
    servers = 1;
    capacity = 10000;
    class WITHDRAWAL {
      service time = 1;
      arrival rate = 0;
      routing CPU(loss): 1;
    }
    class GET_BALANCE {
      service time = 1;
      arrival rate = 0;
      routing CPU(loss): 1;
    }
  }
  queue DISKS {
    priority (1,1);
    servers = 1;
    capacity = 10000;
    class WITHDRAWAL {
      service time = 0.05;
      arrival rate = 0;
      routing CPU(loss): 1;
    }
    class GET_BALANCE {
      service time = 0.05;
      arrival rate = 0;
      routing CPU(loss): 1;
    }
  }
  queue SINKNODE {
    priority (1,1);
    servers = 0;
    capacity = 10000;
    class WITHDRAWAL {
      service time = 0.0;
      arrival rate = 0;
    }
    class GET_BALANCE {
      service time = 0.0;
      arrival rate = 0;
    }
  }
}
```

Modello PEPSY:

```
# @(#)PEPSY:        e_ATM_Orig_PMIF_Example      1.10      2007/03/24 19:03:49
#-------------------------------------------------------------------------------------------
#
#                              P E P S Y
#                     Performance and Evaluation Prediction SYstem
#
#         Copyright 1987, 1988
#                               Universitaet Erlangen-Nuernberg,
#                               Lehrstuhl fuer Betriebssysteme
#                               Martensstrasse 1
#                               91058 Erlangen
#
#         Alle Rechte, einschliesslich Verfilmung und Vertonung, vorbehalten.
#         PEPSY darf ohne Erlaubnis der Verfasser nicht in Lesezirkeln
#         gefuehrt werden.
#
# This is a PEPSY-QNS Model:
#  - Name: ATM_Orig_PMIF_Example AKA ATM Orig PMIF Example
#  - Description: ATM Orig PMIF 2.0 Example  Model
#  - Date-Time: 2007-03-24T19:03:49
#  - Model Type: Multiple-Chain Open Queueing Network Model
#
# Generated from the original PMIF 2.0 Model using PMIF2_to_PEPSY.xsl
# that was part of the Queueing Network Solver Webservice Project
# (CC) Samuel Zallocco - University of L'Aquila - ITALY - Some right reserved
#-------------------------------------------------------------------------------------------
#
# inputfile version  3
#

NUMBER NODES: 3
NUMBER CLASSES: 2

NODE SPECIFICATION

  node   |      name          |     type
---------+--------------------+---------------------
     1   |      CPU           |     M/G/1-PS
     2   |      ATM           |     M/G/0-IS
     3   |      DISKS         |     M/M/1-FCFS


CLASS SPECIFICATION

  class   | arrival rate   number of jobs
----------+---------------------------------
     1    |    1              -
     2    |    1              -


CLASS SPECIFIC PARAMETERS


CLASS 1

     node           |  service_rate    squared_coeff._of_variation
--------------------+---------------------------------------------
     CPU            |  3174.60317460317 1
     ATM            |  1                1
     DISKS          |  20               1


SWITCHING PROBABILITIES

from/to    | outside    CPU        ATM        DISKS
-----------+-------------------------------------------
outside    | 0.000000   1.000000   0.000000   0.000000
CPU        | 0.050000   0.000000   0.550000   0.400000
ATM        | 0.000000   1.000000   0.000000   0.000000
DISKS      | 0.000000   1.000000   0.000000   0.000000


CLASS 2

     node           |  service_rate    squared_coeff._of_variation
--------------------+---------------------------------------------
     CPU            |  4000             1
     ATM            |  1                1
     DISKS          |  20               1


SWITCHING PROBABILITIES

from/to    | outside    CPU        ATM        DISKS
-----------+-------------------------------------------
outside    | 0.000000   1.000000   0.000000   0.000000
CPU        | 0.100000   0.000000   0.600000   0.300000
ATM        | 0.000000   1.000000   0.000000   0.000000
DISKS      | 0.000000   1.000000   0.000000   0.000000

# END MODEL
```

Modello QNAP:

```
& This is a QNAP Model:
&  - Name: ATM Orig PMIF Example
&  - Description: ATM Orig PMIF 2.0 Example  Model
&  - Date-Time: 2007-03-14T10:03:08
& Generated from the original PMIF 2.0 Model using PMIF2_to_QNAP.xsl
& that was part of the Queueing Network Solver Webservice Project
& Original version (c) by Catalina M. Llado (Universitat de les Illes Balears)
& Pretty Layout Modification (c) by Samuel Zallocco (University of L'Aquila Italy)
/DECLARE/   QUEUE CPU;
            QUEUE ATM, DISKS;
            QUEUE SOURCEN1, SOURCEN2;
            CLASS WITHDRAW, GET_BALA;
            REAL TWITHDRA, TGET_BAL;
&
& For Each Server
&
/STATION/   NAME= CPU;
            SCHED = PS;
&
& For Each WorkUnitServer
&
/STATION/   NAME = ATM;
            SERVICE = EXP(1);
            TYPE = INFINITE;
/STATION/   NAME = DISKS;
            SERVICE = EXP(0.05);
            SCHED = FIFO;
&
& For Each OpenWorkload
&
/STATION/   NAME = SOURCEN1;
            TYPE= SOURCE;
            SERVICE= EXP(1);
            TRANSIT= CPU, WITHDRAW;

/STATION/   NAME = SOURCEN2;
            TYPE= SOURCE;
            SERVICE= EXP(1);
            TRANSIT= CPU, GET_BALA;
&
& For Each WorkUnitServiceRequest
&
/STATION/   NAME = ATM;
            TRANSIT(WITHDRAW) = CPU, 1;

/STATION/   NAME = DISKS;
            TRANSIT(WITHDRAW) = CPU, 1;

/STATION/   NAME = ATM;
            TRANSIT(GET_BALA) = CPU, 1;

/STATION/   NAME = DISKS;
            TRANSIT(GET_BALA) = CPU, 1;
&
& For Each DemandServiceRequest
&
/STATION/   NAME = CPU;
            SERVICE(WITHDRAW) = EXP(0.000315);
            TRANSIT(WITHDRAW) = SINKNODE, 0.05, ATM, 0.55, DISKS, 0.4;

/STATION/   NAME = CPU;
            SERVICE(GET_BALA) = EXP(0.00025);
            TRANSIT(GET_BALA) = SINKNODE, 0.1, ATM, 0.6, DISKS, 0.3;
&
& Start Analisys
&
/CONTROL/   CLASS = ALL QUEUE;
/EXEC/ BEGIN
            SOLVE;
            & --- WorkUnitServiceRequest ---
            PRINT ("RESIDENCE TIME FOR WITHDRAW,ATM = " , MCUSTNB(ATM,WITHDRAW) / MTHRUPUT (ATM,WITHDRAW) );
            TWITHDRA := TWITHDRA + MCUSTNB(ATM,WITHDRAW);
            PRINT ("RESIDENCE TIME FOR WITHDRAW,DISKS = " , MCUSTNB(DISKS,WITHDRAW) / MTHRUPUT (DISKS,WITHDRAW) );
            TWITHDRA := TWITHDRA + MCUSTNB(DISKS,WITHDRAW);
            PRINT ("RESIDENCE TIME FOR GET_BALA,ATM = " , MCUSTNB(ATM,GET_BALA) / MTHRUPUT (ATM,GET_BALA) );
            TGET_BAL := TGET_BAL + MCUSTNB(ATM,GET_BALA);
            PRINT ("RESIDENCE TIME FOR GET_BALA,DISKS = " , MCUSTNB(DISKS,GET_BALA) / MTHRUPUT (DISKS,GET_BALA) );
            TGET_BAL := TGET_BAL + MCUSTNB(DISKS,GET_BALA);
            & --- DemandServiceRequest ---
            TWITHDRA := TWITHDRA + MCUSTNB(CPU,WITHDRAW);
            PRINT ("RESIDENCE TIME FOR WITHDRAW,CPU = " , MCUSTNB(CPU,WITHDRAW) / MTHRUPUT (CPU,WITHDRAW) );
            TGET_BAL := TGET_BAL + MCUSTNB(CPU,GET_BALA);
            PRINT ("RESIDENCE TIME FOR GET_BALA,CPU = " , MCUSTNB(CPU,GET_BALA) / MTHRUPUT (CPU,GET_BALA) );
            & --- OpenWorkload ---
            TWITHDRA := TWITHDRA / MTHRUPUT(SOURCEN1);
            PRINT ("RESPONSE TIME FOR WITHDRAW = ", TWITHDRA);
            TGET_BAL := TGET_BAL / MTHRUPUT(SOURCEN2);
            PRINT ("RESPONSE TIME FOR GET_BALA = ", TGET_BAL);
            & --- ClosedWorkload ---
      END;
/END/
```

12.1.2 Output della risoluzione:

Gli unici tool utilizzabili per la risoluzione del modello aperto sono PDQ, OPENQN, PEPSY e MQNA1. I risultati per QNAP sono stati ricavati da [14].

PDQ:

```
ERROR in model:"ATM_Orig_PMIF_Example" at PDQ_Canonical:
Arrival rate 1.000 exceeds system saturation 0.091 = 1/11.000
```

OPENQN:

```
The system is not stable.
```

PEPSY:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNMct8nQW  with method:  priomva2m

PERFORMANCE_INDICES FOR NET:  pmif

description of the network is in file 'e_pmif'

the open net was analysed with method 'priomva2m'  .


jobclass 1

priomva2m  |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
CPU        |  20.000  20.000   0.000   0.006   0.000   0.006   0.000   0.000
ATM        |  11.000  11.000   1.000   0.000   1.000  11.000   0.000   0.000
DISKS      |   8.000   8.000   0.050   0.400   0.111   0.889   0.061   0.489


jobclass 2

priomva2m  |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
CPU        |  10.000  10.000   0.000   0.003   0.000   0.003   0.000   0.000
ATM        |   6.000   6.000   1.000   0.000   1.000   6.000   0.000   0.000
DISKS      |   3.000   3.000   0.050   0.150   0.111   0.333   0.061   0.183


total performance indices:

priomva2m  |  lambda     rho     maa
-----------+--------------------------
CPU        |  30.000   0.009   0.009
ATM        |  17.000   0.000  17.000
DISKS      |  11.000   0.550   1.222



characteristic indices:

priomva2m  |  lambda     mvz     maa
-----------+--------------------------
class  1   |   1.000  11.895  11.895
class  2   |   1.000   6.336   6.336




legend

e   : average number of visits           mue   : service rate
rho : utilisation                        lambda: mean throughput
mvz : average response time
maa : average number of jobs
mwz : average waiting time
mwsl: average queue-length
```

MQNA1:

```
           MQNA - Markovian Queueing Networks Analiser
  MQNA is a software tool for analisys and solving of Markovian
Queueing
 Networks.
 ... ... ... ...
     This MQNA version was released on: Nov 21 2001
     compiled on: Feb 27 2007 (version 1.0)
     Created by: Leonardo Brenner (lbrenner@inf.pucrs.br)
     Adviser by: Paulo Fernandes (paulof@inf.pucrs.br)
     User Interface Modification for Batch execution by:
                 Samuel Zallocco (zallocco@univaq.it)
============================================================
Model............: ATM_Orig_PMIF_Example
Model type.......: Open Queueing Network
Number of classes: 2
Number of queues.: 5
============================================================
--------------------------
Perfomance General Indexes
--------------------------
Queues average throughput
SOURCENODE: 2.000000e+00
CPU: 3.000000e+01
ATM: 1.700000e+01
DISKS: 1.100000e+01
SINKNODE: 2.000000e+00

Queues average population
SOURCENODE: 0.000000e+00
CPU: 8.878128e-03
ATM: -1.000000e+00
DISKS: 1.222222e+00
SINKNODE: 0.000000e+00

Queues average utilization
SOURCENODE: 0.000000e+00
CPU: 8.800000e-03
ATM: -1.000000e+00
DISKS: 5.500000e-01
SINKNODE: 0.000000e+00

Queues average response time
SOURCENODE: 0.000000e+00
CPU: 2.959376e-04
ATM: -1.000000e+00
DISKS: 1.111111e-01
SINKNODE: 0.000000e+00

--------------------------------
Performance Indexes by classes
--------------------------------
Class: WITHDRAWAL

Queues average throughput
SOURCENODE: 1.000000e+00
CPU: 2.000000e+01
ATM: 1.100000e+01
DISKS: 8.000000e+00
SINKNODE: 1.000000e+00

Queues average population
SOURCENODE: 0.000000e+00
CPU: 6.355932e-03
ATM: -1.000000e+00
DISKS: 8.888889e-01
SINKNODE: 0.000000e+00

Queues average utilization
SOURCENODE: 0.000000e+00
CPU: 6.300000e-03
ATM: 1.100000e+01
DISKS: 4.000000e-01
SINKNODE: 0.000000e+00

Queues average response time
SOURCENODE: 0.000000e+00
CPU: 3.177966e-04
ATM: -1.000000e+00
DISKS: 1.111111e-01
SINKNODE: 0.000000e+00

Class: GET_BALANCE

Queues average throughput
SOURCENODE: 1.000000e+00
CPU: 1.000000e+01
ATM: 6.000000e+00
DISKS: 3.000000e+00
SINKNODE: 1.000000e+00

Queues average population
SOURCENODE: 0.000000e+00
CPU: 2.522195e-03
ATM: -1.000000e+00
DISKS: 3.333333e-01
SINKNODE: 0.000000e+00

Queues average utilization
SOURCENODE: 0.000000e+00
CPU: 2.500000e-03
ATM: 6.000000e+00
DISKS: 1.500000e-01
SINKNODE: 0.000000e+00

Queues average response time
SOURCENODE: 0.000000e+00
CPU: 2.522195e-04
ATM: -1.000000e+00
DISKS: 1.111111e-01
SINKNODE: 0.000000e+00
```

12.1.3 Comparazione dei risultati numerici ottenuti:

In rosso gli errori di calcolo e quelli relativi a problemi di instabilità del modello.

PDQ	ERROR in model: "ATM_Orig_PMIF_Example" at PDQ_Canonical: Arrival rate 1.000 exceeds system saturation 0.091 = 1/11.000

OPENQN	The system is not stable.

PEPSY	Throughput	Population	Utilization	Res. Time
Withdrawal	1.000000e+00	1.189500e+01	-	1.189500e+01
GetBalance	1.000000e+00	6.336000e+00	-	6.336000e+00
CPU/ Withdrawal	2.000000e+01	6.000000e-03	6.000000e-03	0.000000e+00*
DISK/ Withdrawal	8.000000e+00	8.890000e-01	4.000000e-01	1.110000e-01
ATM/ Withdrawal	1.100000e+01	1.100000e+01	0.000000e+00	1.000000e+00
CPU/ GetBalance	1.000000e+01	3.000000e-03	3.000000e-03	0.000000e+00*
DISK/ GetBalance	3.000000e+00	3.330000e-01	1.500000e-01	1.110000e-01
ATM/ GetBalance	6.000000e+00	6.000000e+00	0.000000e+00	1.000000e+00
CPU	3.000000e+01	9.000000e-03	9.000000e-03	-
DISK	1.100000e+01	1.222000e+00	5.500000e-01	-
ATM	1.700000e+01	1.700000e+01	0.000000e+00	-

*Questi valori sono dovuti al fatto che Pepsy visualizza solo tre cifre decimali.

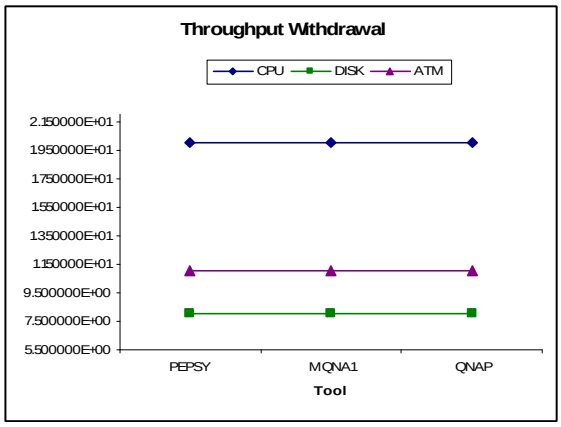
MQNA1	Throughput	Population	Utilization	Res. Time
Withdrawal	-	-	-	-
GetBalance	-	-	-	-
CPU/ Withdrawal	2.000000e+01	6.355932e-03	6.300000e-03	3.177966e-04
DISK/ Withdrawal	8.000000e+00	8.888889e-01	4.000000e-01	1.111111e-01
ATM/ Withdrawal	1.100000e+01	-1.000000e+00	1.100000e+01	-1.000000e+00
CPU/ GetBalance	1.000000e+01	2.522195e-03	2.500000e-03	2.522195e-04
DISK/ GetBalance	3.000000e+00	3.333333e-01	1.500000e-01	1.111111e-01
ATM/ GetBalance	6.000000e+00	-1.000000e+00	6.000000e+00	-1.000000e+00
CPU	3.000000e+01	8.878128e-03	8.800000e-03	2.959376e-04
DISK	1.100000e+01	1.222222e+00	5.500000e-01	1.111111e-01
ATM	1.700000e+01	-1.000000e+00	-1.000000e+00	-1.000000e+00

I risultati di QNAP non sono stati calcolati dal nostro web service ma sono stati presi da [14]:

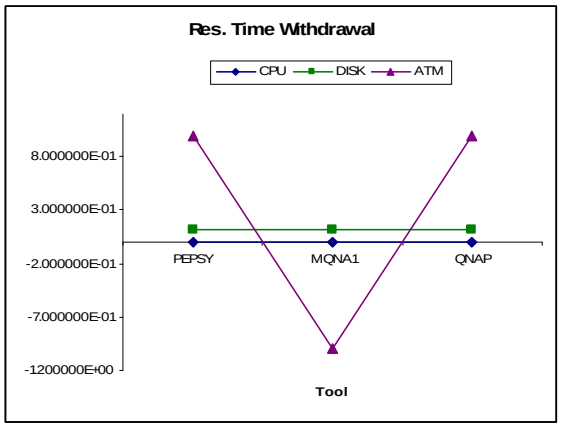
QNAP	Throughput	Population	Utilization	Res. Time
Withdrawal	-	-	-	1.190000e+01
GetBalance	-	-	-	6.336000e+00
CPU/ Withdrawal	2.000000e+01	6.400000e-03	6.300000e-03	3.200000e-04
DISK/ Withdrawal	8.000000e+00	8.889000e-01	4.000000e-01	1.111000e-01
ATM/ Withdrawal	1.100000e+01	1.100000e+01	0.000000e+00	1.000000e+00
CPU/ GetBalance	1.000000e+01	2.500000e-03	2.500000e-03	2.500000e-04
DISK/ GetBalance	3.000000e+00	3.333000e-01	1.500000e-01	1.111000e-01
ATM/ GetBalance	6.000000e+00	6.000000e+00	0.000000e+00	1.000000e+00
CPU	3.000000e+01	8.900000e-03	8.800000e-03	3.000000e-04
DISK	1.100000e+01	1.222000e+00	5.500000e-01	1.111000e-01
ATM	1.700000e+01	1.700000e+01	0.000000e+00	1.000000e+00

12.1.4 Grafici comparativi:

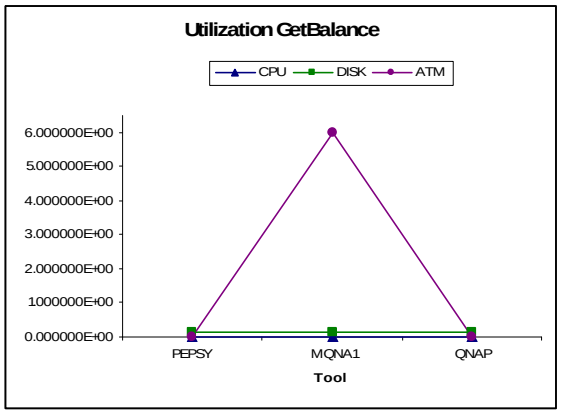
Throughput Withdrawal			
	PEPSY	MQNA1	QNAP
CPU	2.000000E+01	2.000000E+01	2.000000E+01
DISK	8.000000E+00	8.000000E+00	8.000000E+00
ATM	1.100000E+01	1.100000E+01	1.100000E+01



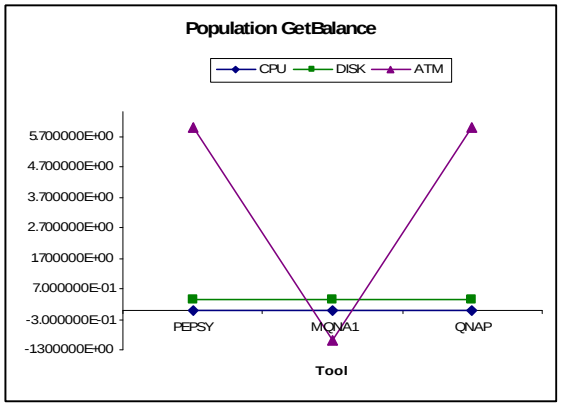
Res. Time Withdrawal			
	PEPSY	MQNA1	QNAP
CPU	0.000000E+00	3.177966E-04	3.200000E-04
DISK	1.110000E-01	1.111111E-01	1.111000E-01
ATM	1.000000E+00	-1.000000E+00	1.000000E+00



Utilization GetBalance			
	PEPSY	MQNA1	QNAP
CPU	3.000000E-03	2.500000E-03	2.500000E-03
DISK	1.500000E-01	1.500000E-01	1.500000E-01
ATM	0.000000E+00	6.000000E+00	0.000000E+00



Population GetBalance			
	PEPSY	MQNA1	QNAP
CPU	3.000000E-03	2.522195E-03	2.500000E-03
DISK	3.330000E-01	3.333333E-01	3.333000E-01
ATM	6.000000E+00	-1.000000E+00	6.000000E+00



12.2 Rete di code chiusa con 3 centri di servizio e 2 classi di clienti

Il modello che andremo ad analizzare è un semplice esempio (preso dalla documentazione di MQNA1) di rete a due classi di clienti, con tre nodi a singolo servente di cui uno di tipo IS e due di tipo FCFS. Questo modello è validato semanticamente e sintatticamente dal web service.

Rappresentazione grafica del modello:

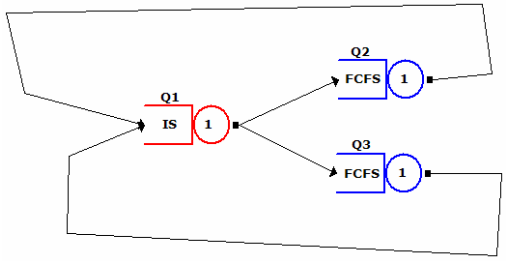


Figura 12.2 - Rete de test por MQNA1

Modello PMIF 2.0 :

```
<?xml version="1.0"?>
<QueueingNetworkModel Name="rede" Description="rede de test por PMIF 2.0 to MQNA1 conversion" Date-Time="2007-03-14T10:03:08"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://www.perfeng.com/pmif/pmifschema.xsd">

  <Node>
    <WorkUnitServer Name="Q2" Quantity="1" SchedulingPolicy="FCFS" ServiceTime="0.4" TimeUnits="Sec"/>
    <WorkUnitServer Name="Q3" Quantity="1" SchedulingPolicy="FCFS" ServiceTime="1" TimeUnits="Sec"/>
    <Server Name="Q1" Quantity="1" SchedulingPolicy="IS"/>
  </Node>
  <Arc FromNode="Q1" ToNode="Q2" Description="Arc From Q1 To Q2"/>
  <Arc FromNode="Q1" ToNode="Q3" Description="Arc From Q1 To Q3"/>
  <Arc FromNode="Q3" ToNode="Q1" Description="Arc From Q3 To Q1"/>
  <Arc FromNode="Q2" ToNode="Q1" Description="Arc From Q2 To Q1"/>
  <Workload>
    <ClosedWorkload WorkloadName="CLASSE1" NumberOfJobs="2" ThinkTime="0.2" TimeUnits="Sec" ThinkDevice="Q1">
      <Transit To="Q2" Probability="0.5"/>
      <Transit To="Q3" Probability="0.5"/>
    </ClosedWorkload>
    <ClosedWorkload WorkloadName="CLASSE2" NumberOfJobs="2" ThinkTime="0.2" TimeUnits="Sec" ThinkDevice="Q1">
      <Transit To="Q2" Probability="0.5"/>
      <Transit To="Q3" Probability="0.5"/>
    </ClosedWorkload>
  </Workload>
  <ServiceRequest>
    <WorkUnitServiceRequest WorkloadName="CLASSE1" ServerID="Q2" NumberOfVisits="1">
      <Transit To="Q1" Probability="1"/>
    </WorkUnitServiceRequest>
    <WorkUnitServiceRequest WorkloadName="CLASSE1" ServerID="Q3" NumberOfVisits="1">
      <Transit To="Q1" Probability="1"/>
    </WorkUnitServiceRequest>
    <WorkUnitServiceRequest WorkloadName="CLASSE2" ServerID="Q2" NumberOfVisits="1">
      <Transit To="Q1" Probability="1"/>
    </WorkUnitServiceRequest>
    <WorkUnitServiceRequest WorkloadName="CLASSE2" ServerID="Q3" NumberOfVisits="1">
      <Transit To="Q1" Probability="1"/>
    </WorkUnitServiceRequest>
  </ServiceRequest>
</QueueingNetworkModel>
```

12.2.1 Traduzioni possibili per modelli chiusi:

- PDQ (Anche risoluzione)
- QNAP (Solo traduzione)
- CLOSEDQN (Anche risoluzione)
- MQNA1 (Anche risoluzione)
- MQNA2 (Solo traduzione)
- MVACCKSW (Anche risoluzione)
- PEPSY (Anche risoluzione)
- PMVA (Anche risoluzione)
- SHARPE (Anche risoluzione)

Modello PDQ:

```
// This is a PDQ Model:
//  - Name: rede
//  - Description: rede de test por PMIF 2.0 to MQNA1 conversion
//  - Date-Time: 2007-03-14T11:03:38
// Generated from the original PMIF 2.0 Model using PMIF2_to_PDQ.xsl
// that was part of the Queueing Network Solver Webservice Project
// Copyright (c) by Samuel Zallocco - University of L'Aquila - ITALY - All right reserved
PROMPT OFF  // Do not show the command prompt
MESSAGE OFF // Do not show the PDQ Shell execution messages
ERROR OFF   // Do not report PDQ Shell error messages but only PDQ LIB errors
//
// Model Type: Multiple-Chain Closed Queueing Network Model
//
INIT "rede"
// SERVER
CREATENODE "Q1" CEN IS // Number of Processing Units = 1
// WORKUNITSERVER
VAR $Q2_ServiceTime = 0.4
CREATENODE "Q2" CEN FCFS // Number of Processing Units = 1
VAR $Q3_ServiceTime = 1
CREATENODE "Q3" CEN FCFS // Number of Processing Units = 1
// Closed Workloads in PMIF 2.0 are TERMINAL (if NumberOfJobs > 0) or BATCH (if NumberOfJobs = 0) Workloads in PQD
CREATECLOSED "CLASSE1" TERM 2 0.2
CREATECLOSED "CLASSE2" TERM 2 0.2
// Work Unit Service Request
SETVISITS "Q2" "CLASSE1" 1 $Q2_ServiceTime
SETVISITS "Q3" "CLASSE1" 1 $Q3_ServiceTime
SETVISITS "Q2" "CLASSE2" 1 $Q2_ServiceTime
SETVISITS "Q3" "CLASSE2" 1 $Q3_ServiceTime
// Solve the model using one of EXACT, APPROX or CANON methods
SOLVE EXACT
REPORT
EXIT
```

Modello QNAP:

```
& This is a QNAP Model:
&  - Name: rede
&  - Description: rede de test por PMIF 2.0 to MQNA1 conversion
&  - Date-Time: 2007-03-14T11:03:38
& Generated from the original PMIF 2.0 Model using PMIF2_to_QNAP.xsl
& that was part of the Queueing Network Solver Webservice Project
& Original version (c) by Catalina M. Llado (Universitat de les Illes Balears)
& Pretty Layout Modification (c) by Samuel Zallocco (University of L'Aquila Italy)
/DECLARE/   QUEUE Q1;
            QUEUE Q2, Q3;
            CLASS CLASSE1, CLASSE2;
            REAL TCLASSE1, TCLASSE2;
& For Each Server
/STATION/   NAME= Q1;
            TYPE = INFINITE;
& For Each WorkUnitServer
/STATION/   NAME = Q2;
            SERVICE = EXP(0.4);
            SCHED = FIFO;
/STATION/   NAME = Q3;
            SERVICE = EXP(1);
            SCHED = FIFO;
& For Each ClosedWorkload
/STATION/   NAME = Q1;
            INIT(CLASSE1) = 2;
            SERVICE(CLASSE1) = EXP(5);
            TRANSIT(CLASSE1)= Q3, 0.5, Q2, 0.5;
/STATION/   NAME = Q1;
            INIT(CLASSE2) = 2;
            SERVICE(CLASSE2) = EXP(5);
            TRANSIT(CLASSE2)= Q3, 0.5, Q2, 0.5;
& For Each WorkUnitServiceRequest
/STATION/   NAME = Q2;
            TRANSIT(CLASSE1) = Q1, 1;
/STATION/   NAME = Q3;
            TRANSIT(CLASSE1) = Q1, 1;
/STATION/   NAME = Q2;
            TRANSIT(CLASSE2) = Q1, 1;
/STATION/   NAME = Q3;
            TRANSIT(CLASSE2) = Q1, 1;
& Start Analisys
/CONTROL/   CLASS = ALL QUEUE;
/EXEC/ BEGIN
            SOLVE;
            & --- WorkUnitServiceRequest ---
            PRINT ("RESIDENCE TIME FOR CLASSE1,Q2 = " , MCUSTNB(Q2,CLASSE1) / MTHRUPUT (Q2,CLASSE1) );
            TCLASSE1 := TCLASSE1 + MCUSTNB(Q2,CLASSE1);
            PRINT ("RESIDENCE TIME FOR CLASSE1,Q3 = " , MCUSTNB(Q3,CLASSE1) / MTHRUPUT (Q3,CLASSE1) );
            TCLASSE1 := TCLASSE1 + MCUSTNB(Q3,CLASSE1);
            PRINT ("RESIDENCE TIME FOR CLASSE2,Q2 = " , MCUSTNB(Q2,CLASSE2) / MTHRUPUT (Q2,CLASSE2) );
            TCLASSE2 := TCLASSE2 + MCUSTNB(Q2,CLASSE2);
            PRINT ("RESIDENCE TIME FOR CLASSE2,Q3 = " , MCUSTNB(Q3,CLASSE2) / MTHRUPUT (Q3,CLASSE2) );
            TCLASSE2 := TCLASSE2 + MCUSTNB(Q3,CLASSE2);
            & --- ClosedWorkload ---
            TCLASSE1 := ( 2 - MCUSTNB( Q1 ) ) / MTHRUPUT(Q1);
            PRINT ("RESPONSE TIME FOR CLASSE1 = ", TCLASSE1);
            TCLASSE2 := ( 2 - MCUSTNB( Q1 ) ) / MTHRUPUT(Q1);
            PRINT ("RESPONSE TIME FOR CLASSE2 = ", TCLASSE2);
      END;
/END/
```

Modello CLOSEDQN:

```
// This is a CLOSEDQN Model:
//  - Name: rede
//  - Description: rede de test por PMIF 2.0 to MQNA1 conversion
//  - Date-Time: 2007-03-14T11:03:38
// Generated from the original PMIF 2.0 Model using PMIF2_to_CLOSEDQN.xsl
// that was part of the Queueing Network Solver Webservice Project
// (CC) by Samuel Zallocco - University of L'Aquila - ITALY - Some right reserved
// Model Type: Multiple-Chain Closed Queueing Network Model
CLOSEDQN "rede"
          // Class Definition Section
          Classes 2
                    // CLASS {WorkloadName} NumberOfJobs
                    Class  {CLASSE1} 2
                    Class  {CLASSE2} 2
          // Server Definition Section
          Nodes 3
                    // Server_Or_WorkUnit Name Type [Saturation ( service_rates... )]
                    Server [Q1] DELAY 0
                    WorkUnitServer [Q2] LI 0
                    WorkUnitServer [Q3] LI 0
          // Service Demand Matrix
          ServiceDemands
                    // Server_Name          Class_Name = Demand   Class_Name = Demand    ...       Class_name = Demand
                    [Q1]                    {CLASSE1} = 0.2       {CLASSE2} = 0.2
                    [Q2]                    {CLASSE1} = 0.4       {CLASSE2} = 0.4
                    [Q3]                    {CLASSE1} = 1         {CLASSE2} = 1
END
```

Modello MQNA1:

```
cqn rede (2,3)
//_This_is_a_Closed_MQNA1_Model
//__-_Name:_rede
//__-_Description:_rede_de_test_por_PMIF_2.0_to_MQNA1_conversion
//__-_Date-Time:_2007-03-14T11:03:38
//_Generated_from_the_original_PMIF_2.0_Model_using_PMIF2_to_MQNA1.xsl
//_that_was_part_of_the_Queueing_Network_Solver_Webservice_Project
//_(CC)_Samuel_Zallocco_-_University_of_L_Aquila_-_ITALY_-_Some_right_reserved
declarations {
  class CLASSE1;
  class CLASSE2;
  queue Q1;
  queue Q2;
  queue Q3;
}
network {
  population class CLASSE1 = 2;
  population class CLASSE2 = 2;
  queue Q1 {
    servers = 1;
    capacity = 10000;
    class CLASSE1 {
      service time = 0.2;
      rot Q3: 0.5;
      rot Q2: 0.5;
    }
    class CLASSE2 {
      service time = 0.2;
      rot Q3: 0.5;
      rot Q2: 0.5;
    }
  }
  queue Q2 {
    servers = 1;
    capacity = 10000;
    class CLASSE1 {
      service time = 0.4;
      rot Q1: 1;
    }
    class CLASSE2 {
      service time = 0.4;
      rot Q1: 1;
    }
  }
  queue Q3 {
    servers = 1;
    capacity = 10000;
    class CLASSE1 {
      service time = 1;
      rot Q1: 1;
    }
    class CLASSE2 {
      service time = 1;
      rot Q1: 1;
    }
  }
}
```

Modello MQNA2:

```
cqn rede (2,3)
//_This_is_a_Closed_MQNA2_Model
//__-_Name:_rede
//__-_Description:_rede_de_test_por_PMIF_2.0_to_MQNA1_conversion
//__-_Date-Time:_2007-03-14T11:03:38
//_Generated_from_the_original_PMIF_2.0_Model_using_PMIF2_to_MQNA2.xsl
//_that_was_part_of_the_Queueing_Network_Solver_Webservice_Project
//_(CC)_Samuel_Zallocco_-_University_of_L_Aquila_-_ITALY_-_Some_right_reserved
declarations {
  class CLASSE1;
  class CLASSE2;
  queue Q1;
  queue Q2;
  queue Q3;
}
network {
  population class CLASSE1 = 2;
  population class CLASSE2 = 2;
  queue Q1 {
    priority (1,1);
    servers = 1;
    capacity = 10000;
    class CLASSE1 {
      service time = 0.2;
      routing Q3(blocking): 0.5;
      routing Q2(blocking): 0.5;
    }
    class CLASSE2 {
      service time = 0.2;
      routing Q3(blocking): 0.5;
      routing Q2(blocking): 0.5;
    }
  }
  queue Q2 {
    priority (1,1);
    servers = 1;
    capacity = 10000;
    class CLASSE1 {
      service time = 0.4;
      routing Q1(blocking): 1;
    }
    class CLASSE2 {
      service time = 0.4;
      routing Q1(blocking): 1;
    }
  }
  queue Q3 {
    priority (1,1);
    servers = 1;
    capacity = 10000;
    class CLASSE1 {
      service time = 1;
      routing Q1(blocking): 1;
    }
    class CLASSE2 {
      service time = 1;
      routing Q1(blocking): 1;
    }
  }
}
```

Modello MVACCKSW:

```
# This is a MVACCKSW Model:
#  - Name: rede
#  - Description: rede de test por PMIF 2.0 to MQNA1 conversion
#  - Date-Time: 2007-03-14T11:03:38
# Generated from the original PMIF 2.0 Model using PMIF2_to_CLOSEDQN.xsl
# that was part of the Queueing Network Solver Webservice Project
# (CC) by Samuel Zallocco - University of L'Aquila - ITALY - Some right reserved
# Model Type: Multiple-Chain Closed Queueing Network Model
#M "rede"  // MODEL NAME
# CLASS DEFINITION
#  NUM  NAME
#C "0" "CLASSE1"
#C "1" "CLASSE2"
# NODE DEFINITION
#  NUM  NAME
#S "0" "Q1"
#S "1" "Q2"
#S "2" "Q3"
# Queueing formalism number:
1
# Number-of-Classes Number-of-Servers
2 3
# Population
2 2
# Think time
0.2 0.2
# Service demands for Class
# SERVER1 ... SERVERn
0.0 0.4 1
0.0 0.4 1
# **Done**
END
```

Modello PEPSY:

```
# @(#)PEPSY:        e_rede    1.10      2007/03/24 19:03:09
#-------------------------------------------------------------------------------------------
#                             P E P S Y
#                     Performance and Evaluation Prediction SYstem
# This is a PEPSY-QNS Model:
#  - Name: rede
#  - Description: rede de test por PMIF 2.0 to MQNA1 conversion
#  - Date-Time: 2007-03-24T19:03:09
#  - Model Type: Multiple-Chain Closed Queueing Network Model
# Generated from the original PMIF 2.0 Model using PMIF2_to_PEPSY.xsl
# that was part of the Queueing Network Solver Webservice Project
# (CC) Samuel Zallocco - University of L'Aquila - ITALY - Some right reserved
#-------------------------------------------------------------------------------------------
# inputfile version  3

NUMBER NODES: 3
NUMBER CLASSES: 2

NODE SPECIFICATION

  node   |      name          |     type
---------+--------------------+----------------------
     1   |      Q1            |     M/G/0-IS
     2   |      Q2            |     M/M/1-FCFS
     3   |      Q3            |     M/M/1-FCFS

CLASS SPECIFICATION

  class   | arrival rate   number of jobs
----------+---------------------------------
     1    |    -              2
     2    |    -              2

CLASS SPECIFIC PARAMETERS

CLASS 1

     node           |  service_rate    squared_coeff._of_variation
--------------------+-----------------------------------------------
     Q1             |  5               1
     Q2             |  2.5             1
     Q3             |  1               1

SWITCHING PROBABILITIES

from/to    | outside    Q1         Q2         Q3
-----------+-----------------------------------------------
outside    | 0.000000   1.000000   0.000000   0.000000
Q1         | 0.000000   0.000000   0.500000   0.500000
Q2         | 1.000000   0.000000   0.000000   0.000000
Q3         | 1.000000   0.000000   0.000000   0.000000

CLASS 2

     node           |  service_rate    squared_coeff._of_variation
--------------------+-----------------------------------------------
     Q1             |  5               1
     Q2             |  2.5             1
     Q3             |  1               1

SWITCHING PROBABILITIES

from/to    | outside    Q1         Q2         Q3
-----------+-----------------------------------------------
outside    | 0.000000   1.000000   0.000000   0.000000
Q1         | 0.000000   0.000000   0.500000   0.500000
Q2         | 1.000000   0.000000   0.000000   0.000000
Q3         | 1.000000   0.000000   0.000000   0.000000
```

Modello PMVA:

```
"
 This is a PMVA Model:
  - Name: rede
  - Description: rede de test por PMIF 2.0 to MQNA1 conversion
  - Date-Time: 2007-03-14T11:03:38
 Generated from the original PMIF 2.0 Model using PMIF2_to_PMVA.xsl
 that was part of the Queueing Network Solver Webservice Project
 (CC) by Samuel Zallocco - University of L'Aquila - ITALY - Some right reserved"
 Model Type: Multiple-Chain Closed Queueing Network Model
"
NETWORK;
CLASSES CLASSE1, CLASSE2;
SERVERS
        Q1 IS LI CLASSE1=0.2 CLASSE2=0.2,
        Q2 FCFS LI ALL = 0.4,
        Q3 FCFS LI ALL = 1;
ROUTING
        Q1/CLASSE1 => Q3 (0.5) Q2 (0.5),
        Q1/CLASSE2 => Q3 (0.5) Q2 (0.5),
        Q2/CLASSE1 => Q1 (1),
        Q3/CLASSE1 => Q1 (1),
        Q2/CLASSE2 => Q1 (1),
        Q3/CLASSE2 => Q1 (1);
END;
MVA POP = (2,2);
STOP;
```

Modello SHARPE:

```
* This is a SHARPE Model:
*  - Name: rede
*  - Description: rede de test por PMIF 2.0 to MQNA1 conversion
*  - Date-Time: 2007-03-24T19:03:09
* Generated from the original PMIF 2.0 Model using PMIF2_to_SHARPE_PFQN.xsl
* that was part of the Queueing Network Solver Webservice Project
* (CC) Samuel Zallocco - University of L'Aquila - ITALY - Some right reserved
* Model Type: Multiple-Chain Closed Queueing Network Model
mpfqn rede

  * Section 1: station-to-station probabilities for each chain
    chain CLASSE1
        * [From-Server] [To-Server] [Probability] Derived from Transit
                  Q1 Q3 0.5
                  Q1 Q2 0.5

        * [From-Server] [To-Server] [Probability] Derived from WorkUnitServiceRequest
                  Q2 Q1 1
                  Q3 Q1 1
    end

    chain CLASSE2
        * [From-Server] [To-Server] [Probability] Derived from Transit
                  Q1 Q3 0.5
                  Q1 Q2 0.5

        * [From-Server] [To-Server] [Probability] Derived from WorkUnitServiceRequest
                  Q2 Q1 1
                  Q3 Q1 1
    end
  end

  * Section 2: Station types and parameters
        * [Server] [Scheduling-Policy] [Service-Rate] Derived from Server + DemandServiceRequest or Server + TimeServiceRequest
                  Q1 IS
                    CLASSE1 5
                    CLASSE2 5
                  end

        * [Server] [Scheduling-Policy] [Service-Rate] Derived from WorkUnitServer
                    Q2 FCFS 2.5
                  end
                    Q3 FCFS 1
                  end
  end

  * Section 3: number of customer per chain
                  CLASSE1 2
                  CLASSE2 2
end

echo----- rede Model Results -----
echo
format 4
verbose off

echo Results for: "CLASSE1 at Q1":
bind Throughput____ mtput(rede, Q1, CLASSE1)
bind Utilization___ mutil(rede, Q1, CLASSE1)
bind Queue_Length__ mqlength(rede, Q1, CLASSE1)
bind Residence_Time mrtime(rede, Q1, CLASSE1)
expr Throughput____, Utilization___, Queue_Length__, Residence_Time

echo Results for: "CLASSE1 at Q2":
bind Throughput____ mtput(rede, Q2, CLASSE1)
bind Utilization___ mutil(rede, Q2, CLASSE1)
bind Queue_Length__ mqlength(rede, Q2, CLASSE1)
bind Residence_Time mrtime(rede, Q2, CLASSE1)
expr Throughput____, Utilization___, Queue_Length__, Residence_Time

echo Results for: "CLASSE1 at Q3":
bind Throughput____ mtput(rede, Q3, CLASSE1)
bind Utilization___ mutil(rede, Q3, CLASSE1)
bind Queue_Length__ mqlength(rede, Q3, CLASSE1)
bind Residence_Time mrtime(rede, Q3, CLASSE1)
expr Throughput____, Utilization___, Queue_Length__, Residence_Time

echo Results for: "CLASSE2 at Q1":
bind Throughput____ mtput(rede, Q1, CLASSE2)
bind Utilization___ mutil(rede, Q1, CLASSE2)
bind Queue_Length__ mqlength(rede, Q1, CLASSE2)
bind Residence_Time mrtime(rede, Q1, CLASSE2)
expr Throughput____, Utilization___, Queue_Length__, Residence_Time

echo Results for: "CLASSE2 at Q2":
bind Throughput____ mtput(rede, Q2, CLASSE2)
bind Utilization___ mutil(rede, Q2, CLASSE2)
bind Queue_Length__ mqlength(rede, Q2, CLASSE2)
bind Residence_Time mrtime(rede, Q2, CLASSE2)
expr Throughput____, Utilization___, Queue_Length__, Residence_Time

echo Results for: "CLASSE2 at Q3":
bind Throughput____ mtput(rede, Q3, CLASSE2)
bind Utilization___ mutil(rede, Q3, CLASSE2)
bind Queue_Length__ mqlength(rede, Q3, CLASSE2)
bind Residence_Time mrtime(rede, Q3, CLASSE2)
expr Throughput____, Utilization___, Queue_Length__, Residence_Time

end
```

12.2.2 Output della risoluzione:

Gli unici tool utilizzabili per la risoluzione del modello chiuso sono SHARPE, CLOSEDQN, MQNA1, PDQ (2 metodi), PMVA (4 metodi), PEPSY (9 metodi) e MVACCKSW (9 metodi).

SHARPE:

```
----- rede Model Results -----
Results for: "CLASSE1 at Q1":
 Throughput____:   9.7158e-01
 Utilization___:   1.9432e-01
 Queue_Length__:   1.9432e-01
 Residence_Time:   2.0000e-01
-----------------------------------------
Results for: "CLASSE1 at Q2":
 Throughput____:   4.8579e-01
 Utilization___:   1.9432e-01
 Queue_Length__:   2.9530e-01
 Residence_Time:   6.0787e-01
-----------------------------------------
Results for: "CLASSE1 at Q3":
 Throughput____:   4.8579e-01
 Utilization___:   4.8579e-01
 Queue_Length__:   1.5104e+00
 Residence_Time:   3.1091e+00
-----------------------------------------
Results for: "CLASSE2 at Q1":
 Throughput____:   9.7158e-01
 Utilization___:   1.9432e-01
 Queue_Length__:   1.9432e-01
 Residence_Time:   2.0000e-01
-----------------------------------------
Results for: "CLASSE2 at Q2":
 Throughput____:   4.8579e-01
 Utilization___:   1.9432e-01
 Queue_Length__:   2.9530e-01
 Residence_Time:   6.0787e-01
-----------------------------------------
Results for: "CLASSE2 at Q3":
 Throughput____:   4.8579e-01
 Utilization___:   4.8579e-01
 Queue_Length__:   1.5104e+00
 Residence_Time:   3.1091e+00
-----------------------------------------
```

CLOSEQN:

```
/-------------------------------------------------------------\
|ClosedQN - (c) Copr. 1994 D. Menasce', V. Almeida, and L. Dowdy.  |
|           All Rights Reserved.                                |
|This program comes with the book 'Capacity Planning and        |
|Performance Modelling: from mainframes to client-server systems'  |
|by Menasce, Almeida, and Dowdy published by Prentice Hall.     |
|07-06-2006: Input file format modification                     |
|By Samuel Zallocco (samuel.zallocco@univaq.it)                 |
|University of L'Aquila - Italy                                 |
\-------------------------------------------------------------/
Reading input file...
Model Name: "REDE"
  N. of Classes = 2
    Class1: "CLASSE1" Population: 2
    Class2: "CLASSE2" Population: 2
    Total Classes Population: 4
  N. of Servers/Nodes = 3
    Server1: "Q1" Type: DELAY Saturation: 0
    Server2: "Q2" Type: LOAD INDEPENDENT Saturation: 0
    Server3: "Q3" Type: LOAD INDEPENDENT Saturation: 0
  Service Demands:
    "Q1" "CLASSE1" =  0.2000000 "CLASSE2" =  0.2000000
    "Q2" "CLASSE1" =  0.4000000 "CLASSE2" =  0.4000000
    "Q3" "CLASSE1" =  1.0000000 "CLASSE2" =  1.0000000
End of Model Description
Computing...
Total No. of Iterations:   9 ( Error <=   0.000083 )
Results...
Class 1: "CLASSE1" metrics:
  Device Residence Times:
     Device 1: "Q1"   0.200000
     Device 2: "Q2"   0.559087
     Device 3: "Q3"   3.460062
   Class 1: "CLASSE1" response time :   4.219149
   Class 1: "CLASSE1" throughput....:   0.474029
Class 2: "CLASSE2" metrics:
  Device Residence Times:
     Device 1: "Q1"   0.200000
     Device 2: "Q2"   0.559087
     Device 3: "Q3"   3.460062
   Class 2: "CLASSE2" response time :   4.219149
   Class 2: "CLASSE2" throughput....:   0.474029
End of Network Analysis
```

MQNA1:

```
  MQNA - Markovian Queueing Networks Analiser
  MQNA is a software tool for analysis and solving of
  Markovian Queueing Networks.
     This MQNA version was released on: Nov 21 2001
     compiled on: Feb 27 2007 (version 1.0)
     Created by: Leonardo Brenner (lbrenner@inf.pucrs.br)
     Adviser by: Paulo Fernandes (paulof@inf.pucrs.br)
     User Interface Modification for Batch execution by:
                 Samuel Zallocco (zallocco@univaq.it)
============================================================
Model.............: rede
Model type.......: Closed Queueing Network
Number of classes: 2
Number of queues.: 3
Total Population.: 4
============================================================

---------------------------
Performance General Indexes
---------------------------
Queues average throughput
Q1: 1.903904e+00
Q2: 9.519520e-01
Q3: 9.519520e-01

Queues average population
Q1: 5.705706e-01
Q2: 5.705706e-01
Q3: 2.858859e+00

Queues average utilization
Q1: 3.807808e-01
Q2: 3.807808e-01
Q3: 9.519520e-01

Queues average response time
Q1: 2.996845e-01
Q2: 5.993691e-01
Q3: 3.003155e+00

--------------------------------
Performance Indexes by classes
--------------------------------
Class: CLASSE1
Population: 2

Queues average throughput
Q1: 9.519520e-01
Q2: 4.759760e-01
Q3: 4.759760e-01

Queues average population
Q1: 2.852853e-01
Q2: 2.852853e-01
Q3: 1.429429e+00

Queues average utilization
Q1: 1.903904e-01
Q2: 1.903904e-01
Q3: 4.759760e-01

Queues average response time
Q1: 2.996845e-01
Q2: 5.993691e-01
Q3: 3.003155e+00

Class: CLASSE2
Population: 2

Queues average throughput
Q1: 9.519520e-01
Q2: 4.759760e-01
Q3: 4.759760e-01

Queues average population
Q1: 2.852853e-01
Q2: 2.852853e-01
Q3: 1.429429e+00

Queues average utilization
Q1: 1.903904e-01
Q2: 1.903904e-01
Q3: 4.759760e-01

Queues average response time
Q1: 2.996845e-01
Q2: 5.993691e-01
Q3: 3.003155e+00
```

PDQ EXACT:

```
                    ***************************************
                    ******   PDQ Model OUTPUTS      *******
                    ***************************************

Solution Method: EXACT

                    ******   SYSTEM Performance     *******

Metric                              Value          Unit
-----------------                   -----          ----
Workload: "CLASSE1"
Mean Throughput          0.4895          Job/Sec
Response Time            3.8859          Sec
Mean Concurrency         1.9021          Job
Stretch Factor           2.7756

Bounds Analysis:
Max Throughput           1.0000          Job/Sec
Min Response             1.4000          Sec
Max Demand               1.0000          Sec
Tot Demand               1.4000          Sec
Think time               0.2000          Sec
Optimal Clients          1.6000          Clients

Workload: "CLASSE2"
Mean Throughput          0.4895          Job/Sec
Response Time            3.8859          Sec
Mean Concurrency         1.9021          Job
Stretch Factor           2.7756

Bounds Analysis:
Max Throughput           1.0000          Job/Sec
Min Response             1.4000          Sec
Max Demand               1.0000          Sec
Tot Demand               1.4000          Sec
Think time               0.2000          Sec
Optimal Clients          1.6000          Clients

                    ******   RESOURCE Performance   *******

Metric          Resource     Work              Value   Unit
---------       ------       ----              -----   ----
Throughput      Q2           CLASSE1          0.4895
Visits/Sec
Utilization     Q2           CLASSE1         19.5796   Percent
Queue Length    Q2           CLASSE1          0.3018   Job
Residence Time  Q2           CLASSE1          0.6166   Sec
Waiting Time    Q2           CLASSE1          0.2166   Sec

Throughput      Q3           CLASSE1          0.4895
Visits/Sec
Utilization     Q3           CLASSE1         48.9490   Percent
Queue Length    Q3           CLASSE1          1.6003   Job
Residence Time  Q3           CLASSE1          3.2693   Sec
Waiting Time    Q3           CLASSE1          2.2693   Sec

Throughput      Q2           CLASSE2          0.4895
Visits/Sec
Utilization     Q2           CLASSE2         19.5796   Percent
Queue Length    Q2           CLASSE2          0.3018   Job
Residence Time  Q2           CLASSE2          0.6166   Sec
Waiting Time    Q2           CLASSE2          0.2166   Sec

Throughput      Q3           CLASSE2          0.4895
Visits/Sec
Utilization     Q3           CLASSE2         48.9490   Percent
Queue Length    Q3           CLASSE2          1.6003   Job
Residence Time  Q3           CLASSE2          3.2693   Sec
Waiting Time    Q3           CLASSE2          2.2693   Sec
```

PDQ APPROX:

```
                    ***************************************
                    ******   PDQ Model OUTPUTS      *******
                    ***************************************

Solution Method: APPROX      (Iterations: 8; Accuracy:
0.1000%)

                    ******   SYSTEM Performance     *******

Metric                              Value          Unit
-----------------                   -----          ----
Workload: "CLASSE1"
Mean Throughput          0.4741          Job/Sec
Response Time            4.0188          Sec
Mean Concurrency         1.9052          Job
Stretch Factor           2.8706

Bounds Analysis:
Max Throughput           1.0000          Job/Sec
Min Response             1.4000          Sec
Max Demand               1.0000          Sec
Tot Demand               1.4000          Sec
Think time               0.2000          Sec
Optimal Clients          1.6000          Clients

Workload: "CLASSE2"
Mean Throughput          0.4741          Job/Sec
Response Time            4.0188          Sec
Mean Concurrency         1.9052          Job
Stretch Factor           2.8706

Bounds Analysis:
Max Throughput           1.0000          Job/Sec
Min Response             1.4000          Sec
Max Demand               1.0000          Sec
Tot Demand               1.4000          Sec
Think time               0.2000          Sec
Optimal Clients          1.6000          Clients

                    ******   RESOURCE Performance   *******

Metric          Resource     Work              Value   Unit
---------       ------       ----              -----   ----
Throughput      Q2           CLASSE1          0.4741
Visits/Sec
Utilization     Q2           CLASSE1         18.9627   Percent
Queue Length    Q2           CLASSE1          0.2651   Job
Residence Time  Q2           CLASSE1          0.5593   Sec
Waiting Time    Q2           CLASSE1          0.1593   Sec

Throughput      Q3           CLASSE1          0.4741
Visits/Sec
Utilization     Q3           CLASSE1         47.4068   Percent
Queue Length    Q3           CLASSE1          1.6400   Job
Residence Time  Q3           CLASSE1          3.4595   Sec
Waiting Time    Q3           CLASSE1          2.4595   Sec

Throughput      Q2           CLASSE2          0.4741
Visits/Sec
Utilization     Q2           CLASSE2         18.9627   Percent
Queue Length    Q2           CLASSE2          0.2651   Job
Residence Time  Q2           CLASSE2          0.5593   Sec
Waiting Time    Q2           CLASSE2          0.1593   Sec

Throughput      Q3           CLASSE2          0.4741
Visits/Sec
Utilization     Q3           CLASSE2         47.4068   Percent
Queue Length    Q3           CLASSE2          1.6400   Job
Residence Time  Q3           CLASSE2          3.4595   Sec
Waiting Time    Q3           CLASSE2          2.4595   Sec
```

PMVA MVA:

```
>MVA POP = (2,2);
*** global performance measures ***
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             1.9432         0.2000         0.3886
 Q2             0.9716         0.6079         0.5906        0.3886
 Q3             0.9716         3.1091         3.0208        0.9716

 *** performance measures for class CLASSE1
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             0.9716         0.2000         0.1943
 Q2             0.4858         0.6079         0.2953        0.1943
 Q3             0.4858         3.1091         1.5104        0.4858

 *** performance measures for class CLASSE2
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             0.9716         0.2000         0.1943
 Q2             0.4858         0.6079         0.2953        0.1943
 Q3             0.4858         3.1091         1.5104        0.4858
```

PMVA SCHWEITZER:

```
>SCHWEITZER POP = (2,2);
 *** global performance measures ***
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             1.8662         0.2000         0.3732
 Q2             0.9331         0.5555         0.5183        0.3732
 Q3             0.9331         3.3313         3.1084        0.9331

 *** performance measures for class CLASSE1
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             0.9331         0.2000         0.1866
 Q2             0.4665         0.5555         0.2592        0.1866
 Q3             0.4665         3.3313         1.5542        0.4665

 *** performance measures for class CLASSE2
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             0.9331         0.2000         0.1866
 Q2             0.4665         0.5555         0.2592        0.1866
 Q3             0.4665         3.3313         1.5542        0.4665
```

PMVA LIN:

```
>LIN POP = (2,2);
 *** global performance measures ***
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             1.9374         0.2000         0.3875
 Q2             0.9687         0.6028         0.5839        0.3875
 Q3             0.9687         3.1265         3.0286        0.9687

 *** performance measures for class CLASSE1
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             0.9687         0.2000         0.1937
 Q2             0.4843         0.6028         0.2920        0.1937
 Q3             0.4843         3.1265         1.5143        0.4843

 *** performance measures for class CLASSE2
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             0.9687         0.2000         0.1937
 Q2             0.4843         0.6028         0.2920        0.1937
 Q3             0.4843         3.1265         1.5143        0.4843
```

PMVA ASYMP :

```
>ASYMP POP = (2,2);
 *** global performance measures ***
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             1.5298         0.2000         0.3060
 Q2             0.7649         0.5763         0.4408        0.3060
 Q3             0.7649         4.2532         3.2532        0.7649

 *** performance measures for class CLASSE1
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             0.7649         0.2000         0.1530
 Q2             0.3824         0.5763         0.2204        0.1530
 Q3             0.3824         4.2532         1.6266        0.3824

 *** performance measures for class CLASSE2
 service                  mean waiting    mean queue
 center      throughput        time          length      utilization
 Q1             0.7649         0.2000         0.1530
 Q2             0.3824         0.5763         0.2204        0.1530
 Q3             0.3824         4.2532         1.6266        0.3824
```

PEPSY:
Pepsy è in grado di analizzare il modello a reti di code e di suggerire quali metodi sono ad esso applicabili. Invocando la risoluzione del web service, senza specificare alcun metodi di risoluzione, Pepsy, per il modello in oggetto, fornisce la seguente lista dei possibili metodi applicabili:

```
PEPSY Auswahl wrapper execution result on /tmp/QNMxq97TF

e_pmif:
    usable                |  need further specification
------------------------------------------------------------
    ammva                 |
    bol_aky               |
    cmva                  |
                          |  hm
    mmva                  |
    multisum              |
    num_app               |
    pm_2                  |
    priomva2c             |
    priomva2m             |
    recal                 |
                          |  sim2

These are the only solution methods you can use to solve your model.
Only the usable column methods are available, the other need additional parameters specification
```

Nella colonna “usable” sono presenti i metodi di risoluzione applicabili al modello, nella colonna “need further specification” sono presenti i metodi usabili ma che necessitano di ulteriori parametri per la loro esecuzione. Questi ultimi sono generalmente metodi simulativi che, ad esempio, hanno bisogno che venga specificato il numero massimo di cicli da effettuare.

Il significato delle colonne delle tabelle fornite in output da Pepsy è il seguente:

- **e**: average number of visits;
- **rho**: utilisation;
- **mvz**: average response time;
- **maa**: average number of jobs;
- **mwz**: average waiting time;
- **mwsl**: average queue-length;
- **mue**: service rate;
- **lambda**: mean throughput;

PEPSY AMMVA:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNMkAl5PX  with method:  ammva
PERFORMANCE_INDICES FOR NET:  pmif
description of the network is in file 'e_pmif'
the closed net was analysed with method 'ammva'  .

jobclass 1

ammva      |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025

jobclass 2

ammva      |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025

total performance indices:

ammva      |  lambda     rho     maa
-----------+--------------------------
Q1         |   1.943   0.000   0.389
Q2         |   0.972   0.389   0.591
Q3         |   0.972   0.972   3.021

characteristic indices:

ammva      |  lambda     mvz     maa
-----------+--------------------------
class  1   |   0.972   2.058   2.000
class  2   |   0.972   2.058   2.000
```

PEPSY BOL_AKY:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNM95EClV  with method:  bol_aky
PERFORMANCE_INDICES FOR NET:  pmif
description of the network is in file 'e_pmif'
the closed net was analysed with method 'bol_aky'  .

jobclass 1

bol_aky    |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
Q1         |   0.970   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.485   0.500   0.400   0.194   0.607   0.295   0.207   0.101
Q3         |   0.485   0.500   1.000   0.485   3.115   1.511   2.115   1.026

jobclass 2

bol_aky    |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
Q1         |   0.970   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.485   0.500   0.400   0.194   0.607   0.295   0.207   0.101
Q3         |   0.485   0.500   1.000   0.485   3.115   1.511   2.115   1.026

total performance indices:

bol_aky    |  lambda     rho     maa
-----------+--------------------------
Q1         |   1.941   0.000   0.388
Q2         |   0.970   0.388   0.589
Q3         |   0.970   0.970   3.023

characteristic indices:

bol_aky    |  lambda     mvz     maa
-----------+--------------------------
class  1   |   0.970   2.061   2.000
class  2   |   0.970   2.061   2.000

Anzahl der Core - Iterationen  = 27
```

PEPSY CMVA:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNMKI7Kzb  with method:  cmva
PERFORMANCE_INDICES FOR NET:  pmif
description of the network is in file 'e_pmif'
the closed net was analysed with method 'cmva'  .

jobclass 1

cmva       |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025

jobclass 2

cmva       |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025

total performance indices:

cmva       |  lambda     rho     maa
-----------+--------------------------
Q1         |   1.943   0.000   0.389
Q2         |   0.972   0.389   0.591
Q3         |   0.972   0.972   3.021

characteristic indices:

cmva       |  lambda     mvz     maa
-----------+--------------------------
class  1   |   0.972   2.058   2.000
class  2   |   0.972   2.058   2.000
```

PEPSY MMVA:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNM8fHS8F  with method:  mmva
PERFORMANCE_INDICES FOR NET:  pmif
description of the network is in file 'e_pmif'
the closed net was analysed with method 'mmva'  .

jobclass 1

mmva       |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025

jobclass 2

mmva       |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025

total performance indices:

mmva       |  lambda     rho     maa
-----------+--------------------------
Q1         |   1.943   0.000   0.389
Q2         |   0.972   0.389   0.591
Q3         |   0.972   0.972   3.021

characteristic indices:

mmva       |  lambda     mvz     maa
-----------+--------------------------
class  1   |   0.972   2.058   2.000
class  2   |   0.972   2.058   2.000
```

PEPSY MULTISUM:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNMxKRxn3  with method:  multisum
PERFORMANCE_INDICES FOR NET:  pmif
description of the network is in file 'e_pmif'
the closed net was analysed with method 'multisum'  .

jobclass 1

multisum   |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------
Q1         |   0.933   1.000   0.200   0.000   0.200   0.187   0.000   0.000
Q2         |   0.467   0.500   0.400   0.187   0.556   0.259   0.156   0.073
Q3         |   0.467   0.500   1.000   0.467   3.331   1.554   2.331   1.088

jobclass 2

multisum   |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------
Q1         |   0.933   1.000   0.200   0.000   0.200   0.187   0.000   0.000
Q2         |   0.467   0.500   0.400   0.187   0.556   0.259   0.156   0.073
Q3         |   0.467   0.500   1.000   0.467   3.331   1.554   2.331   1.088

total performance indices:

multisum   |  lambda     rho     maa
-----------+--------------------------
Q1         |   1.866   0.000   0.374
Q2         |   0.934   0.374   0.518
Q3         |   0.934   0.934   3.108

characteristic indices:

multisum   |  lambda     mvz     maa
-----------+--------------------------
class  1   |   0.933   2.144   2.000
class  2   |   0.933   2.144   2.000

Anzahl Iterationen: 8
```

PEPSY NUM_APP:

Questo metodo, se pur nella lista di quelli utilizzabili secondo Pepsy, fallisce la sua esecuzione con un messaggio di errore in tedesco poco comprensibile:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNMBjiLy5  with method:  num_app
method - Datei 'Codierung': Fehler beim Oeffnen
num_app - problems with analyse (errorcode 256)
```

PEPSY PM_2:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNM3MGLlk  with method:  pm_2
PERFORMANCE_INDICES FOR NET:  pmif
description of the network is in file 'e_pmif'
the closed net was analysed with method 'pm_2'  .

jobclass 1

pm_2       |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------
Q1         |   0.970   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.485   0.500   0.400   0.194   0.607   0.295   0.207   0.101
Q3         |   0.485   0.500   1.000   0.485   3.115   1.511   2.115   1.026

jobclass 2

pm_2       |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------
Q1         |   0.970   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.485   0.500   0.400   0.194   0.607   0.295   0.207   0.101
Q3         |   0.485   0.500   1.000   0.485   3.115   1.511   2.115   1.026

total performance indices:

pm_2       |  lambda     rho     maa
-----------+--------------------------
Q1         |   1.941   0.000   0.388
Q2         |   0.970   0.388   0.589
Q3         |   0.970   0.970   3.023

characteristic indices:

pm_2       |  lambda     mvz     maa
-----------+--------------------------
class  1   |   0.970   2.061   2.000
class  2   |   0.970   2.061   2.000

Anzahl der Core - Iterationen  = 27
```

PEPSY PRIOMVA2C:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNMCTuTi7  with method:  priomva2c
PERFORMANCE_INDICES FOR NET:  pmif
description of the network is in file 'e_pmif'
the closed net was analysed with method 'priomva2c'  .

jobclass 1

priomva2c  |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025

jobclass 2

priomva2c  |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025

total performance indices:

priomva2c  |  lambda     rho     maa
-----------+--------------------------
Q1         |   1.943   0.000   0.389
Q2         |   0.972   0.389   0.591
Q3         |   0.972   0.972   3.021

characteristic indices:

priomva2c  |  lambda     mvz     maa
-----------+--------------------------
class  1   |   0.972   2.059   2.000
class  2   |   0.972   2.059   2.000
```

PEPSY PRIOMVA2M:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNMOZyUm8  with method:  priomva2m
PERFORMANCE_INDICES FOR NET:  pmif
description of the network is in file 'e_pmif'
the closed net was analysed with method 'priomva2m'  .

jobclass 1

priomva2m  |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025

jobclass 2

priomva2m  |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025

total performance indices:

priomva2m  |  lambda     rho     maa
-----------+--------------------------
Q1         |   1.943   0.000   0.389
Q2         |   0.972   0.389   0.591
Q3         |   0.972   0.972   3.021

characteristic indices:

priomva2m  |  lambda     mvz     maa
-----------+--------------------------
class  1   |   0.972   2.059   2.000
class  2   |   0.972   2.059   2.000
```

PEPSY RECAL:

```
PEPSY Analyse wrapper execution result on:  /tmp/QNMiIrDfq  with method:  recal
PERFORMANCE_INDICES FOR NET:  pmif
description of the network is in file 'e_pmif'
the closed net was analysed with method 'recal'  .

jobclass 1

recal      |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025


jobclass 2

recal      |  lambda      e    1/mue     rho     mvz     maa     mwz    mwsl
-----------+-------------------------------------------------------------------
Q1         |   0.972   1.000   0.200   0.000   0.200   0.194   0.000   0.000
Q2         |   0.486   0.500   0.400   0.194   0.608   0.295   0.208   0.101
Q3         |   0.486   0.500   1.000   0.486   3.109   1.510   2.109   1.025


total performance indices:

recal      |  lambda     rho     maa
-----------+--------------------------
Q1         |   1.943   0.000   0.389
Q2         |   0.972   0.389   0.591
Q3         |   0.972   0.972   3.021



characteristic indices:

recal      |  lambda     mvz     maa
-----------+--------------------------
class  1   |   0.972   2.059   2.000
class  2   |   0.972   2.059   2.000
```

MVACCKSW:
MVACCKSW come PEPSY, PMVA e PDQ supporta più metodi di risoluzione specificabili dal menu dell'applicazione da noi sviluppata. Tra questi metodi c'è un metodo speciale (metodo 0) che fornisce l'output di tutti i metodi in un unico file. Riportiamo quindi l'intestazione del file di output poi separatamente i risultati dei vari metodi.

Intestazione comune:

```
 ------------------------------------------------------------------------
| MVA Queueing Formalism Parser                                          |
|                                                                        |
| GNU Mean Value Analysis Algorithm Implementation by                    |
|   Chandrakanth Chereddi <chereddi@gmail.com> or <cchered2@uiuc.edu>    |
|                                                                        |
| Modification for Model, Class and Server naming by                     |
|   Samuel Zallocco University of L'Aquila (ITALY) <zallocco@univaq.it> |
 ------------------------------------------------------------------------

 -------------------
| Model Description |
 -------------------

Model Name "rede"
Class id 0 is "CLASSE1"
Class id 1 is "CLASSE2"
Server id 0 is "Q1"
Server id 1 is "Q2"
Server id 2 is "Q3"

 --------------------------
| Queueing formalism: rede |
 --------------------------

Classes: 2
Servers: 3
Population:
  Class CLASSE1 = 2 Jobs
  Class CLASSE2 = 2 Jobs

Think Time:
  Class CLASSE1 = 0.200 Sec
  Class CLASSE2 = 0.200 Sec

Service demand:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.400
  Class CLASSE2 at Server Q2 = 0.400
  Class CLASSE1 at Server Q3 = 1.000
  Class CLASSE2 at Server Q3 = 1.000
```

MVACCKSW EXACT MVA:

```
 -----------------
| MVA Computation |
 -----------------

PropEst Total iterations 8
PropEst II Total iterations 7
QL Total iterations 6
FL Total iterations 6
Quitting from AQL QL

 ---------------
| MVA Solutions |
 ---------------

-----------------------------------------------------------------------------------------------
 Mean values for Queueing formalism: "rede"
 Algorithm used: Exact  MVA - Exact Mean Value Analysis by M. Raiser and S. S. Lavenberg
-----------------------------------------------------------------------------------------------

Classes: 2
Servers: 3
Throughput:
  Class CLASSE1 = 0.489
  Class CLASSE2 = 0.489

Res time:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.617
  Class CLASSE2 at Server Q2 = 0.617
  Class CLASSE1 at Server Q3 = 3.269
  Class CLASSE2 at Server Q3 = 3.269

Mean queue length:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.302
  Class CLASSE2 at Server Q2 = 0.302
  Class CLASSE1 at Server Q3 = 1.600
  Class CLASSE2 at Server Q3 = 1.600
```

MVACCKSW APPROX MVA - LCP:

```
-----------------------------------------------------------------------------------------
 Mean values for Queueing formalism: "rede"
 Algorithm used: Approx MVA - Large Customer Population (LCP)
-----------------------------------------------------------------------------------------

Classes: 2
Servers: 3
Throughput:
  Class CLASSE1 = 0.609
  Class CLASSE2 = 0.609

Res time:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.529
  Class CLASSE2 at Server Q2 = 0.529
  Class CLASSE1 at Server Q3 = 2.556
  Class CLASSE2 at Server Q3 = 2.556

Mean queue length:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.322
  Class CLASSE2 at Server Q2 = 0.322
  Class CLASSE1 at Server Q3 = 1.556
  Class CLASSE2 at Server Q3 = 1.556

Errors for QF 1 with Approx Algorithm nr. 2
  Algorithm description.................: Approx MVA - Large Customer Population (LCP)
  Mean relative throughput error.......: 0.24382
  Mean relative response time error....: 0.20612
  Mean relative mean queue length error: -1.#IND0
  Max relative mean queue length error.: 0.06689
```

MVACCKSW APPROX MVA - PE:

```
-----------------------------------------------------------------------------------------
 Mean values for Queueing formalism: "rede"
 Algorithm used: Approx MVA - Proportional Estimation (PE)
-----------------------------------------------------------------------------------------

Classes: 2
Servers: 3
Throughput:
  Class CLASSE1 = 0.474
  Class CLASSE2 = 0.474

Res time:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.559
  Class CLASSE2 at Server Q2 = 0.559
  Class CLASSE1 at Server Q3 = 3.460
  Class CLASSE2 at Server Q3 = 3.460

Mean queue length:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.265
  Class CLASSE2 at Server Q2 = 0.265
  Class CLASSE1 at Server Q3 = 1.640
  Class CLASSE2 at Server Q3 = 1.640

Errors for QF 1 with Approx Algorithm nr. 3
  Algorithm description.................: Approx MVA - Proportional Estimation (PE)
  Mean relative throughput error.......: 0.03158
  Mean relative response time error....: 0.03429
  Mean relative mean queue length error: -1.#IND0
  Max relative mean queue length error.: 0.12185
```

MVACCKSW APPROX MVA - PAMB:

```
-----------------------------------------------------------------------------------------
 Mean values for Queueing formalism: "rede"
 Algorithm used: Approx MVA - Proportional Approximation Method - Basic (PAMB)
-----------------------------------------------------------------------------------------

Classes: 2
Servers: 3
Throughput:
  Class CLASSE1 = 0.490
  Class CLASSE2 = 0.490

Res time:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.743
  Class CLASSE2 at Server Q2 = 0.743
  Class CLASSE1 at Server Q3 = 3.143
  Class CLASSE2 at Server Q3 = 3.143

Mean queue length:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.571
  Class CLASSE2 at Server Q2 = 0.571
  Class CLASSE1 at Server Q3 = 1.429
  Class CLASSE2 at Server Q3 = 1.429
```

```
Errors for QF 1 with Approx Algorithm nr. 4
  Algorithm description.................: Approx MVA - Proportional Approximation Method - Basic (PAMB)
  Mean relative throughput error.......: 0.00004
  Mean relative response time error....: 0.00004
  Mean relative mean queue length error: -1.#IND0
  Max relative mean queue length error.: 0.89342
```

MVACCKSW APPROX MVA - AQL:

```
-----------------------------------------------------------------------------------------
 Mean values for Queueing formalism: "rede"
 Algorithm used: Approx MVA - Zahorjan-Eager-Sweillam Aggregate Queue Length (AQL)
-----------------------------------------------------------------------------------------

Classes: 2
Servers: 3
Throughput:
  Class CLASSE1 = 0.488
  Class CLASSE2 = 0.488

Res time:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.610
  Class CLASSE2 at Server Q2 = 0.610
  Class CLASSE1 at Server Q3 = 3.288
  Class CLASSE2 at Server Q3 = 3.288

Mean queue length:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.298
  Class CLASSE2 at Server Q2 = 0.298
  Class CLASSE1 at Server Q3 = 1.605
  Class CLASSE2 at Server Q3 = 1.605

Errors for QF 1 with Approx Algorithm nr. 5
  Algorithm description.................: Approx MVA - Zahorjan-Eager-Sweillam Aggregate Queue Length (AQL)
  Mean relative throughput error.......: 0.00301
  Mean relative response time error....: 0.00318
  Mean relative mean queue length error: -1.#IND0
  Max relative mean queue length error.: 0.01395
```

MVACCKSW APPROX MVA – PE II:

```
-----------------------------------------------------------------------------------------
 Mean values for Queueing formalism: "rede"
 Algorithm used: Approx MVA - Proportional Estimation - Varied initial conditions (PE II)
-----------------------------------------------------------------------------------------

Classes: 2
Servers: 3
Throughput:
  Class CLASSE1 = 0.474
  Class CLASSE2 = 0.474

Res time:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.559
  Class CLASSE2 at Server Q2 = 0.559
  Class CLASSE1 at Server Q3 = 3.460
  Class CLASSE2 at Server Q3 = 3.460

Mean queue length:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.265
  Class CLASSE2 at Server Q2 = 0.265
  Class CLASSE1 at Server Q3 = 1.640
  Class CLASSE2 at Server Q3 = 1.640

Errors for QF 1 with Approx Algorithm nr. 6
  Algorithm description.................: Approx MVA - Proportional Estimation - Varied initial conditions (PE II)
  Mean relative throughput error.......: 0.03155
  Mean relative response time error....: 0.03426
  Mean relative mean queue length error: -1.#IND0
  Max relative mean queue length error.: 0.12168
```

MVACCKSW APPROX MVA - QL:

```
-----------------------------------------------------------------------------------------
 Mean values for Queueing formalism: "rede"
 Algorithm used: Approx MVA - Queue Line by Wang & Sevcik (QL)
-----------------------------------------------------------------------------------------
Classes: 2
Servers: 3
Throughput:
  Class CLASSE1 = 0.477
  Class CLASSE2 = 0.477
Res time:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.570
  Class CLASSE2 at Server Q2 = 0.570
  Class CLASSE1 at Server Q3 = 3.420
  Class CLASSE2 at Server Q3 = 3.420
Mean queue length:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.272
  Class CLASSE2 at Server Q2 = 0.272
  Class CLASSE1 at Server Q3 = 1.633
  Class CLASSE2 at Server Q3 = 1.633
Errors for QF 1 with Approx Algorithm nr. 7
  Algorithm description.................: Approx MVA - Queue Line by Wang & Sevcik (QL)
  Mean relative throughput error.......: 0.02484
  Mean relative response time error....: 0.02678
  Mean relative mean queue length error: -1.#IND0
  Max relative mean queue length error.: 0.09878
```

MVACCKSW APPROX MVA - FL:

```
-----------------------------------------------------------------------------------------
 Mean values for Queueing formalism: "rede"
 Algorithm used: Approx MVA - Fraction Line by Wang & Sevcik (FL)
-----------------------------------------------------------------------------------------
Classes: 2
Servers: 3
Throughput:
  Class CLASSE1 = 0.477
  Class CLASSE2 = 0.477
Res time:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.570
  Class CLASSE2 at Server Q2 = 0.570
  Class CLASSE1 at Server Q3 = 3.420
  Class CLASSE2 at Server Q3 = 3.420
Mean queue length:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.272
  Class CLASSE2 at Server Q2 = 0.272
  Class CLASSE1 at Server Q3 = 1.633
  Class CLASSE2 at Server Q3 = 1.633
Errors for QF 1 with Approx Algorithm nr. 8
  Algorithm description.................: Approx MVA - Fraction Line by Wang & Sevcik (FL)
  Mean relative throughput error.......: 0.02484
  Mean relative response time error....: 0.02678
  Mean relative mean queue length error: -1.#IND0
  Max relative mean queue length error.: 0.09878
```

MVACCKSW APPROX MVA – AQL QL:

```
-----------------------------------------------------------------------------------------
 Mean values for Queueing formalism: "rede"
 Algorithm used: Approx MVA - AQL + Queue Line (AQL QL)
-----------------------------------------------------------------------------------------
Classes: 2
Servers: 3
Throughput:
  Class CLASSE1 = 0.481
  Class CLASSE2 = 0.392
Res time:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.598
  Class CLASSE2 at Server Q2 = 0.633
  Class CLASSE1 at Server Q3 = 3.362
  Class CLASSE2 at Server Q3 = 4.266
Mean queue length:
  Class CLASSE1 at Server Q1 = 0.000
  Class CLASSE2 at Server Q1 = 0.000
  Class CLASSE1 at Server Q2 = 0.287
  Class CLASSE2 at Server Q2 = 0.248
  Class CLASSE1 at Server Q3 = 1.616
  Class CLASSE2 at Server Q3 = 1.673
Errors for QF 1 with Approx Algorithm nr. 9
  Algorithm description.................: Approx MVA - AQL + Queue Line (AQL QL)
  Mean relative throughput error.......: 0.10826
  Mean relative response time error....: 0.13994
  Mean relative mean queue length error: -1.#IND0
  Max relative mean queue length error.: 0.17703
```

12.2.3 Comparazione dei risultati numerici ottenuti:

CLOSEDQN	Throughput	Population	Utilization	Res. Time
CLASSE1	0.474029			4.219149
CLASSE2	0.474029			4.219149
Q1/CLASSE1				0.200000
Q2/CLASSE1				0.559087
Q3/CLASSE1				3.460062
Q1/CLASSE2				0.200000
Q2/CLASSE2				0.559087
Q3/CLASSE2				3.460062
Q1				
Q2				
Q3				

MQNA1	Throughput	Population	Utilization	Res. Time
CLASSE1				
CLASSE2				
Q1/CLASSE1	0.951952	0.285285	0.190390	0.299685
Q2/CLASSE1	0.475976	0.285285	0.190390	0.599369
Q3/CLASSE1	0.475976	1.429429	0.475976	3.003155
Q1/CLASSE2	0.951952	0.285285	0.190390	0.299685
Q2/CLASSE2	0.475976	0.285285	0.190390	0.599369
Q3/CLASSE2	0.475976	1.429429	0.475976	3.003155
Q1	1.903904	0.570571	0.380781	0.299685
Q2	0.951952	0.570571	0.380781	0.599369
Q3	0.951952	2.858859	0.951952	3.003155

SHARPE	Throughput	Population	Utilization	Res. Time
CLASSE1				
CLASSE2				
Q1/CLASSE1	0.971580	0.194320	0.194320	0.200000
Q2/CLASSE1	0.485790	0.295300	0.194320	0.607870
Q3/CLASSE1	0.485790	1.510400	0.485790	3.109100
Q1/CLASSE2	0.971580	0.194320	0.194320	0.200000
Q2/CLASSE2	0.485790	0.295300	0.194320	0.607870
Q3/CLASSE2	0.485790	1.510400	0.485790	3.109100
Q1				
Q2				
Q3				

PDQ EXACT	Throughput	Population	Utilization	Res. Time
CLASSE1	0.489500			3.885900
CLASSE2	0.489500			3.885900
Q1/CLASSE1				
Q2/CLASSE1	0.489500	0.301800	0.195796	0.616600
Q3/CLASSE1	0.489500	1.600300	0.489490	3.269300
Q1/CLASSE2				
Q2/CLASSE2	0.489500	0.301800	0.195796	0.616600
Q3/CLASSE2	0.489500	1.600300	0.489490	3.269300
Q1				
Q2				
Q3				

PDQ APPROX	Throughput	Population	Utilization	Res. Time
CLASSE1	0.474100			4.018800
CLASSE2	0.474100			4.018800
Q1/CLASSE1				
Q2/CLASSE1	0.474100	0.265100	0.189627	0.559300
Q3/CLASSE1	0.474100	1.640000	0.474068	3.459500
Q1/CLASSE2				
Q2/CLASSE2	0.474100	0.265100	0.189627	0.559300
Q3/CLASSE2	0.474100	1.640000	0.474068	3.459500
Q1				
Q2				
Q3				

PMVA MVA	Throughput	Population	Utilization	Res. Time
CLASSE1				
CLASSE2				
Q1/CLASSE1	0.971600	0.194300		0.200000
Q2/CLASSE1	0.485800	0.295300	0.194300	0.607900
Q3/CLASSE1	0.485800	1.510400	0.485800	3.109100
Q1/CLASSE2	0.971600	0.194300		0.200000
Q2/CLASSE2	0.485800	0.295300	0.194300	0.607900
Q3/CLASSE2	0.485800	1.510400	0.485800	3.109100
Q1	1.943200	0.388600		0.200000
Q2	0.971600	0.590600	0.388600	0.607900
Q3	0.971600	3.020800	0.971600	3.109100

PMVA SCHWEITZER	Throughput	Population	Utilization	Res. Time
CLASSE1				
CLASSE2				
Q1/CLASSE1	0.933100	0.186600		0.200000
Q2/CLASSE1	0.466500	0.259200	0.186600	0.555500
Q3/CLASSE1	0.466500	1.554200	0.466500	3.331300
Q1/CLASSE2	0.933100	0.186600		0.200000
Q2/CLASSE2	0.466500	0.259200	0.186600	0.555500
Q3/CLASSE2	0.466500	1.554200	0.466500	3.331300
Q1	1.866200	0.373200		0.200000
Q2	0.933100	0.518300	0.373200	0.555500
Q3	0.933100	3.108400	0.933100	3.331300

PMVA ASYMP	Throughput	Population	Utilization	Res. Time
CLASSE1				
CLASSE2				
Q1/CLASSE1	0.764900	0.153000		0.200000
Q2/CLASSE1	0.382400	0.220400	0.153000	0.576300
Q3/CLASSE1	0.382400	1.626600	0.382400	4.253200
Q1/CLASSE2	0.764900	0.153000		0.200000
Q2/CLASSE2	0.382400	0.220400	0.153000	0.576300
Q3/CLASSE2	0.382400	1.626600	0.382400	4.253200
Q1	1.529800	0.306000		0.200000
Q2	0.764900	0.440800	0.306000	0.576300
Q3	0.764900	3.253200	0.764900	4.253200

PMVA LIN	Throughput	Population	Utilization	Res. Time
CLASSE1				
CLASSE2				
Q1/CLASSE1	0.968700	0.193700		0.200000
Q2/CLASSE1	0.484300	0.292000	0.193700	0.602800
Q3/CLASSE1	0.484300	1.514300	0.484300	3.126500
Q1/CLASSE2	0.968700	0.193700		0.200000
Q2/CLASSE2	0.484300	0.292000	0.193700	0.602800
Q3/CLASSE2	0.484300	1.514300	0.484300	3.126500
Q1	1.937400	0.387500		0.200000
Q2	0.968700	0.583900	0.387500	0.602800
Q3	0.968700	3.028600	0.968700	3.126500

MVACCKSW EXACT	Throughput	Population	Utilization	Res. Time
CLASSE1	0.489000			
CLASSE2	0.489000			
Q1/CLASSE1				
Q2/CLASSE1		0.302000		0.617000
Q3/CLASSE1		1.600000		3.269000
Q1/CLASSE2				
Q2/CLASSE2		0.302000		0.617000
Q3/CLASSE2		1.600000		3.269000
Q1				
Q2				
Q3				

MVACCKSW LCP	Throughput	Population	Utilization	Res. Time
CLASSE1	0.609000			
CLASSE2	0.609000			
Q1/CLASSE1				
Q2/CLASSE1		0.322000		0.529000
Q3/CLASSE1		1.556000		2.556000
Q1/CLASSE2				
Q2/CLASSE2		0.322000		0.529000
Q3/CLASSE2		1.556000		2.556000
Q1				
Q2				
Q3				

MVACCKSW PE	Throughput	Population	Utilization	Res. Time
CLASSE1	0.474000			
CLASSE2	0.474000			
Q1/CLASSE1				
Q2/CLASSE1		0.265000		0.559000
Q3/CLASSE1		1.640000		3.460000
Q1/CLASSE2				
Q2/CLASSE2		0.265000		
Q3/CLASSE2		1.640000		
Q1				
Q2				
Q3				

MVACCKSW PAMB	Throughput	Population	Utilization	Res. Time
CLASSE1	0.490000			
CLASSE2	0.490000			
Q1/CLASSE1				
Q2/CLASSE1		0.571000		0.743000
Q3/CLASSE1		1.429000		3.143000
Q1/CLASSE2				
Q2/CLASSE2				
Q3/CLASSE2				
Q1				
Q2				
Q3				

MVACCKSW AQL	Throughput	Population	Utilization	Res. Time
CLASSE1	0.488000			
CLASSE2	0.488000			
Q1/CLASSE1				
Q2/CLASSE1		0.298000		0.610000
Q3/CLASSE1		1.605000		3.288000
Q1/CLASSE2				
Q2/CLASSE2		0.298000		0.610000
Q3/CLASSE2		1.605000		3.288000
Q1				
Q2				
Q3				

MVACCKSW PE II	Throughput	Population	Utilization	Res. Time
CLASSE1	0.474000			
CLASSE2	0.474000			
Q1/CLASSE1				
Q2/CLASSE1		0.265000		0.559000
Q3/CLASSE1		1.640000		3.460000
Q1/CLASSE2				
Q2/CLASSE2		0.265000		0.559000
Q3/CLASSE2		1.640000		3.460000
Q1				
Q2				
Q3				

MVACCKSW QL	Throughput	Population	Utilization	Res. Time
CLASSE1	0.477000			
CLASSE2	0.477000			
Q1/CLASSE1				
Q2/CLASSE1		0.272000		0.570000
Q3/CLASSE1		1.633000		3.420000
Q1/CLASSE2				
Q2/CLASSE2		0.272000		0.570000
Q3/CLASSE2		1.633000		3.420000
Q1				
Q2				
Q3				

MVACCKSW FL	Throughput	Population	Utilization	Res. Time
CLASSE1	0.477000			
CLASSE2	0.477000			
Q1/CLASSE1				
Q2/CLASSE1		0.272000		0.570000
Q3/CLASSE1		1.633000		3.420000
Q1/CLASSE2				
Q2/CLASSE2		0.272000		0.570000
Q3/CLASSE2		1.633000		3.420000
Q1				
Q2				
Q3				

MVACCKSW AQL QL	Throughput	Population	Utilization	Res. Time
CLASSE1	0.481000			
CLASSE2	0.392000			
Q1/CLASSE1				
Q2/CLASSE1		0.287000		0.598000
Q3/CLASSE1		1.616000		3.362000
Q1/CLASSE2				
Q2/CLASSE2		0.248000		0.633000
Q3/CLASSE2		1.673000		4.266000
Q1				
Q2				
Q3				

PEPSY AMMVA	Throughput	Population	Utilization	Res. Time
CLASSE1	0.972000	2.000000		2.058000
CLASSE2	0.972000	2.000000		2.058000
Q1/CLASSE1	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE1	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE1	0.486000	1.510000	0.486000	3.109000
Q1/CLASSE2	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE2	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE2	0.486000	1.510000	0.486000	3.109000
Q1	1.943000	0.389000	0.000000	
Q2	0.972000	0.591000	0.389000	
Q3	0.972000	3.021000	0.972000	

PEPSY BOL_AKY	Throughput	Population	Utilization	Res. Time
CLASSE1	0.970000	2.000000		2.061000
CLASSE2	0.970000	2.000000		2.061000
Q1/CLASSE1	0.970000	0.194000	0.000000	0.200000
Q2/CLASSE1	0.485000	0.295000	0.194000	0.607000
Q3/CLASSE1	0.485000	1.511000	0.485000	3.115000
Q1/CLASSE2	0.970000	0.194000	0.000000	0.200000
Q2/CLASSE2	0.485000	0.295000	0.194000	0.607000
Q3/CLASSE2	0.485000	1.511000	0.485000	3.115000
Q1	1.941000	0.388000	0.000000	
Q2	0.970000	0.589000	0.388000	
Q3	0.970000	3.023000	0.970000	

PEPSY CMVA	Throughput	Population	Utilization	Response Time
CLASSE1	0.972000	2.000000		2.058000
CLASSE2	0.972000	2.000000		2.058000
Q1/CLASSE1	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE1	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE1	0.486000	1.510000	0.486000	3.109000
Q1/CLASSE2	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE2	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE2	0.486000	1.510000	0.486000	3.109000
Q1	1.943000	0.389000	0.000000	
Q2	0.972000	0.591000	0.389000	
Q3	0.972000	3.021000	0.972000	

PEPSY MMVA	Throughput	Population	Utilization	Res. Time
CLASSE1	0.972000	2.000000		2.058000
CLASSE2	0.972000	2.000000		2.058000
Q1/CLASSE1	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE1	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE1	0.486000	1.510000	0.486000	3.109000
Q1/CLASSE2	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE2	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE2	0.486000	1.510000	0.486000	3.109000
Q1	1.943000	0.389000	0.000000	
Q2	0.972000	0.591000	0.389000	
Q3	0.972000	3.021000	0.972000	

PEPSY MULTISUM	Throughput	Population	Utilization	Res. Time
CLASSE1	0.933000	2.000000		2.144000
CLASSE2	0.933000	2.000000		2.144000
Q1/CLASSE1	0.933000	0.187000	0.000000	0.200000
Q2/CLASSE1	0.467000	0.259000	0.187000	0.556000
Q3/CLASSE1	0.467000	1.554000	0.467000	3.331000
Q1/CLASSE2	0.933000	0.187000	0.000000	0.200000
Q2/CLASSE2	0.467000	0.259000	0.187000	0.556000
Q3/CLASSE2	0.467000	1.554000	0.467000	3.331000
Q1	1.866000	0.374000	0.000000	
Q2	0.934000	0.518000	0.374000	
Q3	0.934000	3.108000	0.934000	

PEPSY PM_2	Throughput	Population	Utilization	Res. Time
CLASSE1	0.970000	2.000000		2.061000
CLASSE2	0.970000	2.000000		2.061000
Q1/CLASSE1	0.970000	0.194000	0.000000	0.200000
Q2/CLASSE1	0.485000	0.295000	0.194000	0.607000
Q3/CLASSE1	0.485000	1.511000	0.485000	3.115000
Q1/CLASSE2	0.970000	0.194000	0.000000	0.200000
Q2/CLASSE2	0.485000	0.295000	0.194000	0.607000
Q3/CLASSE2	0.485000	1.511000	0.485000	3.115000
Q1	1.941000	0.388000	0.000000	
Q2	0.970000	0.589000	0.388000	
Q3	0.970000	3.023000	0.970000	

PEPSY PRIOMVA2C	Throughput	Population	Utilization	Res. Time
CLASSE1	0.972000	2.000000		2.059000
CLASSE2	0.972000	2.000000		2.059000
Q1/CLASSE1	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE1	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE1	0.486000	1.510000	0.486000	3.109000
Q1/CLASSE2	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE2	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE2	0.486000	1.510000	0.486000	3.109000
Q1	1.943000	0.389000	0.000000	
Q2	0.972000	0.591000	0.389000	
Q3	0.972000	3.021000	0.972000	

PEPSY PRIOMVA2M	Throughput	Population	Utilization	Res. Time
CLASSE1	0.972000	2.000000		2.059000
CLASSE2	0.972000	2.000000		2.059000
Q1/CLASSE1	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE1	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE1	0.486000	1.510000	0.486000	3.109000
Q1/CLASSE2	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE2	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE2	0.486000	1.510000	0.486000	3.109000
Q1	1.943000	0.389000	0.000000	
Q2	0.972000	0.591000	0.389000	
Q3	0.972000	3.021000	0.972000	

PEPSY RECAL	Throughput	Population	Utilization	Res. Time
CLASSE1	0.972000	2.000000		2.059000
CLASSE2	0.972000	2.000000		2.059000
Q1/CLASSE1	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE1	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE1	0.486000	1.510000	0.486000	3.109000
Q1/CLASSE2	0.972000	0.194000	0.000000	0.200000
Q2/CLASSE2	0.486000	0.295000	0.194000	0.608000
Q3/CLASSE2	0.486000	1.510000	0.486000	3.109000
Q1	1.943000	0.389000	0.000000	
Q2	0.972000	0.591000	0.389000	
Q3	0.972000	3.021000	0.972000	

12.2.4 Grafici comparativi:

Throughput per Classe di clienti:

Throughput	CLOSEDQN	PDQ EX	PDQ APP	CCK EXACT	CCK LCP	CCK PE	CCK PAMB	CCK AQL	CCK PE II	CCK QL	CCK FL	CCK AQLQL	P-AMMVA	P-BOL_AKY	P-CMVA	P-MMVA	P-MULTISUM	P-PM_2	P-PRIOMVA2C	P-PRIOMVA2M	P-RECAL
CLASSE1	0.474029	0.489500	0.474100	0.489000	0.609000	0.474000	0.490000	0.488000	0.474000	0.477000	0.477000	0.481000	0.972000	0.970000	0.972000	0.972000	0.933000	0.970000	0.972000	0.972000	0.972000
CLASSE2	0.474029	0.489500	0.474100	0.489000	0.609000	0.474000	0.490000	0.488000	0.474000	0.477000	0.477000	0.392000	0.972000	0.970000	0.972000	0.972000	0.933000	0.970000	0.972000	0.972000	0.972000

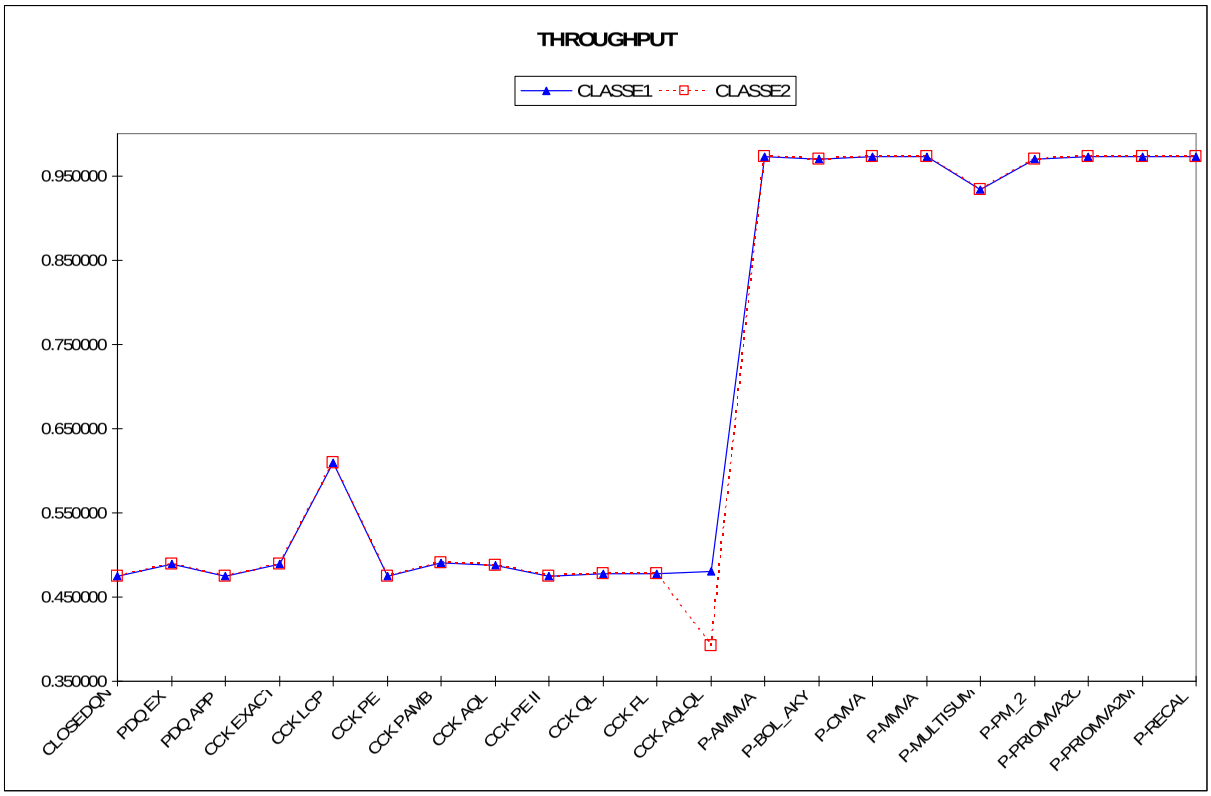


Figura 12.3 - Throughput per classe fornito da alcuni tool

Utilizzazione per Nodo:

Utilization per Nodo	Q1	Q2	Q3
MQNA1	0.380781	0.380781	0.951952
PMVA MVA		0.388600	0.971600
PMVA SCHWEITZER		0.373200	0.933100
PMVA ASYMP		0.306000	0.764900
PMVA LIN		0.387500	0.968700
PEPSY AMMVA	0.000000	0.389000	0.972000
PEPSY BOL_AKY	0.000000	0.388000	0.970000
PEPSY CMVA	0.000000	0.389000	0.972000
PEPSY MMVA	0.000000	0.389000	0.972000
PEPSY MULTISUM	0.000000	0.374000	0.934000
PEPSY PM_2	0.000000	0.388000	0.970000
PEPSY PRIOMVA2C	0.000000	0.389000	0.972000
PEPSY PRIOMVA2M	0.000000	0.389000	0.972000
PEPSY RECAL	0.000000	0.389000	0.972000

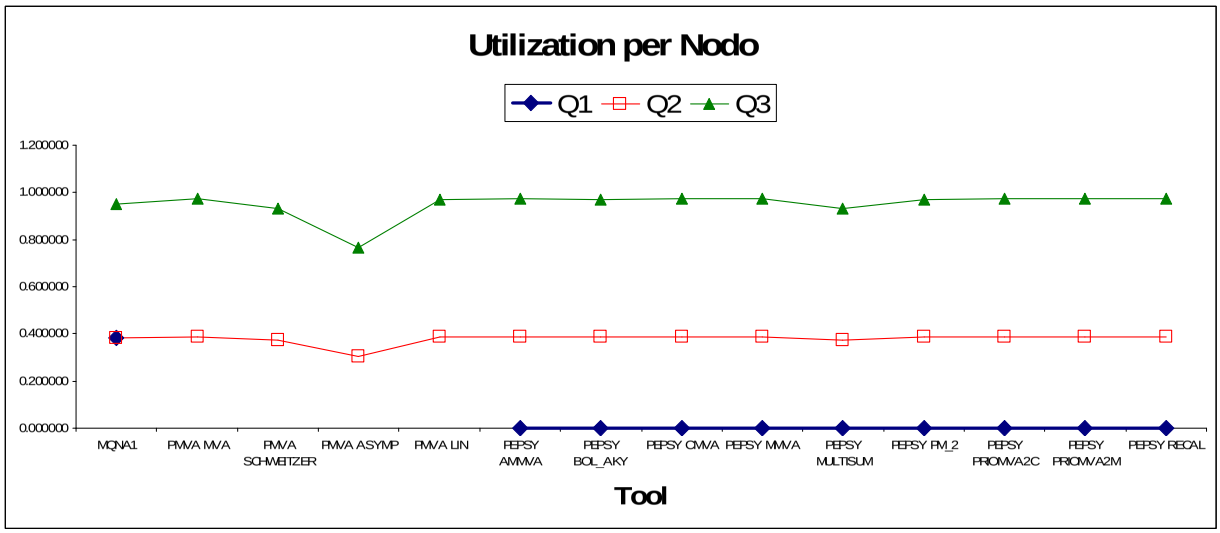
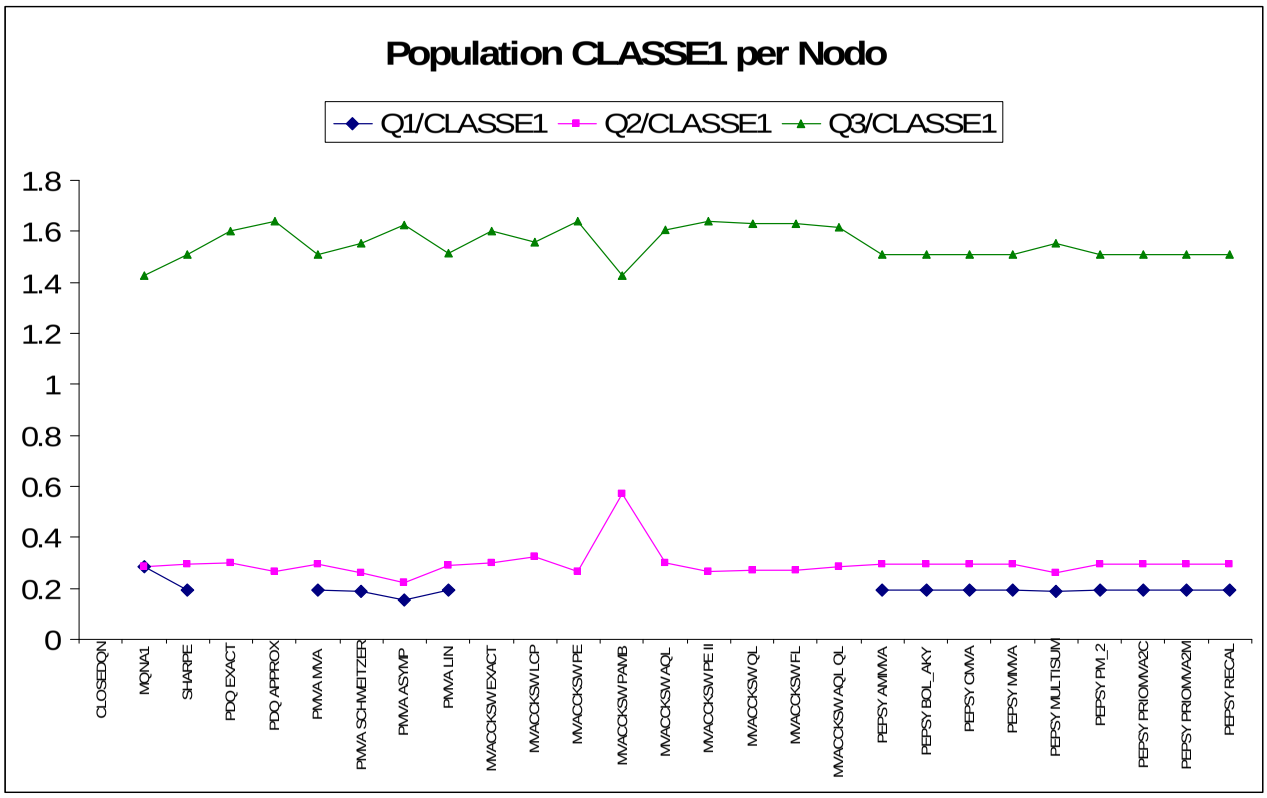
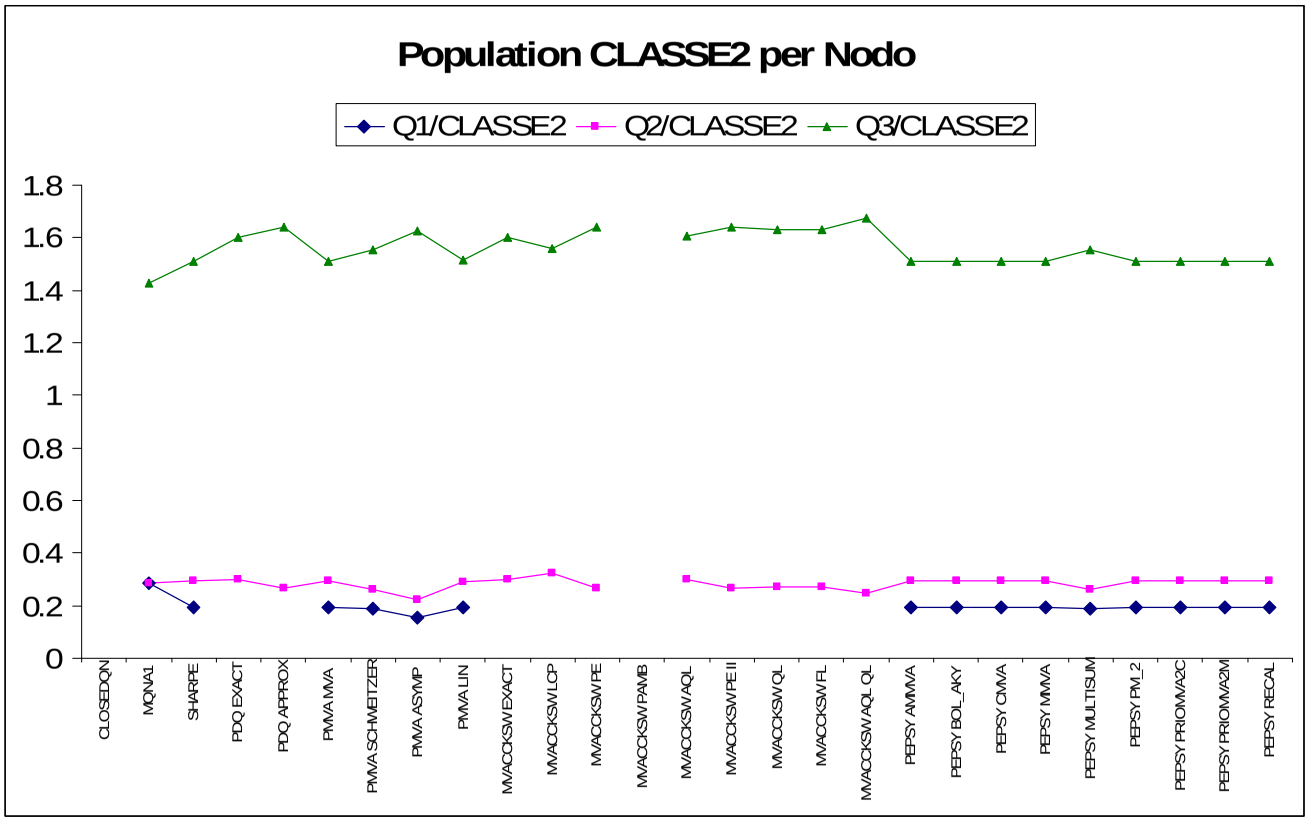


Figura 12.4 - Utilization per Nodo fornita da alcuni tool

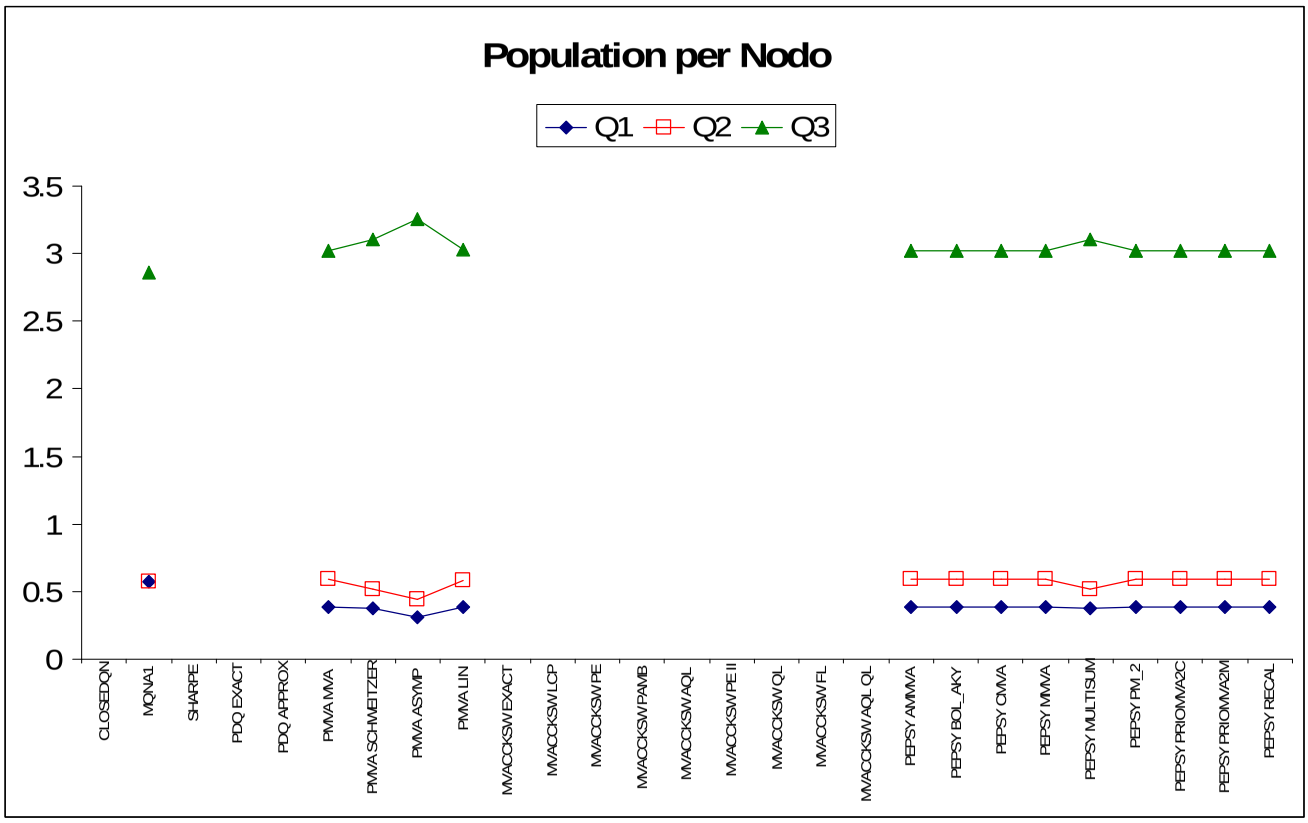
Popolazione per Nodo della Classe1:



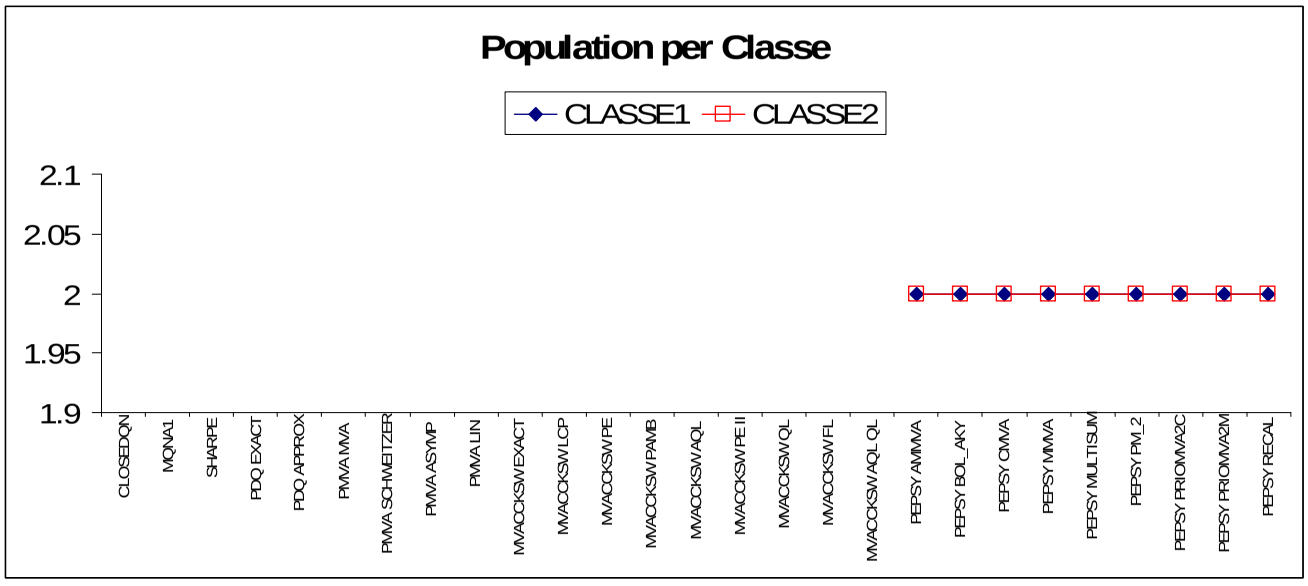
Popolazione per Nodo della Classe2:



Popolazione per Nodo:



Popolazione per Classe:



13 Conclusioni e sviluppi futuri

In questa tesi ci siamo occupati dello sviluppo di un web service per la risoluzione di modelli a reti di code mediante l'utilizzo di tool esterni. Prendendo spunto dal lavoro fatto da Connie U. Smith, che aveva integrato solamente il tool QNAP2, il nostro obiettivo era verificare la possibilità di integrare un numero maggiore di tool. La ricerca di tool adeguati allo scopo è risultata molto più ardua del previsto. I tool più interessanti disponibili in rete sono per lo più a pagamento. Ad esempio non ci è stato possibile riuscire ad avere una copia di QNAP2, il tool usato da C. U. Smith, in quanto ora è diventato un tool commerciale di nome Modline. Questo è un vero peccato in quanto quasi tutti gli altri tool fanno, chi più chi meno, riferimento a QNAP2 come termine di paragone, e ci sarebbe piaciuto poter confrontare i risultati ottenibili da QNAP2 con quelli degli altri tool. Altri tool liberamente scaricabili da internet sono invece piuttosto datati e sono quindi disponibili per piattaforme ormai obsolete e sistemi operativi non più disponibili per le piattaforme attuali. Per alcuni dei tool che siamo riusciti ad integrare è stato necessario apportare profonde modifiche al sorgente (MVAQFP, OPENQN, CLOSEDQN) per rendere il formalismo di descrizione della rete di code più significativo e più facile la conversione da PMIF 2.0 a linguaggio del tool. Per altri, disponibili come librerie e non come veri e propri tool, abbiamo addirittura dovuto inventarci un tool ed un formalismo che fosse integrabile con il webservice (PDQ). Ed infine, per altri ancora, è stato sufficiente riadattare il codice per il funzionamento batch (MQNA1) e ricompilarli. Sicuramente, anche tra i tool scartati, ce ne sarà qualcuno che, con un po' di lavoro aggiuntivo e qualche modifica al sorgente, potrebbe risultare adatto ad essere integrato nel webservice. Ma per qualcun'altro invece non c'è alcuna possibilità di integrazione, come ad esempio tutti quei tool che sono prettamente grafici e non offrono interfacce testuali o meccanismi di integrazione come OLE, DDE, CORBA o SOAP. Tra i tool non integrati, quelli più interessanti sono sicuramente QNAT, JMT, Mosel e Möbius. Versioni future ed estensioni della presente tesi potrebbero indagare più a fondo la natura di questi tool, e magari contattare gli autori per verificare la possibilità di interfacciare questi tool (o quanto meno le loro librerie interne di risoluzione) con il webservice. In particolare il tool MOSEL rappresenta una versione "locale" di quello che stiamo cercando di fare in questa tesi. Il concetto su cui si basa MOSEL è quello di avere un linguaggio comune per la descrizione del modello, e utilizzare poi il tool più adatto per la sua risoluzione (SHARPE, PEPSY, TimeNET, SPNP, tool di simulazione o MOSEL-Tool), che è esattamente quello che abbiamo realizzato in questa tesi con PMIF 2.0 ed un web service.
Il web service ed il client Windows™ sono basati sulla versione 2.0 di PMIF. Come già detto, questo formato di descrizione dei modelli a rete di code è carente sotto alcuni aspetti ed andrebbe quindi esteso per includere quelle feature che alcuni tool hanno ma che non sono previste da PMIF. Primo tra tutti la possibilità, che a volte diventa una necessità, di specificare la funzione di distribuzione degli arrivi ai centri di servizio. Oppure la possibilità di specificare centri di servizio Load Dependent, utili a modellare una più ampia varietà di casi reali.
Un'altra estensione alla presente tesi potrebbe essere quella di prevedere un meccanismo di autenticazione per l'accesso al webservice. Questa feature consentirebbe agli autori di algoritmi e tool di fornire il servizio di risoluzione a pagamento o comunque di tenere traccia degli accessi e di capire chi usa il webservice. L'idea sarebbe quella di trasformare questo webservice in un prodotto commerciabile, dove il codice sorgente del webservice potrebbe essere distribuito sotto licenza GNU General Public License o Creative Commons Public License, mentre il client Windows potrebbe diventare un prodotto a pagamento. Si potrebbe progettare un repository ad accesso pubblico dove registrare tutti i webservice ed i tool che essi integrano (qualcosa di simile ad UDDI o perché no un registro UDDI già esistente). Il client potrebbe accedere a questo repository e presentare all'utente una lista di webservice a cui collegarsi. Questo fornirebbe all'utente finale un'ampia scelta di tool ed algoritmi per l'analisi dei suoi modelli e consentirebbe invece ai progettisti di tool di esporre i loro algoritmi tramite webservice con autenticazione degli accessi e di farsi pagare solo il reale utilizzo.
Un'ulteriore e necessaria modifica al webservice dovrebbe essere quella di studiare un meccanismo di protezione per la verifica del modello e dei parametri inviati dal client al server e quindi ai tool di

risoluzione. Alcuni tool necessitano infatti di parametri sulla linea di comando (PMVA e PEPSY) per specificare, ad esempio, quale algoritmo usare per la risoluzione del modello. Un utente malintenzionato potrebbe invocare il webservice inserendo, come parametro, invece del nome dell'algoritmo, del codice malevolo che a quel punto sarebbe eseguito sulla macchina server con gli stessi privilegi del server HTTP che ospita il webservice. Nel nostro caso, trattandosi di un esempio accademico, per non appesantire il codice, non è stato previsto alcun controllo sui parametri passati dal client al server sotto forma di messaggi SOAP. Versioni future di questo webservice dovrebbero quindi prevedere dei controlli di sicurezza sui messaggi scambiati tra client e server e, soprattutto, su cosa viene passato ai tool esterni sulla riga di comando, pena una grave falla di sicurezza su qualsiasi server web esso venga installato.

Finito di stampare: domenica 25 marzo 2007

14 Appendici

Appendice A: Metodi per la risoluzione di modelli a Rete di Code

Dato un modello a rete di code di un sistema, i metodi analitici si propongono di descriverne il comportamento attraverso relazioni funzionali che legano tra loro i parametri di ingresso del modello ricavando le misure di prestazione desiderate. La potenza dei metodi analitici sta nel fatto che le formule che esprimono gli indici di prestazione desiderati forniscono spesso informazioni aggiuntive sul comportamento del sistema permettendo all'analista di verificare, direttamente, quali variabili di ingresso sono importanti per determinare un dato indice di prestazione e l'andamento che questi indici possono avere.

Spesso si ha a che fare con modelli la cui complessità è tale che l'analisi matematica (probabilistica) del loro comportamento diventa troppo difficile da eseguire. La stima dei parametri di ingresso e delle caratteristiche operative del sistema reale sono, in questi casi, corrette e semplificate per rendere il modello trattabile matematicamente. Nella pratica si riscontra che, in molti casi, queste semplificazioni non producono delle approssimazioni dei risultati così drastiche come ci si potrebbe aspettare. Questo risultato sorprendente è dovuto al fatto che molti degli indici di comune interesse sono in realtà sensibili solo alla media delle distribuzioni che entrano nella descrizione del modello e quindi questi sono spesso gli unici valori che l'analista deve introdurre nel modello per descrivere con sufficiente accuratezza la realtà modellata.

L'**analisi operazionale** dei modelli a rete di code parte da questa constatazione pragmatica, per sviluppare una serie di relazioni, che devono legare le quantità misurate su un sistema quando questo si sia trovato, durante il periodo di osservazione, in equilibrio operazionale, ed abbia obbedito ad alcune ipotesi che, almeno in linea di principio, l'analista potrebbe sempre verificare senza dover fare uso di proprietà valide solo al limite (insiemi di dimensione infinita, periodi di osservazione di lunghezza infinita, ecc). In tutti quei casi in cui le misure eseguite sul sistema reale non permettono di rappresentare con ragionevole approssimazione il sistema stesso, con uno dei modelli analizzati operazionalmente, l'analista dovrà ricorrere a tecniche probabilistiche di analisi, che sono più potenti ma richiedono anche una conoscenza della teoria delle probabilità e dei processi stocastici molto più approfondita.

La valutazione di un sistema può essere effettuata anche attraverso l'analisi dei limiti superiore e inferiore delle prestazioni che questo sistema può fornire. Le semplici relazioni dell'analisi operazionale, valide sotto assunzioni generali, permettono di derivare limiti superiori ed inferiori alle prestazioni del sistema in funzione del carico. Si parla di **analisi asintotica** in quanto si va a valutare il sistema in condizioni estreme di calcolo. Questo approccio è caratterizzato dalla semplicità di calcolo, che permette di ottenere rapidamente una prima approssimazione del comportamento del sistema. Per semplicità ci limiteremo a sistemi con una sola classe di clienti. L'analisi dei limiti sulle prestazioni di sistemi con più classi di clienti è possibile, ma vengono a mancare le caratteristiche di semplicità e rapidità che rendono conveniente l'applicazione di questa tecnica. È possibile valutare sistemi in fase di progetto, non completamente definiti, così come sistemi in fase di dimensionamento, confrontando i risultati derivanti da configurazioni alternative del sistema. Con l'analisi asintotica possiamo individuare fattori critici del sistema. In particolare possiamo identificare i colli di bottiglia (*bottleneck*) e valutarne quantitativamente l'impatto sulle prestazioni del sistema. Il procedimento di analisi e rimozione dei colli di bottiglia è iterativo, in quanto la rimozione di un collo di bottiglia può portare alla identificazione di altri colli di bottiglia, detti secondari, che si manifestano solo dopo aver eliminato i primi. Quando tutti i colli di bottiglia sono stati eliminati abbiamo un sistema bilanciato, maggiori dettagli sul processo di rimozione dei colli di bottiglia possono essere trovati in [10] e [8].

Lo studio asintotico di configurazioni alternative può essere usato per stimare le performance di sistemi esistenti o upgrade degli stessi. Una semplice condizione per definire un insieme di configurazioni alternative da analizzare, è l'identificazione della o delle risorse critiche; valutando poi le prestazioni del sistema in funzione di queste configurazioni.

A.1 Analisi Operazionale

L'analisi operazionale dei modelli a Rete di Code costituisce uno strumento semplice e generale di analisi dei modelli, applicabile non soltanto a reti di code. L'analisi operazionale si basa sull'osservazione di un sistema in un intervallo di tempo finito da cui deriva alcune semplici relazioni e indici di prestazione.
Consideriamo un sistema aperto generico:

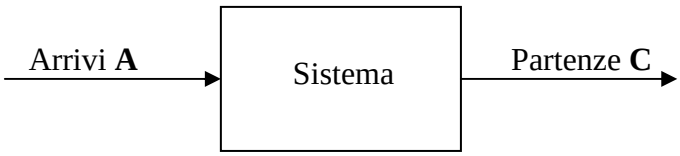


Figura A.1 - Sistema Aperto Generico

Supponiamo di osservare questo sistema per un intervallo di tempo $[0,t]$. Si possono misurare alcune grandezze caratteristiche:

t	tempo di osservazione;
$A(t)$	numero totale di arrivi nel sistema in $[0,t]$;
$C(t)$	numero totale di utenti che hanno completato il servizio al tempo $[0,t]$;
$B(t)$	busy time o tempo di occupazione del sistema in $[0,t]$;

da cui si ricavano:

$\lambda(t) = A(t) / t$	tasso di arrivo al sistema in $[0,t]$;
$X(t) = C(t) / t$	throughput del sistema in $[0,t]$;
$U(t) = B(t) / t$	utilizzazione del sistema in $[0,t]$;
$S(t) = B(t) / C(t)$	servizio medio per utente in $[0,t]$;

dove $A(t), C(t) \in \mathbf{N}$; $\lambda(t), X(t), S(t) \in \mathbf{R}_0^+$; $B(t) \in [0,t]$; $U(t) \in [0,1]$ e va tenuto presente che $S(t)$ può essere affetta da errori dovuti alle condizioni iniziali e finali del sistema.

É ovvio che le quantità sopra elencate, sono variabili, il cui valore può cambiare da un periodo di osservazione a l'altro. Tuttavia vale sempre la seguente:

Legge di Utilizzazione (LU): l'utilizzazione di una risorsa è uguale al prodotto del throughput di detta risorsa e il tempo medio del singolo servizio:

$$U(t) = \frac{B(t)}{t} = \frac{B(t)}{t}1 = \frac{B(t)}{t}\frac{C(t)}{C(t)} = \frac{C(t)}{t}\frac{B(t)}{C(t)} \qquad \text{ma } \frac{C(t)}{t} = X(t) \text{ e } \frac{B(t)}{C(t)} = S(t)$$

da cui si ricava la legge di utilizzazione:

$$U(t) = X(t)S(t)$$

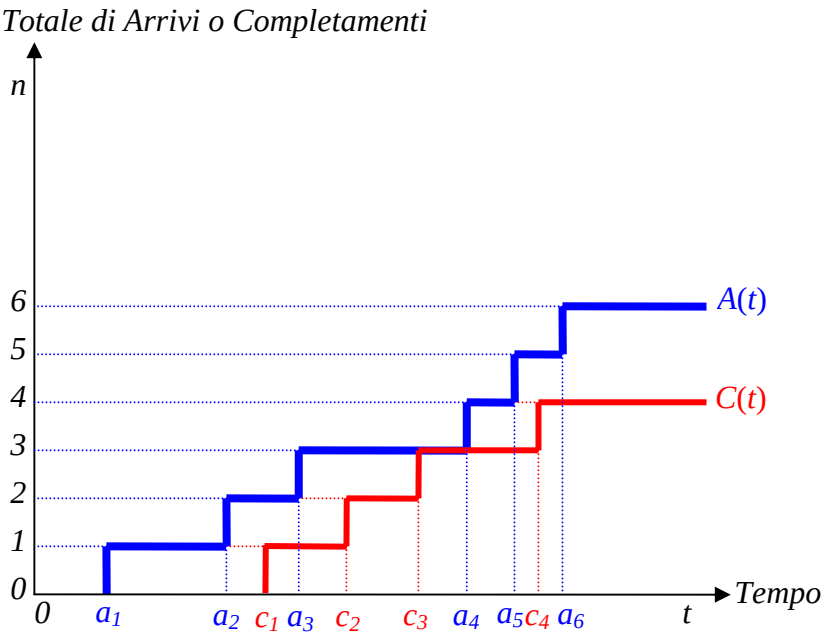
o in forma compatta:

$$U = X\,S$$

Ad esempio, si consideri un hard disk per computer che debba servire una media di X=40*r/s* (richieste al secondo), ciascuna delle quali richieda un tempo di servizio pari S=2,5*msec* (millisecondi). L'utilizzo del disco risulta:

$$U = X\,S = 40 \times \frac{22{,}5}{1000} = 0{,}9 = 90\%\,.$$

Consideriamo ora il numero di utenti presenti nel sistema al tempo t, dato da $A(t) - C(t)$, l'evoluzione temporale di questa quantità evolve nel tempo come nella figura seguente:



Istante t	$A(t)$	$C(t)$	$A(t)$-$C(t)$
0	0	0	0
a_1	1	0	1
a_2	2	0	2
c_1	2	1	1
a_3	3	1	2
c_2	3	2	1
c_3	3	3	0
a_4	4	3	1
a_5	5	3	2
c_4	5	4	1
a_6	6	4	2

Figura A.2 - Andamento delle funzioni *A*(t) e *C*(t)

È da notare che solo nel caso di singolo servente e disciplina di servizio FIFO/FCFS gli utenti escono nello stesso ordine in cui entrano. In caso contrario, ad esempio, l'utente che arriva all'istante a_2 se richiede un minor tempo di servizio rispetto all'utente che arriva al tempo a_1, potrebbe uscire prima.

Definiamo: **lavoro totale accumulato in [*0,t*]**

$$W(t) = \int_0^t \big(A(t) - C(t)\big)\, d\tau$$

Da cui si ricava:

$$N(t) = \frac{W(t)}{t} \quad \text{numero medio di utenti in } [0,t]$$

$$R(t) = \frac{W(t)}{C(t)} \quad \text{tempo medio di risposta in } [0,t]$$

dove $N(t)$, $R(t) \in \mathbf{R_o}^+$.

Per costruzione e definizione otteniamo la seguente legge di Little, da J.D.C. Little che per primo nel 1961 [47] ne diede una formulazione rigorosa:

Legge di Little (LL): lega numero medio di utenti, throughput e tempo medio di risposta in un intervallo di tempo:

$$N(t) = X(t)R(t)$$

La legge di Little dice che il numero medio di richieste/utenti nel sistema è uguale al prodotto del throughput del sistema e il tempo medio speso dalle richieste/utenti nel sistema.
Algebricamente la legge di Little si ricava da:

$$N(t) = \frac{W(t)}{t} = 1\frac{W(t)}{t} = \frac{C(t)}{C(t)}\frac{W(t)}{t} = \frac{C(t)}{t}\frac{W(t)}{C(t)} = X(t)R(t)$$

questo per definizione di $X(t)$ e $R(t)$.

La legge di Little è importante per tre ragioni. Primo, siccome è così ampiamente applicabile (infatti richiede solo poche e deboli assunzioni), risulterà utile nella valutazione della consistenza dei dati misurati. Secondo, nello studio dei sistemi di elaborazione spesso abbiamo a disposizione solo due

dati: il numero medio di richieste e il throughput, proprio ciò che ci serve per poter applicare la legge di Little e ricavare la terza quantità il residence time. Terzo, la legge di Little è di centrale importanza negli algoritmi per la valutazione dei modelli a rete di code.

La legge di utilizzazione può essere vista come un caso particolare della legge di Little, se applicata al singolo servente, ovvero ad un sistema che ammette al più un utente.

Se durante il periodo di osservazione $[0,t]$ si verifica l'uguaglianza $A(t) = C(t)$, cioè il numero di arrivi uguaglia il numero di partenze, allora si dice che il sistema è stato in grado di smaltire tutto il lavoro che gli è stato sottoposto ovvero si ha una condizione di **bilanciamento del flusso**, ovvero non vi sono nel sistema né perdite né ingressi.
Intervalli di osservazione in cui l'identità $A(t) = C(t)$ sia verificata sono piuttosto rari, tuttavia se il periodo di osservazione è sufficientemente lungo, si può avere $A(t) \approx C(t)$ a meno di un ε.

Un metodo per verificare che in un intervallo di misura vi sia **equilibrio operazionale**, consiste nel verificare che $A(t)$ e $C(t)$ siano ε-uguali:

$$\left|\frac{A(t)-C(t)}{C(t)}\right| < \varepsilon$$

L'ε-uguaglianza è anche un buon test per decidere quando terminare il periodo di osservazione.
Negli intervalli in equilibrio operazionale risulta:

$$\lambda(t) = X(t)$$

il throughput uguaglia la velocità di arrivo, con una precisione dell'ordine di ε.

Spesso può risultare conveniente, per motivi di studio e semplificazione, supporre che il sistema soddisfi la proprietà di bilanciamento del flusso. Questa assunzione è nota come:

Assunzione del Bilanciamento del Flusso (ABF): $\boldsymbol{A = C \Rightarrow \lambda = X}$**.**

Sotto questa ipotesi la legge di utilizzazione diventa:

$$U(t) = \lambda(t)S(t)$$

Si osservi che essendo $U(t) \in [0,1]$ e quindi $0 \le U(t) \le 1$ deve risultare:

$$X(t) \le \frac{1}{S(t)} \qquad \text{ma anche} \qquad \lambda(t) \le \frac{1}{S(t)}$$

così quando $X(t) = \frac{1}{S(t)} \left(= \lambda(t)\right)$ allora $U = 1$ e il **sistema è saturato**. Da questo punto di vista possiamo concludere che $\frac{1}{S(t)}$ è la capacità massima di servizio del sistema in esame. Occorre anche notare che in un sistema che abbia raggiunto la saturazione potrebbe anche accadere che $A(t) >> C(t)$, ma chiaramente in questo caso non ci sarebbe più equilibrio operazionale.

Quando consideriamo un sistema in condizioni di equilibrio la legge di Little viene riformulata come segue.

Teorema di Little (TL): Sia r_i il tempo speso nel sistema dall'*i*-esimo utente. Se esistono e sono finiti i limiti:

$$\lambda = \lim_{t\to\infty} \frac{A(t)}{t} = \lim_{t\to\infty} \lambda(t)$$
$$N = \lim_{t\to\infty} N(t)$$

$$R = \lim_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} r_i$$

allora

$$N = \lambda R .$$

Quindi in condizioni di bilanciamento del flusso, dato che il throughput eguaglia la velocità di arrivo ($X = \lambda$) il teorema di Little afferma che:

$$N = \lambda R = XR .$$

Se stiamo analizzando un sistema interattivo, dotato cioè di terminali di interfaccia, è possibile che gli utenti non lavorino in continuazione ma abbiano dei tempi in cui sono fermi a pensare (*think time*). Chiamato Z questo tempo medio e indicato con $R_d = R + Z$ il tempo medio di residenza nel sistema, la legge di Little assume la forma: $N = XR_d = X(R+Z)$ da cui si ricava la:

Legge del Tempo di Risposta (LTR):

$$N = X(R+Z)$$
$$N = XR + XZ$$
$$XR = N - XZ$$
$$R = \frac{N}{X} - Z .$$

Come esempio di applicazione della legge del tempo di risposta supponiamo di avere un sistema con 64 utenti interattivi, un tempo di riflessione medio di 30 secondi, un throughput di 2 interazioni/secondo. Allora la legge del tempo di risposta ci dice che il tempo di risposta deve essere pari a $R = 64/2 - 30 = 2$ secondi.

Dato un sistema di elaborazione, la legge di Little può essere applicata a differenti livelli di astrazione: alla singola risorsa, ad un sottosistema o all'intero sistema. Consideriamo la seguente figura [8]:

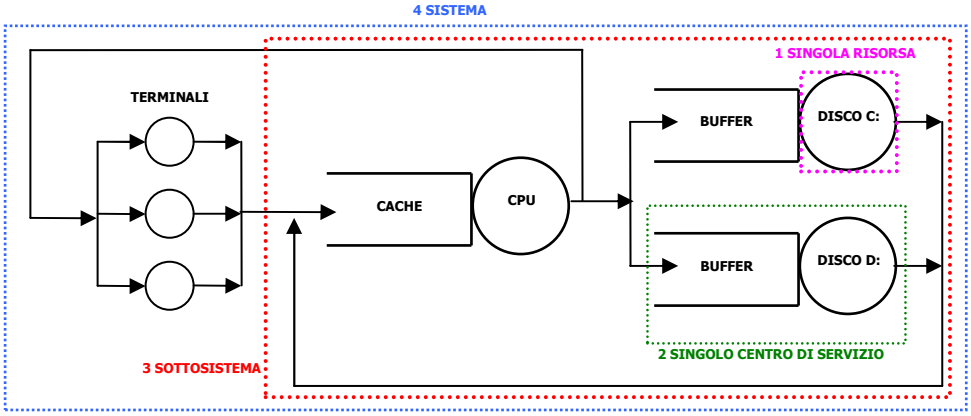


Figura A.3 - La legge di Little applicata a differenti livelli del sistema

1. ***Singola risorsa senza coda***: Consideriamo il caso di applicazione della Legge di Little alla singola risorsa del sistema sotto esame. In ogni istante di tempo *t*, può essere presente nel sistema 1 cliente oppure nessuno, e la risorsa "DISCO C:" è utilizzata solo quando è presente un cliente. Per come sono stati definiti: S è il tempo medio di servizio per richiesta, mentre R è il tempo medio di residenza nel sistema. Siccome il sistema in esame è costituito da una sola risorsa (in particolare da un centro di servizio senza coda), R deve essere uguale a S cioè il

tempo di residenza coincide con il tempo di servizio, quindi la legge di utilizzo $U = XS$ e la legge di Little $N = XR$ coincidono e risulta $N = U$: la popolazione media N del centro di servizio coincide con l'utilizzo U delle risorse presenti nel centro stesso.
Come esempio consideriamo il caso in cui il disco sia in grado di servire 40 richieste al secondo (X=40), e che il tempo di servizio medio per richiesta sia pari a 0,0225 secondi del tempo di servizio del disco (S=0,0225). La legge di Little ($U = XS$) ci dice che l'utilizzazione del disco deve essere pari a: $U = X\,S = 40 \times 0{,}0225 = 0{,}9 = 90\%$.

2. ***Singolo centro di servizio con coda***: In questo caso bisogna tener presenti anche i clienti che sono in coda e il tempo che vi trascorrono in attesa che il centro di servizio si liberi. Risulta quindi $N \neq U$ e $R \neq S$, mentre valgono le seguenti relazioni:
   - N popolazione = clienti in coda + clienti in servizio;
   - R tempo di residenza = tempo di attesa in coda + tempo di servizio;
   - X velocità alla quale vengono soddisfatte le richieste.

   Come esempio supponiamo che il numero di richieste presenti nel sistema siano 4 (N=4), e che il disco sia in grado di servire 40 richieste al secondo (X=40). Allora la legge di Little ($N = XR$) ci dice che il tempo medio speso da una richiesta presso il disco deve essere ($R = N/X$) R=4/40=0,1 secondi. Ora possiamo calcolare il tempo medio di attesa in coda di una richiesta, infatti nei 0,1 secondi di residenza calcolati, 0,0225 secondi sono dedicati al servizio mentre il restante 0,0775 è il tempo medio di attesa in coda. Possiamo calcolare anche il numero medio di richieste in coda. Abbiamo un numero medio di clienti nel sistema pari a 4 e abbiamo visto che l'utilizzazione del disco è pari al 0,9 di 4, quindi il restante 3,1 di 4 è il numero medio di richieste in attesa in coda.
3. ***Sistema senza terminali***: A questo livello di astrazione, le grandezze messe in relazione dalla legge di Little si riferiscono al vero e proprio sistema di calcolo:
   - N clienti presenti nel sottoinsieme centrale, cioè i processi che non sono sospesi in attesa della risposta dell'utente;
   - R tempo in cui le transazioni lanciate dal terminale restano nel sottosistema centrale (si noti come questo sia il concetto tradizionale di tempo di risposta di un sistema di calcolo, perciò in questo caso il tempo di residenza coincide con il tempo di risposta);
   - X frequenza alla quale fluiscono le transazioni tra sottosistema centrale e terminali.

   Supponiamo che il throughput sia di ½ transazioni al secondo (X=0,5) e che, in media, ci siano 7,5 utenti "pronti" (N=7,5). Allora la legge di Little ($N = XR$) ci dice che il tempo medio di risposta deve essere ($R = N/X$) R=7,5/0,5=15 secondi.
4. ***Intero sistema compresi i terminali***: In questo caso bisogna considerare anche i processi che sono momentaneamente sospesi in attesa che l'utente dia una risposta (*think time*):
   - N numero totale degli utenti interattivi;
   - R_d tempo di residenza = tempo di risposta (R) + tempo di riflessione (Z);
   - X frequenza alla quale fluiscono le transazioni tra sottosistema centrale e terminali.

   Si noti che il tempo di risposta coincide con il tempo di residenza del sistema al livello precedente, quando non erano stati considerati i terminali.
   Supponiamo che: ci sia un totale di 10 utenti (N=10), che il tempo medio di riflessione (*think time*) sia di 5 secondi (Z=5) e che il tempo medio di risposta sia di 15 secondi (R=15).
   Allora per la legge di Little $N = XR_d = X(R + Z)$ si ricava $X = N/(R + Z)$=10/(15+5)=10/20=0,5 interazioni/secondo.

Se consideriamo un modello di sistema costituito da M risorse, possiamo estendere l'analisi fin qui introdotta, considerando le seguenti grandezze ed indici di prestazione per ogni risorsa $1 \leq i \leq M$ in un intervallo di osservazione $[0,t]$:

$A_i(t)$	numero totale di arrivi ad i in $[0,t]$;
$C_i(t)$	numero totale di completamenti di servizio da i in $[0,t]$;
$\lambda_i(t) = \frac{A_i(t)}{t}$	tasso di arrivo ad i in $[0,t]$;
$X_i(t) = \frac{C_i(t)}{t}$	throughput di i in $[0,t]$;
$B_i(t)$	tempo di occupazione di i in $[0,t]$;
$U_i(t) = \frac{B_i(t)}{t}$	utilizzazione di i in $[0,t]$;
$S_i(t) = \frac{B_i(t)}{C_i(t)}$	servizio medio per utente di i in $[0,t]$;

da cui si ricavano:

$V_i(t) = \frac{C_i(t)}{C(t)}$	numero medio di visite (*frequenza*) ad i in $[0,t]$;
$D_i(t) = V_i(t)S(t)$	domanda totale di servizio ad i in $[0,t]$;

dove $C(t)$ e $S(t)$ sono riferiti all'intero sistema.

Per definizione e costruzione si ricava la seguente relazione fra il throughput locale e globale, nota anche come:

Legge del Flusso Forzato (LFF): le varie componenti di un sistema devono svolgere un ammontare di lavoro proporzionale a quello del sistema e al numero di visite alla risorsa:

$$X_i(t) = V_i(t)\, X(t).$$

Dato che la legge del flusso forzato deve valere per ogni i, risulta:

$$X(t) = \frac{X_i(t)}{V_i(t)}$$

per cui, considerati $1 \leq i,j \leq M$, deve risultare:

$$X(t) = \frac{X_i(t)}{V_i(t)}$$

$$X(t) = \frac{X_j(t)}{V_j(t)}$$

da cui si ricava:

$$\frac{X_i(t)}{V_i(t)} = \frac{X_j(t)}{V_j(t)}$$

che ci permette di derivare la seguente:

Legge di Consistenza (LC):

$$\frac{X_i(t)}{X_j(t)} = \frac{V_i(t)}{V_j(t)}.$$

Si noti come le V_i siano caratteristiche del sistema indipendenti dal carico di lavoro imposto al sistema stesso. Dipendono infatti solo dalle probabilità che definiscono il *routing* fra le risorse che assumiamo costanti e indipendenti dal carico. Da questo consegue che il rapporto fra i throughput di due risorse dipende solo dalle V_i , e in ultima analisi dalla probabilità che le politiche di routing portino la richiesta alla risorsa *i*-esima,, mentre non dipendono dal carico di lavoro a cui è sottoposto il sistema.

A.2 Analisi Asintotica

Il metodo di analisi asintotica è semplice ed utile per una prima valutazione dei sistemi, mentre risultati più accurati si possono ottenere con i metodi di analisi operazionale (visti nel paragrafo precedente) e gli algoritmi di risoluzione (che introdurremo nel capitolo successivo).

Due sono i tipi di limiti che verranno presentati di seguito: **limiti asintotici** e **limiti di sistema bilanciato**. I primi sono più semplici da calcolare, ma i secondi forniscono risultati più precisi.
In entrambi i casi il sistema viene definito indicando:

- il tipo dei clienti (transazionali, interattivi o batch);
- il numero di centri di servizio ($1 \leq i \leq M$);
- la domanda di servizio massima tra quelle relative ai diversi centri di servizio:

$$D_{max} = max\{D_i\}$$

- la domanda complessiva di servizio:

$$D = \sum_{i=1}^{M} D_i$$

- in caso di clienti interattivi, il tempo medi di riflessione (Z).

Gli indici di prestazione a cui siamo interessati sono:

- throughput: X
- tempo di risposta: R

in funzione del:

- numero di utenti N per un sistema chiuso;
- tasso di arrivo λ per un sistema aperto.

Calcoleremo quindi gli estremi inferiori e superiori per throughput e tempo medio di risposta indicandoli in funzione di N o λ:

- $X(N)$ e $R(N)$ per un sistema chiuso;
- $X(\lambda)$ e $R(\lambda)$ per un sistema aperto.

Per quanto riguarda i limiti, si avranno **limiti ottimistici** se considereremo l'estremo superiore di X ed inferiore di R, altrimenti avremo **limiti pessimistici**.

Dalle relazioni su X e R possiamo ricavare estremi inferiori e superiori anche relativamente ad altri indici di prestazione del sistema, sulla base delle leggi e relazioni viste in precedenza, ed in particolare per l'utilizzazione e il throughput locale:

$$U_i(N) = X(N)\, D_i$$
$$X_i(N) = X(N)\, V_i$$

con $1 \leq i \leq M$.

A.2.1 Limiti asintotici

Il metodo di seguito presentato si applica a modelli generali di cui i modelli a rete di code sono un caso particolare. L'ipotesi di base è che la domanda di servizio di un cliente nei confronti di un centro non dipenda né dal numero di clienti presenti nel sistema, né dalla loro distribuzione nei diversi centri che compongono il sistema stesso.
Distinguiamo i due casi di sistema aperto e sistema chiuso.

Modelli aperti (carico transazionale)

Al crescere della frequenza di arrivo dei clienti λ, il sistema raggiunge un limite λ_{max} oltre il quale non è più capace di soddisfare le richieste dei nuovi clienti (saturazione). Il valore di saturazione λ_{max} è quindi il valore massimo di frequenza di arrivo sopportabile dal sistema.
Si hanno le seguenti:

Definizione: **Sistema Saturo**

Un sistema si dice saturo se il tempo di risposta non è limitato superiormente, ovvero un sistema è saturo se contiene almeno una risorsa satura.

Definizione: **Risorsa Satura**

Una risorsa i-esima si dice satura se risulta: $U_i = 1$ (100%).

Si consideri la legge di utilizzo applicata alla singola risorsa *i*-esima:

$$U_i = X_i S_i$$

Dalla legge del flusso forzato:

$$X_i = X V_i$$

e dall'ipotesi di bilanciamento del flusso:

$$\lambda = X$$

si ricava:

$$X_i = \lambda V_i$$

sostituendo quest'ultima espressione nella legge di utilizzo si ricava:

$$U_i = \lambda V_i S_i$$

da cui per definizione di $D_i = V_i S_i$ si ricava:

$$U_i = \lambda D_i$$

Ricordando la definizione di risorsa satura deve risultare:

$$U_i = \lambda D_i \leq 1$$

da cui deriviamo che il massimo carico che una risorsa può sostenere è dato da:

$$\lambda \leq \frac{1}{D_i}$$

Il massimo carico che il modello/sistema può sostenere è dato da:

$$\lambda \leq \lambda_{max} = \frac{1}{D_{max}}$$

Finché il tasso di arrivo $\lambda<\lambda_{max}$ allora il sistema è stabile e vale la condizione di bilanciamento del flusso $X = \lambda$. Quando $\lambda \geq \lambda_{max}$, il sistema è instabile ed i tempi di attesa e di risposta tendono a crescere senza limite finito. Il nodo saturo rappresenta un collo di bottiglia per il sistema.

Limiti ottimistici:

Abbiamo:

$$X(\lambda) \leq \lambda_{max}$$
$$R(\lambda) \geq D$$

Infatti, nel caso migliore (limite inferiore), ogni cliente non interferisce con nessun altro cliente e quindi non ci sono ritardi dovuti all'attesa in coda. In questo caso il tempo di risposta equivale alla domanda di servizio complessiva ($R = D$).

Limiti pessimistici:
Per un sistema aperto non è possibile valutare il caso peggiore (limite superiore). È facile verificare infatti che se, per assurdo, R_{max} è un limite superiore al tempo di risposta del sistema con tasso di arrivo λ qualsiasi, se consideriamo, ad esempio, la possibilità che i clienti arrivino a gruppi *n* ogni $\frac{n}{\lambda}$ unità di tempo (il tasso di arrivo del sistema è ancora $\lambda = \frac{n}{n/\lambda}$), scegliendo $n > \frac{R_{max}}{D}$ osserviamo un tempo di risposta superiore ad R_{max} , che quindi non può essere un limite superiore. Perciò se non è nota la distribuzione degli arrivi, ma si conosce solo la frequenza media λ, non è possibile stabilire un limite superiore al tempo di risposta.

Modelli chiusi (carico batch o interattivo)
Per la definizione dei limiti nel caso dei modelli chiusi utilizziamo carichi di tipo interattivo. Per determinare i carichi batch sarà poi sufficiente eliminare il tempo di riflessione *Z*, in quanto i carichi batch sono equivalenti a quelli interattivi con *Z*=0.
In maniera analoga a quanto fatto in precedenza, e notando che valgono le seguenti relazioni:

$$\lambda \leq \frac{1}{D_i}$$
$$U_i = X_i S_i$$
$$X_i = X V_i$$
$$D_i = V_i S_i$$

segue:

$$U_i = X_i S_i = X V_i S_i = X D_i$$

e quindi si ricava la legge di utilizzazione:

$$U_i(N) = X(N) D_i$$

Siccome l'utilizzo di ogni centro deve comunque essere inferiore al 100%, cioè $U_i(N) \leq 1$, si ricava che per ogni centro *i* deve valere la seguente:

$$X(N) \leq \frac{1}{D_i}$$

e quindi si ottiene il primo vincolo sul traffico del sistema:

$$X(N) \leq \frac{1}{D_{max}}$$

ed il massimo corrisponde al collo di bottiglia del sistema.

Si consideri ora il caso in cui nel sistema ci sia un solo utente interattivo *N*=1; il traffico è dato da:

$$X(1) = \frac{1}{D+Z}$$

infatti si potrà osservare un completamento ogni $D + Z$ unità di tempo.

Quando si aggiungono *N*-1 clienti, nel caso migliore non si verifica nessuna interferenza e si ottiene:

$$X(N) = \frac{N}{D+Z}$$

cioè si osserveranno N completamenti ogni $D + Z$ secondi.

Nel caso peggiore, invece, ogni cliente deve aspettare in coda che tutti gli altri siano serviti, e perciò deve aspettare $(N\text{-}1)D$ unità di tempo che gli altri clienti terminino il loro servizio, oltre ad una ulteriore unità D di tempo del proprio servizio e Z unità di tempo di riflessione. Si ottiene quindi:

$$X(N) = \frac{N}{ND + Z}$$

dove ND è dato da $ND = (N\text{-}1)\,D + D$.
Riassumendo, nel caso di carico interattivo si hanno i seguenti limiti asintotici sul throughput del sistema:

$$\frac{N}{ND + Z} \le X(N) \le \min\left\{\frac{1}{D_{\max}}, \frac{N}{D + Z}\right\}$$

mentre in modo analogo, per carichi di tipo batch si ha:

$$\frac{1}{D} = \frac{N}{ND} \le X(N) \le \min\left\{\frac{1}{D_{\max}}, \frac{N}{D}\right\}$$

Il cui andamento può essere graficamente rappresentato dai grafici seguenti [8][11]:

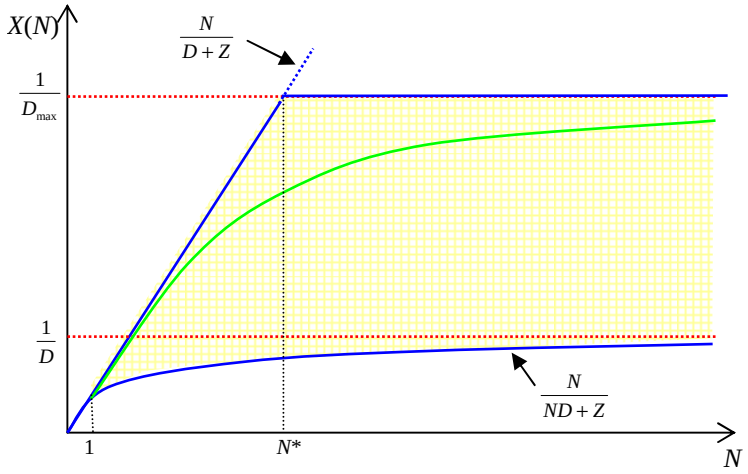


Figura A.4 - Grafico dell'andamento del throughput (utenti interattivi)

Come si vede dal grafico, per carichi con utenti interattivi, $X(N)$ deve essere contenuto all'interno dell'area evidenziata a quadratini gialli contornata da linea blu continua, il cui bound inferiore è dato dalla curva: $\frac{N}{ND + Z}$ e il bound superiore dato dall'intersezione delle due curve $\frac{1}{D_{\max}}$ e $\frac{N}{D + Z}$ rappresentata da: $\min\left\{\frac{1}{D_{\max}}, \frac{N}{D + Z}\right\}$.

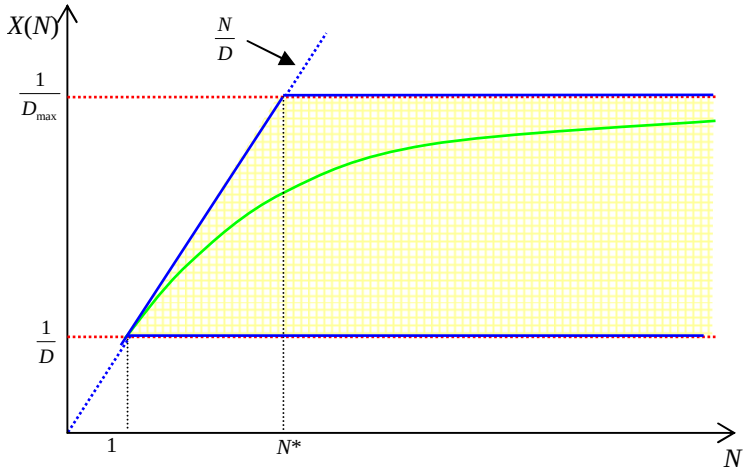


Figura A.5 - Grafico dell'andamento del throughput (utenti batch)

Anche in questo caso il grafico evidenzia che, per carichi batch, $X(N)$ deve essere contenuto all'interno dell'area evidenziata a quadratini gialli contornata da linea blu continua, il cui bound inferiore è dato dalla curva: $\frac{1}{D}$ e il bound superiore dato dall'intersezione delle due curve $\frac{1}{D_{\max}}$ e $\frac{N}{D}$ rappresentata da: $\min\left\{\frac{1}{D_{\max}}, \frac{N}{D}\right\}$.

Dalla legge del tempo di risposta $R(N) = \frac{N}{X} - Z$, si ricava:

$$X = \frac{N}{R(N) + Z}$$

sostituendo questa espressione nelle disequazioni dei limiti asintotici del throughput sopra esposte, si ottengono i limiti asintotici del tempo di risposta per sistemi interattivi:

$$\frac{N}{ND + Z} \le X(N) \le \min\left\{\frac{1}{D_{\max}}, \frac{N}{D + Z}\right\}$$

$$\frac{N}{ND + Z} \le \frac{N}{R(N) + Z} \le \min\left\{\frac{1}{D_{\max}}, \frac{N}{D + Z}\right\}$$

invertendo ogni termine si ottiene:

$$\frac{ND + Z}{N} \ge \frac{R(N) + Z}{N} \ge \max\left\{D_{\max}, \frac{D + Z}{N}\right\}$$

moltiplicando il tutto per N la disuguaglianza si mantiene inalterata:

$$ND + Z \ge R(N) + Z \ge \max\{ND_{\max}, D + Z\}$$

sottraendo Z si ottiene:

$$ND \ge R(N) \ge \max\{ND_{\max} - Z, D\}$$

da cui:

$$\max\{D, (ND_{\max} - Z)\} \le R(N) \le ND$$

Per i sistemi batch sarà sufficiente porre Z=0, ottenendo:

$$\max\{D, ND_{\max}\} \le R(N) \le ND$$

il cui andamento è rappresentato dal seguente grafico [8][11]:

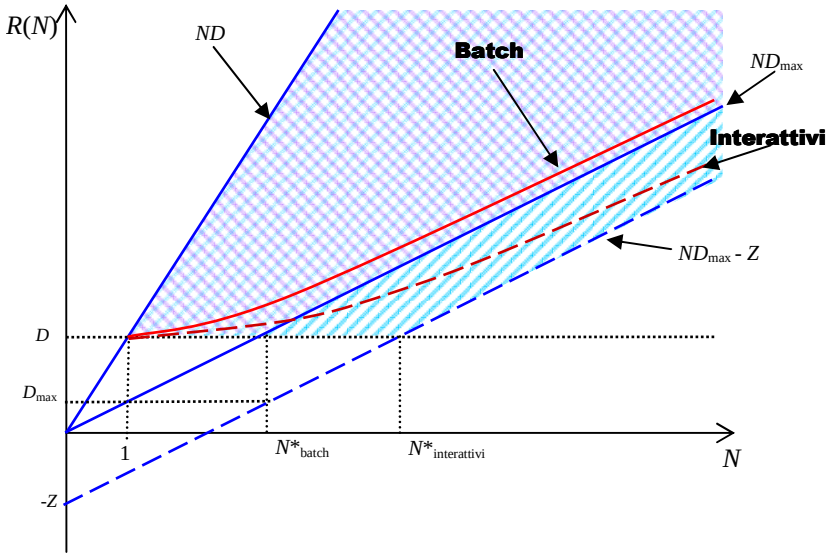


Figura A.6 - Andamento del tempo di risposta per utenti batch (linea continua) e interattivi (tratteggiata)

In questo caso i due grafici sono stati uniti per meglio porre in evidenza le similitudini e le differenze. *ND* è un limite superiore sia per clienti batch che interattivi. Invece, come limite inferiore, gli utenti interattivi hanno $\max\{D,(ND_{\max}-Z)\}$, mentre gli utenti batch hanno $\max\{D,(ND_{\max})\}$.

Il punto di equivalenza [11] dei due limiti superiori del traffico e dei due limiti inferiori del tempo di risposta è:

$$N^* = \frac{D+Z}{D_{\max}}$$

Definizione: **Carico leggero**

Si definisce carico leggero, un carico dovuto a un numero di clienti inferiore a *N**, quando cioè il limite superiore al traffico è stabilito dalla retta:

$$X = \frac{N}{D+Z}$$

mentre il limite inferiore del tempo di risposta è indicato dalla retta:

$$R = D\,.$$

Definizione: **Carico pesante**

Si definisce carico pesante, un carico dovuto a un numero di clienti superiore a *N**, quando cioè il limite superiore al traffico è stabilito dalla retta:

$$X = \frac{1}{D_{\max}}$$

mentre il limite inferiore del tempo di risposta è indicato dalla retta:

$$R = ND_{\max} - Z\,.$$

Nel caso di carichi leggeri, traffico e tempo di risposta sono vincolati dal valore della domanda complessiva, quindi è possibile migliorare le prestazioni diminuendone il valore. Quando invece ci si trova in situazione di carico pesante il vincolo è la domanda massima, quella cioè che identifica quale dei centri di servizio è il collo di bottiglia del sistema. Miglioramenti delle prestazioni sono possibili solo se viene rimosso questo collo di bottiglia.

In pratica questo vuol dire che, nel caso di carichi batch, oltre N^* un incremento del carico ha un impatto più limitato se non addirittura negativo sulle prestazioni del sistema. Il throughput è infatti limitato dall'asintoto $1/D_{max}$ e il tempo di risposta tende a crescere almeno linearmente con il carico.
$N \leq N^*$ rappresenta quindi un possibile criterio da adottare per garantire prestazioni accettabili del sistema. Nel caso di utenti interattivi, la soglia N^* può essere utilizzata in questo caso, ad esempio, per determinare il numero massimo di terminali collegabili al sistema in modo da mantenere un buon livello di prestazioni.

Tabella A-1 - Schema riassuntivo per il calcolo dei limiti asintotici

Sistemi	Throughput	Response Time
Transazionali	$X(\lambda) \leq \lambda_{max} = \frac{1}{D_{max}}$	$D \leq R(\lambda)$
Batch	$\frac{1}{D} = \frac{N}{ND} \leq X(N) \leq \min\left\{\frac{1}{D_{max}}, \frac{N}{D}\right\}$	$\max\{D, ND_{max}\} \leq R(N) \leq ND$
Interattivi	$\frac{N}{ND+Z} \leq X(N) \leq \min\left\{\frac{1}{D_{max}}, \frac{N}{D+Z}\right\}$	$\max\{D, (ND_{max} - Z)\} \leq R(N) \leq ND$

A.2.2 Limiti di sistemi bilanciati

I limiti di sistemi bilanciati consentono di ottenere risultati più precisi rispetto a quelli forniti dai limiti asintotici, richiedendo però dei calcoli più complessi.

Definizione: **Sistema Bilanciato**

Un sistema si dice bilanciato quando tutte le domande ai diversi centri di servizio sono uguali:

$$D_1 = D_2 = \ldots = D_M$$

ovvero:

$$D_i = \frac{D}{M} = D_{balanced} \equiv D_b \,.$$

Per semplicità analizzeremo solo il caso di clienti batch, ma gli altri casi possono essere ottenuti con le stesse procedure usate in precedenza per i sistemi non bilanciati [par. A.2.1].

Il tempo di residenza per il centro *i*-esimo è dato da:

$$R_i(N) = D_i[1 + A_i(N)]$$

dove $A_i(N)$ è la coda che il cliente *N*-esimo vede davanti a se quando arriva al centro *i*-esimo. $A_i(N)$ non deve essere confuso con $Q_i(N)$ che rappresenta invece il numero dei clienti compreso l'*N*-esimo presenti in media al centro *i*-esimo. È da notare inoltre che per i centri di ritardo risulta: $A_i(N) = 0$.

Il tempo di risposta del sistema è dato dalla somma dei tempi di residenza agli *M* centri:

$$R(N) = \sum_{i=1}^{M} R_i(N) = \sum_{i=1}^{M} D_i[1 + A_i(N)]$$

Dato che stiamo parlando di sistemi bilanciati e risulta quindi $\forall\, i : D_i = D_b$ si ottiene:

$$R(N) = \sum_{i=1}^{M} D_i[1 + A_i(N)] = \sum_{i=1}^{M} D_b[1 + A_i(N)] = D_b \sum_{i=1}^{M} 1 + A_i(N) =$$

$$= D_b\left[\sum_{i=1}^{M} 1 + \sum_{i=1}^{M} A_i(N)\right] = D_b\left[M + \sum_{i=1}^{M} A_i(N)\right]$$

Per i sistemi batch, risulta che il numero totale di clienti visti in coda dall'*N*-esimo cliente è pari al numero totale dei clienti presenti nel sistema (*N*) tranne se stesso, quindi si ha:

$$\sum_{i=1}^{M} A_i(N) = N - 1$$

Perciò ci è consentito esprimere il tempo di risposta come:

$$R(N) = D_b[M + N - 1]$$

Considerando tutti i possibili sistemi con *M* centri ed *N* clienti con domanda massima:

$$D_{\max} = \max\{D_i\}_{1 \le i \le M}$$

e domanda minima:

$$D_{\min} = \min\{D_i\}_{1 \le i \le M}$$

quello che avrà le **migliori prestazioni** sarà il sistema bilanciato con $\boldsymbol{D_b = D_{min}}$ mentre quello con le **peggiori prestazioni** sarà quello con $\boldsymbol{D_b = D_{max}}$.
Da questi due casi limite si ricava:

$$(M + N - 1)D_{\min} \le R(N) \le (M + N - 1)D_{\max}$$

Sfruttando la legge di Little $R = N / X$ è possibile ottenere i limite del throughput:

$$\frac{N}{(M+N-1)}\frac{1}{D_{\max}} \le X(N) \le \frac{N}{(M+N-1)}\frac{1}{D_{\min}}$$

È possibile ottenere delle stime ancora più restrittive per questi limiti, considerando solo quei sistemi con domanda complessiva pari a:

$$D = \sum_{i=1}^{M} D_i$$

fra tutti questi sistemi, quello che avrà le migliori prestazioni è quello bilanciato, in cui la domanda di servizio ai diversi centri è pari a:

$$D_{\text{med}} = \frac{D}{M}$$

da cui si possono ricavare i seguenti:

Limiti ottimistici:

$$R(N) \ge (M + N - 1)D_{\text{med}} = MD_{\text{med}} + (N - 1)D_{\text{med}} = D + (N - 1)D_{\text{med}}$$

$$X(N) \le \frac{N}{(M+N-1)}\frac{1}{D_{\text{med}}} = \frac{N}{MD_{\text{med}} + (N-1)D_{\text{med}}} = \frac{N}{D + (N-1)D_{\text{med}}}$$

Il caso peggiore, invece, si ha quando solo $M' = \frac{D}{D_{\max}}$ centri hanno domanda pari a $D_{\max}$, mentre la domanda degli altri $M'' = M - M'$ centri è pari a 0. Si hanno quindi i seguenti:

Limiti pessimistici:

$$R(N) \le \left(\frac{D}{D_{\max}} + N - 1\right)D_{\max} = D + (N-1)D_{\max}$$

$$X(N) \ge \frac{N}{\frac{D}{D_{\max}} + N - 1}\frac{1}{D_{\max}} = \frac{N}{D + (N-1)D_{\max}}$$

Tabella A-2 - Schema riassuntivo per il calcolo dei limiti di sistemi bilanciati

Sistemi	Throughput	Response Time
Transazionali	$X(\lambda) \le \lambda_{\max} = \frac{1}{D_{\max}}$	$\frac{D}{1-\lambda D_{\text{med}}} \le R(\lambda) \le \frac{D}{1-\lambda D_{\max}}$
Batch	$\frac{N}{D+(N-1)D_{\max}} \le X(N) \le \min\left\{\frac{1}{D_{\max}}, \frac{N}{D+(N-1)D_{\text{med}}}\right\}$	$\max\{ND_{\max}, D+(N-1)D_{\text{med}}\} \le R(N) \le D+(N-1)D_{\max}$
Interattivi	$\frac{N}{D+Z+\frac{(N-1)D_{\max}}{1+\frac{Z}{ND}}} \le X(N) \le \min\left\{\frac{1}{D_{\max}}, \frac{N}{D+Z+\frac{(N-1)D_{\text{med}}}{1+\frac{Z}{D}}}\right\}$	$\max\left\{(ND_{\max}-Z), D+\frac{(N-1)D_{\text{med}}}{1+\frac{Z}{D}}\right\} \le R(N) \le D+\frac{(N-1)D_{\max}}{1+\frac{Z}{ND}}$

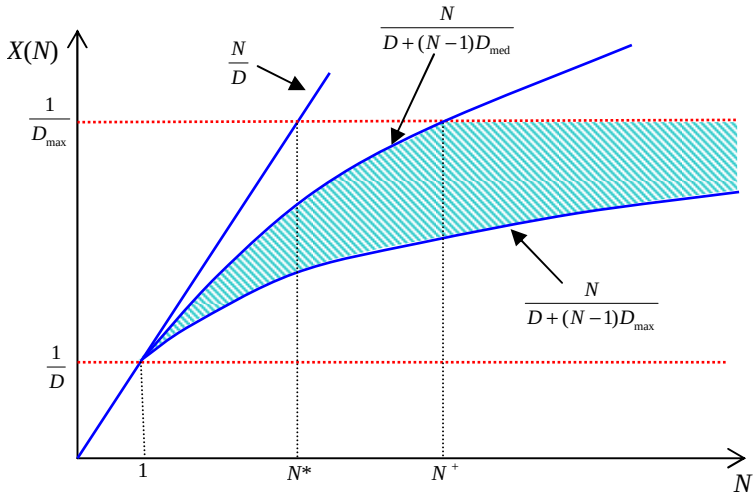


Figura A.7 - Sistemi bilanciati, limiti sulle prestazioni (Throughput con clienti Batch)

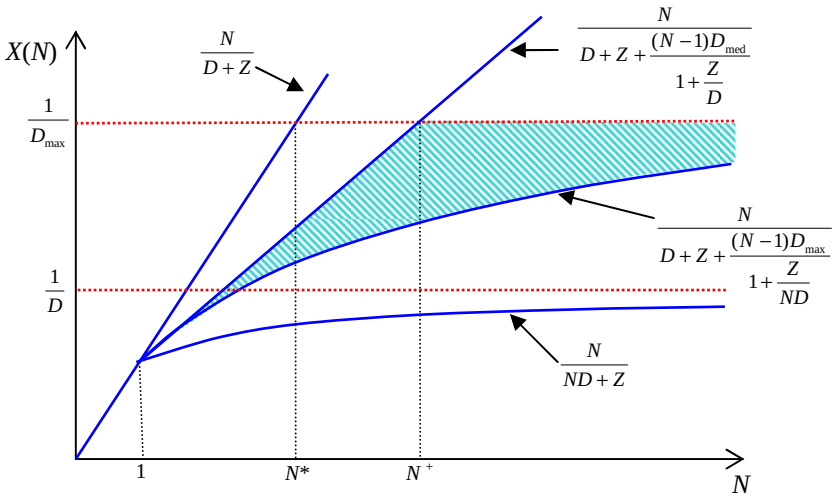


Figura A.8 - Sistemi bilanciati, limiti sulle prestazioni (Throughput con clienti Interattivi)

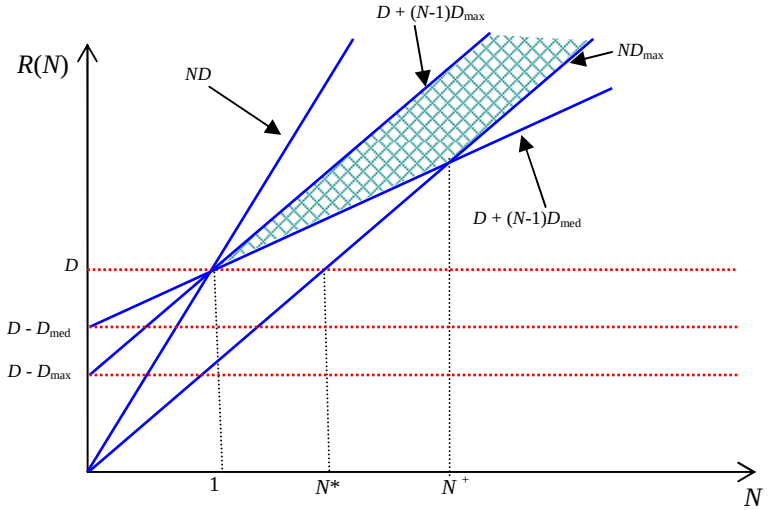


Figura A.9 - Sistemi bilanciati, limiti sulle prestazioni (Tempo di risposta con clienti Batch)

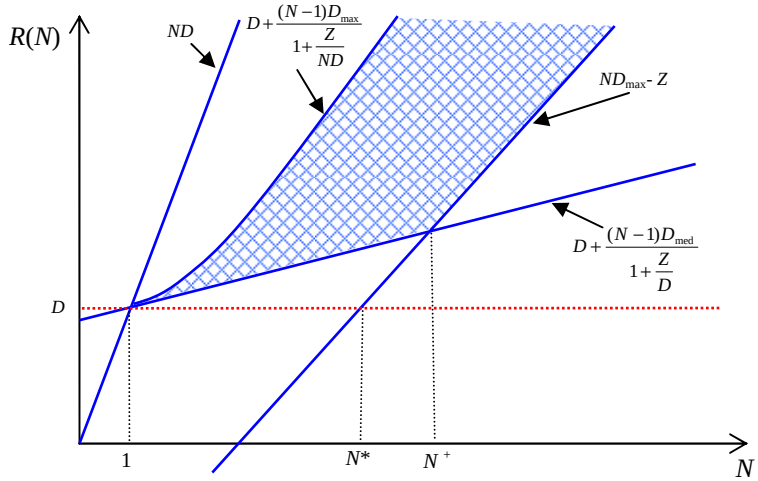


Figura A.10 - Sistemi bilanciati, limiti sulle prestazioni (Tempo di risposta con clienti Interattivi)

A.3 Soluzione Analitica

Man mano che la tecnologia avanza, i sistemi di elaborazione e le reti di comunicazione diventano sempre più complessi. Le moderne applicazioni Client-Server e le applicazioni multimediali sono ora ampiamente usati. Le interazioni tra le numerose componenti di cui sono formate, e i carichi di lavoro a cui danno origine, rendono l'intuizione umana non più adeguata alla comprensione dei delicati meccanismi e interazioni che si generano in un moderno sistema di elaborazione. Soprattutto risulta molto difficile predire con un ridotto margine d'errore quale sarà l'impatto che una modifica nella configurazione di un sistema produrrà sulle sue performance globali.

I modelli a reti di code sono stati ampiamente adottati come valido strumento per la valutazione delle performance dei moderni sistemi di elaborazione e delle reti di comunicazione. Tuttavia risulta quasi impossibile calcolare la soluzione esatta per la classe dei modelli a reti di code generali. Ma la soluzione esatta può essere trovata per un importante sottoinsieme delle reti di code: le **reti di code separabili** (*separable*) o in **forma-prodotto** (*product-form*). La soluzione di un modello a reti di code separabili o in forma-prodotto può essere agevolmente trovata con un modesto costo computazionale usando alcuni algoritmi esatti e molto efficienti. I due principali algoritmi per modelli di code chiuse in forma prodotto sono: l'algoritmo di Convoluzione di Buzen [21] e l'algoritmo MVA (Mean Value Analysis) di Reiser e Lavenberg [22].

L'algoritmo di convoluzione di Buzen risale al 1973 e si applica alle sole reti di code separabili a singola classe e disciplina di servizio FIFO (*First Come First Served*). Più tardi, Reiser e Lavenberg nel 1980 svilupparono l'algoritmo MVA migliorando un lavoro di Reiser e Kobayashi del 1975 [44], che estendeva l'algoritmo di convoluzione alle reti di code separabili multi classe.

Non si può affermare che un algoritmo sia migliore dell'altro in assoluto, in quanto oltre alla complessità computazionale occorre considerare altri fattori, quali, ad esempio, il tipo di rete e di nodi nella rete, lo sforzo di programmazione e i problemi di instabilità numerica. In generale, si può affermare che l'MVA è da preferire per reti costituite da nodi a capacità fissa e reti con infiniti serventi. Per modelli a rete formate da nodi a capacità dipendente dal carico la versione originale dell'algoritmo MVA può risultare numericamente instabile. È stata proposta una versione modificata (MMVA) [23] numericamente stabile ma che comporta un aumento della complessità computazionale in termini di spazio e tempo molto elevato.

Dal punto di vista della complessità computazionale i due algoritmi sono equivalenti e richiedono tempo e spazio di ordine polinomiale nelle componenti della rete, ovvero nel numero di nodi e di clienti. Il vantaggio principale dell'algoritmo MVA, rispetto a quello di convoluzione, è che l'MVA riesce a superare alcuni problemi di stabilità numerica dell'algoritmo di convoluzione. L'algoritmo di convoluzione [21] è basato sul calcolo della costante di normalizzazione G e può dar luogo a problemi di instabilità numerica sia dovuti ad overflow/underflow e sia dovuti ad errori di troncamento ed arrotondamento.

Per quanto riguarda i modelli a rete in forma prodotto con più classi di clienti gli algoritmi di Convoluzione ed MVA richiedono una complessità computazionale che è ancora polinomiale nel numero di nodi ed utenti, ma esponenziale nel numero di catene. Per tali modelli sono stati proposti in letteratura altri algoritmi basati su espressioni ricorsive sul numero di catene, la cui complessità computazionale è polinomiale nel numero di catene, ma esponenziale nel numero di nodi o nel numero di utenti. Fra questi ricordiamo gli algoritmi: "Recursion by Chains" (RECAL) [24], "Mean Value Analysis by Chains" (MVAC) [25], rispettivamente basati sull'operazione di convoluzione e di MVA, e l'algoritmo: "Distribution Analysis by Chain" (DAC) [26] che permette di calcolare soltanto

alcuni indici di prestazione globali del modello [27]. Infine per modelli cosiddetti sparsi in cui gli utenti di ogni catena visitano pochi nodi, può essere molto efficiente applicare algoritmi strutturati ad albero, sia di tipo convoluzione (*The Tree Convolution Algorithm*) [28] che MVA (*The Tree MVA*) [29].
È comunque da considerare che la complessità computazionale di questi algoritmi alternativi, nel caso peggiore, non è così buona come quella dell'MVA e dell'algoritmo di convoluzione.

È chiaro quindi, che per le reti di code separabili, il costo computazionale di una soluzione esatta diventa proibitivamente costoso al crescere del numero delle classi, dei clienti e dei centri di servizio.
Per questo motivo, sono stati sviluppati alcuni metodi di approssimazione basati sull'algoritmo MVA. Gli algoritmi che sfruttano questi metodi di approssimazione prendono il nome di algoritmi MVA approssimati (*Approximate MVA Algorithms*), e sono classificabili in due tipi:

- metodi utili per l'analisi di reti di code non separabili;
- metodi utili per l'analisi di reti di code separabili.

Questi ultimi sono una semplice versione approssimata dell'algoritmo MVA esatto, che riduce il costo computazionale della soluzione.
Esistono diversi algoritmi approssimati per le reti di code separabili, ma quelli che hanno avuto il maggior successo presso gli analisti di performance sono:

- Algoritmo di Bard-Schweitzer: Proportional Estimation (PE) [42];
- Algoritmo di Chandy-Neuse: Linearizer Algorithm [43].

Entrambi sono algoritmi approssimati iterativi che richiedono un minor costo computazionale rispetto alla versione esatta dell'MVA. L'algoritmo Linearizer risulta essere più accurato rispetto all'algoritmo PE, ma con un costo computazionale maggiore.
Esistono un certo numero di varianti dell'algoritmo Linearizer originale, sviluppate per tentare di ridurre la sua complessità spaziale e temporale. I più popolari sono:

- The AQL Algorithm;
- The improved Linearizer Algorithm;
- The SSB Algorithm;
- The Queue Line Algorithm (QL);
- The Fraction Line Algorithm (FL).

Tralasciando i dettagli di queste implementazioni si può dare una schematizzazione della loro efficienza mediante il seguente grafico di prestazioni [41]:

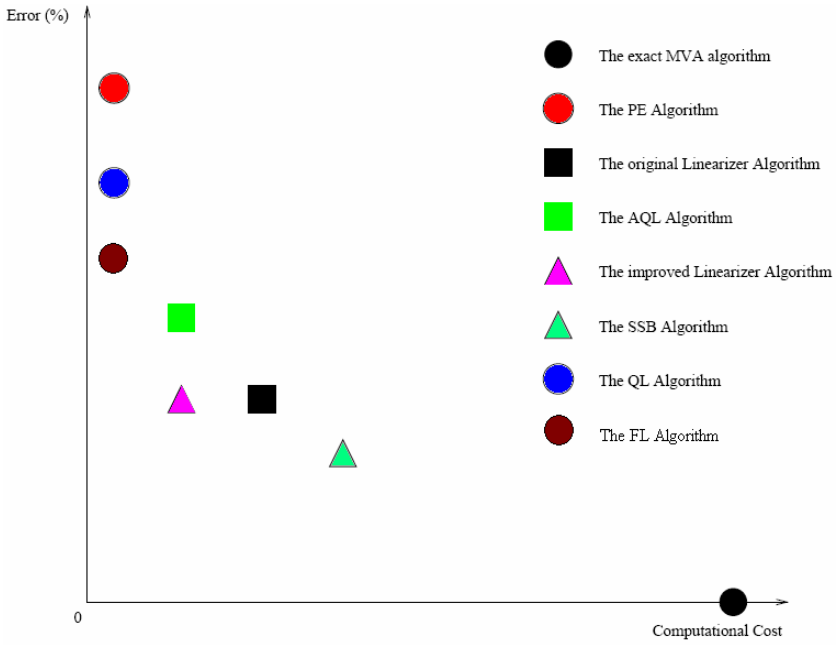


Figura A.11- Prestazioni di alcuni algoritmi MVA approssimati

In questo capitolo distinguiamo tra modelli chiusi e modelli aperti, modelli a singola classe di clienti (*One Job Class* - OJC) e modelli con classi multiple di clienti (*Multiple Job Class* - MJC). I modelli con più classi di clienti possono essere classificati a seconda che i clienti siano solo transazionali (modelli aperti), solo interattivi o batch (modelli chiusi) oppure sia transazionali che interattivi o batch (modelli misti). In questo paragrafo analizzeremo le tecniche e gli algoritmi classici per la risoluzione di queste tipologie di modelli fornendo anche esempi di reti di code e della loro risoluzione analitica.

A.3.1 Sistemi aperti (Transazionali) a singola classe di clienti

Sia λ la frequenza di arrivo (*arrival rate*) dei clienti (o *job*) nel sistema aperto. Indichiamo con D_k la domanda di servizio (*service demand*) che un singolo cliente fa al centro di servizio *k*-esimo.
Sia inoltre V_k il numero di visite che quel cliente fa al centro di servizio *k*-esimo, e sia il centro di servizio $k = 0$ la coda di riferimento (il mondo esterno o *source node*).
Definiamo μ_k come la frequenza di servizio (*service rate*), e quindi $S_k = 1/\mu_k$ come il tempo di servizio (*service time*) del *k*-esimo centro di servizio.

Risulta:

$$D_k = V_k \frac{1}{\mu_k} = V_k \cdot S_k$$

Per meglio comprendere queste grandezze facciamo un esempio:

ES:
*Se la domanda totale di servizio di un cliente ad un centro di servizio è pari a 10sec, e il centro di servizio ha un service time ti 5sec allora il numero di visite (o richieste di servizio) che il cliente fa al centro di servizio sarà pari a 10/5 = 2 visite/richieste, per cui risulta 5×2=10, ed il service rate sarà quindi pari a 1/5 = 0,2***.**

Il massimo throughput per il sistema si ha per il valore della frequenza di arrivo che satura il centro di servizio con la domanda più grande:

$$X_{max} = \frac{1}{\max\limits_{1 \le k \le K}(D_k)} = \lambda_{sat}$$

Oltre questo valore il sistema diventa saturo. Di seguito considereremo sempre un sistema non saturo cioè tale che $\lambda < \lambda_{sat}$ ovvero un sistema per cui vale l'ABF (assunzione del bilanciamento del flusso):

$$\text{Arrivi} = \text{Completamenti} \Rightarrow \lambda = X.$$

Per ogni centro di servizio *k* si può calcolare:

- il **throughput**, che per la legge del flusso forzato è dato da:

$$X_k(\lambda) = \lambda V_k = \lambda_k$$

- l'**utilizzazione** dei diversi centri di servizio, che per la legge di utilizzo è data da:

$$U_k(\lambda) = X_k(\lambda) \cdot S_k = \lambda \cdot V_k \cdot S_k = \lambda \cdot V_k \cdot \frac{1}{\mu_k} = \lambda D_k$$

  ma anche:

$$U_k(\lambda) = \lambda_k \cdot \frac{1}{\mu_k}$$

- **tempo di risposta (o residenza) della singola risorsa,** distinguiamo due casi in base alla disciplina di servizio della coda:

- Disciplina IS (centri di ritardo o *Delay Center*), non c'è tempo di attesa in coda in quanto ci sono infiniti serventi, quindi risulta:

$$R_k(\lambda) = V_k S_k = V_k \frac{1}{\mu_k} = D_k$$

ovvero il tempo di residenza è uguale alla domanda di servizio.

- Discipline FCFS, LCFSPR o PS (centri di accodamento o *Service Center*). Il tempo di risposta è dato dal tempo di attesa in servizio più il tempo di attesa in coda.
Il tempo di attesa in servizio è semplicemente dato dalla domanda di servizio:

$$D_k = V_k S_k = V_k \frac{1}{\mu_k}$$

Il tempo di attesa in coda è dato da:

$$A_k(\lambda) \cdot D_k = A_k(\lambda) \cdot V_k S_k = A_k(\lambda) \cdot V_k \cdot \frac{1}{\mu_k}$$

dove con $A_k(\lambda)$ si indica la **lunghezza della coda** che un cliente trova quando arriva nel centro *k*-esimo.

ricapitolando si ha:

$$R_k(\lambda) = \begin{cases} D_k & \textit{per disciplina IS} \\ D_k\left(1 + A_k(\lambda)\right) & \textit{per discipline FCFS, PS o LCFSPR} \end{cases}$$

Per reti di code aperte in forma prodotto a singola classe, la lunghezza della coda che un nuovo cliente vede quando arriva al centro *k* (espressa da $A_k(\lambda)$) è uguale al numero medio dei clienti presenti allo stesso centro (indicato con $Q_k(\lambda)$).

$$Q_k(\lambda) = A_k(\lambda)$$

La media è da intendersi come calcolata su tutto il periodo di osservazione, non solo fino all'istante d'arrivo del nuovo cliente.

Definiamo, per comodità, il tempo di residenza al centro k per ogni singola visita come:

$$\overline{R_k(\lambda)} = \frac{R_k(\lambda)}{V_k}$$

Applicando la legge di Little, $N = X\,R$, al centro *k* si ottiene:

$$Q_k(\lambda) = X_k(\lambda) \cdot \overline{R_k(\lambda)}$$

applicando la legge del flusso forzato, come già fatto in precedenza, si ricava:

$$Q_k(\lambda) = \lambda V_k \cdot \overline{R_k(\lambda)} = \lambda V_k \cdot \frac{R_k(\lambda)}{V_k} = \lambda R_k(\lambda)$$

$$Q_k(\lambda) = \lambda R_k(\lambda)$$

Così, dato che per code IS risulterà $Q_k(\lambda) = 0$, nel periodo totale di osservazione si può riscrivere la definizione di $R_k(\lambda)$ come:

$$R_k(\lambda) = D_k\left(1 + Q_k(\lambda)\right) = D_k + \lambda\, D_k\, R_k(\lambda) = D_k + U_k(\lambda)\, R_k(\lambda)$$

cioè:

$$D_k = R_k(\lambda) - U_k(\lambda) R_k(\lambda) = R_k(\lambda)\left(1 - U_k(\lambda)\right)$$

da cui si ricava la seguente espressione del tempo di risposta:

$$R_k(\lambda) = \frac{D_k}{1 - U_k(\lambda)}$$

Questa espressione mette meglio in evidenza il concetto intuitivo di tempo di risposta, secondo cui valgono i seguenti comportamenti asintotici:

$$\lim_{U_k(\lambda)\to 0} R_k(\lambda) = D_k$$

$$\lim_{U_k(\lambda)\to 1} R_k(\lambda) = \infty$$

Ovvero, quando l'utilizzazione tende a zero, il tempo di risposta tende alla domanda di servizio; mentre quando l'utilizzazione si avvicina al punto di saturazione, allora il tempo di risposta tende all'infinito.

- il **tempo di risposta complessivo del sistema** si ottiene sommando i tempi di residenza ai diversi centri di servizio:

$$R(\lambda) = \sum_{k=1}^{K} R_k(\lambda)$$

- la **lunghezza della coda al centro k** (ovvero il numero medio di clienti presenti in un centro), come abbiamo già visto, risulta essere pari a:

$$Q_k(\lambda) = \lambda R_k(\lambda)$$

Quando si considerano centri di ritardo (disciplina di servizio IS) si ha: $R_k = D_k$, mentre per centri di accodamento (discipline di servizio FCFS, LCFSPR, PS) si ha: $R_k = \frac{D_k}{1 + U_k(\lambda)}$

da cui si ricava:

$$Q_k(\lambda) = \lambda R_k(\lambda) = \begin{cases} \lambda D_k \\ \lambda \dfrac{D_k}{1 + U_k(\lambda)} \end{cases} = \begin{cases} U_k(\lambda) & \textit{per disciplina IS} \\ \dfrac{U_k}{1 + U_k(\lambda)} & \textit{per discipline FCFS, LCFSPR, PS} \end{cases}$$

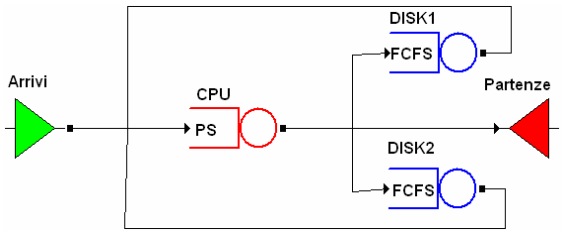
- il **numero medio di clienti** presenti complessivamente nel sistema, si ottiene sommando le code presenti ai diversi centri di servizio:

$$Q(\lambda) = \sum_{k=1}^{K} Q_k(\lambda) = \sum_{k=1}^{K} \lambda R_k(\lambda) = \lambda \sum_{k=1}^{K} R_k(\lambda) = \lambda R(\lambda)$$

Tutte queste formule sono sufficienti a risolvere in maniera analitica un modello con un carico di tipo transazionale (aperto) di parametro λ.

***Esempio*:**

Consideriamo la seguente rete di code aperta:



Sia λ = 0,3 job/sec per l'unica classe di clienti della rete di code, e siano dati:

k	CPU	DISK1	DISK2
V_k	121	70	50
S_k	0,005	0,030	0,027

da $D_k = V_k \times S_k$ si ricava:

k	CPU	DISK1	DISK2
D_k	0,605	2,100	1,350

da cui si possono calcolare gli indici di prestazione:

- $\lambda_{sat} = \dfrac{1}{D_{\max}} = \dfrac{1}{D_{\text{DISK1}}} = 0{,}476 \text{ job/sec}$
- $X_{\text{CPU}}(\lambda) = \lambda V_{\text{CPU}} = 0{,}3 \times 121 = 36{,}3 \text{ job/sec}$
- $X_{\text{DISK1}}(\lambda) = \lambda V_{\text{DISK1}} = 0{,}3 \times 70 = 21{,}0 \text{ job/sec}$
- $X_{\text{DISK2}}(\lambda) = \lambda V_{\text{DISK2}} = 0{,}3 \times 50 = 15{,}0 \text{ job/sec}$
- $R_{\text{CPU}}(\lambda) = \dfrac{D_{\text{CPU}}}{1\text{-}U_{\text{CPU}}(\lambda)} = \dfrac{D_{\text{CPU}}}{1\text{-}\lambda D_{\text{CPU}}} = \dfrac{0{,}605}{1-(0{,}3 \times 0{,}605)} = 0{,}740 \text{ sec}$
- $R_{\text{DISK1}}(\lambda) = \dfrac{D_{\text{DISK1}}}{1\text{-}U_{\text{DISK1}}(\lambda)} = \dfrac{D_{\text{DISK1}}}{1\text{-}\lambda D_{\text{DISK1}}} = \dfrac{2{,}1}{1-(0{,}3 \times 2{,}1)} = 5{,}676 \text{ sec}$
- $R_{\text{DISK2}}(\lambda) = \dfrac{D_{\text{DISK2}}}{1\text{-}U_{\text{DISK2}}(\lambda)} = \dfrac{D_{\text{DISK2}}}{1\text{-}\lambda D_{\text{DISK2}}} = \dfrac{1{,}35}{1-(0{,}3 \times 1{,}35)} = 2{,}269 \text{ sec}$
- $Q_{\text{CPU}}(\lambda) = \lambda R_{\text{CPU}}(\lambda) = 0{,}3 \times 0{,}740 = 0{,}222 \text{ job}$
- $Q_{\text{DISK1}}(\lambda) = \lambda R_{\text{DISK1}}(\lambda) = 0{,}3 \times 5{,}676 = 1{,}7028 \text{ job}$
- $Q_{\text{DISK2}}(\lambda) = \lambda R_{\text{DISK2}}(\lambda) = 0{,}3 \times 2{,}269 = 0{,}6807 \text{ job}$
- $R(\lambda) = \sum_{k=1}^{K} R_k(\lambda) = R_{\text{CPU}}(\lambda) + R_{\text{DISK1}}(\lambda) + R_{\text{DISK2}}(\lambda) \approx 8{,}685 \text{ sec}$
- $Q(\lambda) = \sum_{k=1}^{K} Q_k(\lambda) = Q_{\text{CPU}}(\lambda) + Q_{\text{DISK1}}(\lambda) + Q_{\text{DISK2}}(\lambda) \approx 2{,}6055 \text{ job}$ ▪

Algoritmo per reti di code in forma-prodotto aperte a singola classe di clienti:

```
Algorithm OpenSingleClassProductFormQN(λ,K,Sk,Vk,Tk)
    Input parameters:
        λ ∈ R+ with λ > 0; // is the external arrival rate
        K ∈ I+ with K > 0; // is the number of service center
        Sk ∈ R+ for 1 ≤ k ≤ K; // is the service time of k-th service center
        Vk ∈ I+ for 1 ≤ k ≤ K; // is the number of visits at the k-th service center
        Tk ∈ String for 1 ≤ k ≤ K; // is the service centers scheduling policy
    Local variables:
        k ∈ I+; // used in loops
        λsat ∈ R+; // store the computed saturation ratio for external arrival rate
        Dk ∈ R+ for 1 ≤ k ≤ K; // store the computed service demand for each service center
        Xk ∈ R+ for 1 ≤ k ≤ K; // store the computed throughput for each service center
        Uk ∈ R+ for 1 ≤ k ≤ K; // store the computed utilization for each service center
        Rk ∈ R+ for 1 ≤ k ≤ K; // store the computed residence time for each service center
        Qk ∈ R+ for 1 ≤ k ≤ K; // store the computed queue length for each service center
        R ∈ R+; // store the computed system response time
        Q ∈ R+; // store the computed average number of customer in system
begin
    // Compute the Service Demand for all K service centers
    for k:=1 to K do
        Dk := Vk × Sk;
        Output "Service Demand for the ",k,"-th service center = ",Dk;
    endfor;
    // Compute the saturation ratio for external arrival rate
    λsat := 0;
    for k:=1 to K do
        if (1 / Dk)>λsat then
                λsat := 1 / Dk;
        endif;
    endfor;
    Output "Saturation ratio for external arrival rate = ",λsat;

    // Compute throughput for all the K service centers
    for k:=1 to K do
        Xk := λ × Vk;
        Output "Throughput for the ",k,"-th service center is = ",Xk;
    endfor;
    // Compute utilization for all the K service centers
    for k:=1 to K do
        Uk := λ × Dk;
        Output "Utilization for the ",k,"-th service center is = ",Uk;
    endfor;
    // Compute residence time at each service center
    // also compute the system global response time
    R := 0;
    for k:=1 to K do
        if (Tk="IS") then
                Rk := Dk;
        elseif (Tk="FCFS") or (Tk="LCFSPR") or (Tk="PS") then
                Rk := Dk / (1 - Uk);
        endif
        R := R + Rk;
        Output "Residence time at ",k,"-th service center = ",Rk;
    endfor;
    Output "System Response Time = ",R;
    // Compute mean queue length at each service center
    // also compute the average number of customer in system
    Q := 0;
    for k:=1 to K do
        Qk := λ × Rk;
        Q := Q + Qk;
        Output "Mean Queue length at ",k,"-th service center = ",Qk;
    endfor;
    Output "Average number of customer in system = ",R;
end.
```

A.3.2 Sistemi chiusi (Batch o Interattivi) a singola classe di clienti

In un sistema chiuso non vi sono arrivi dal mondo esterno, né job che escono dalla rete di code. A differenza dei sistemi aperti, si ha una popolazione fissa di clienti che si indica con N e, come per il caso di sistemi aperti, K centri di servizio che possono essere centri di accodamento (FCFS,PS,LCFSPR) o di ritardo (IS). Sia inoltre definito Z come il tempo di ritardo (o *sleep time*), associato a classi di lavoro interattivo, che risulterà $Z = 0$, nel caso di lavori di tipo batch.
La tecnica che permette la soluzione di modelli chiusi a singola classe di clienti prende il nome di analisi al valor medio (*Mean Value Analysis* - MVA), e si basa sull'applicazione ripetuta delle seguenti formule:

- **tempo di residenza per singolo centro** di servizio:

$$R_k(N) = \begin{cases} D_k & \text{per disciplina IS} \\ D_k\left(1 + A_k(N)\right) & \text{per discipline FCFS, PS o LCFSPR} \end{cases}$$

  da cui si ricava il **tempo di residenza del sistema**:

$$R(N) = \sum_{k=1}^{K} R_k(N)$$

- **throughput del sistema**, che si ricava dall'applicazione della legge di Little alla rete di code nel suo complesso:

$$X(N) = \frac{N}{Z + R(N)}$$

  da cui si può ricavare il **throughput per il singolo centro** di servizio k:

$$X_k(N) = X(N)V_k$$

  l'**utilizzazione per il singolo centro** di servizio k:

$$U_k(N) = X(N)D_k$$

  il **tempo di risposta del sistema**:

$$R(N) = \frac{N}{X(N) - Z}$$

  numero medio di clienti/job nel sistema:

$$Q(N) = N - \left(Z \cdot X(N)\right)$$

- **lunghezza media della coda per singolo centro** di servizio:

$$Q_k(N) = X(N) \cdot R_k(N)$$

Notare che, così come per il caso di reti di code aperte, anche per il caso di reti di code chiuse, la chiave per poter calcolare tutte le misure del sistema sia la conoscenza di $A_k(N)$. Se conosciamo $A_k(N)$, possiamo calcolare $R_k(N)$, e quindi $X(N)$, $Q_k(N)$ e tutte le altre misure di interesse.
Nel caso di reti di code aperte è stato possibile effettuare la sostituzione della lunghezza della coda all'istante d'arrivo $A_k(N)$, con la lunghezza media della coda nel periodo di osservazione $Q_k(N)$. Nel caso di reti chiuse, questa sostituzione non è possibile.
Per verificare che in sistemi chiusi risulta: $A_k(N) \neq Q_k(N)$ è sufficiente trovare almeno un caso in cui questa disuguaglianza sia soddisfatta. Per dimostrare questa affermazione consideriamo una rete di code formata da due centri di servizio ed un solo cliente, con una domanda di servizio di 1sec ad ogni centro. Dato che c'è un solo cliente che circola nella rete, la lunghezza media della coda dei centri di servizio nel periodo di osservazione è pari alla loro utilizzazione:

$$Q_1(1) = Q_2(1) = 1/2$$

mentre risulterà:

$$A_1(1) = A_2(1) = 0$$

in quanto, essendoci un solo cliente in tutto il sistema, è impossibile che arrivando ad un centro esso veda di fronte a se qualche altro cliente in coda.
La differenza sostanziale tra le due grandezze è che la lunghezza della coda nell'istante di arrivo A_k(N) è calcolata dal punto di vista del cliente che arriva al centro (cioè mentre sta "arrivando" al centro), quindi non vi è ancora fisicamente, e non può rientrare nel computo degli utenti in coda (perché non può vedere se steso in coda). Mentre la lunghezza della coda nel periodo medio di osservazione Q_k(N) viene calcolata in base ad un istante di osservazione casuale, cioè da un punto di vista esterno alla rete. Così può capitare che tutti gli utenti si trovino, in quel istante, nel centro.

Risulta evidente che per risolvere un modello di reti di code chiuso, bisogna prima di tutto poter calcolare A_k(N). Ci sono due tecniche di base per effettuare questo calcolo, una prende il nome di soluzione esatta e l'altra soluzione approssimata. Queste due tecniche generano due tipi di algoritmi di risoluzione, uno prende il nome di algoritmo MVA esatto (Exact MVA), e l'altro prende il nome di algoritmo MVA approssimato. È da enfatizzare come la distinzione tra esatto e approssimato sia in relazione a come viene calcolata la soluzione del modello, e non al sistema modellato dalla rete di code. L'accuratezza della soluzione relativa alle performance del sistema dipende, quindi, non da quale dei due algoritmi si sceglie, ma dall'accuratezza della parametrizzazione del modello.

A.3.2.1 Soluzione esatta per modelli chiusi a singola classe di clienti

La tecnica di soluzione MVA esatta è importante per due ragioni:

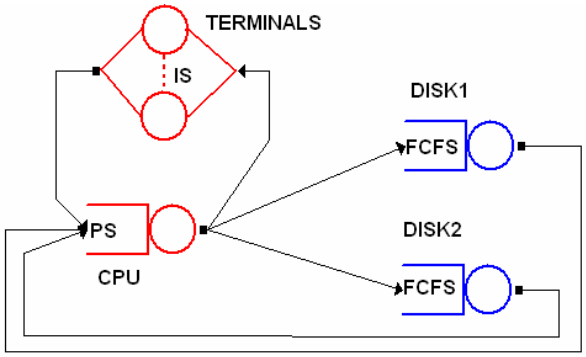
- è la base da cui viene derivata la tecnica soluzione approssimata;
- non vi sono limiti conosciuti relativamente all'esattezza della tecnica di soluzione approssimata. Anche se, tipicamente, la soluzione approssimata si discosta di pochi punti percentuali da quella esatta, non vi è alcuna garanzia che, in casi particolari, essa possa essere totalmente errata.

Il nostro obiettivo è calcolare A_k(N) in modo da poter derivare le altre misure di interesse per la rete di code. Abbiamo già detto che con A_k(N) si indica la lunghezza della coda vista dall'ultimo cliente (l'N-esimo) quando sta entrando nel centro di servizio *k*. Questo valore equivale al numero di clienti presenti in media al centro *k* quando i clienti complessivi non sono *N*, ma *N*-1. Per questo motivo, per reti di code chiuse separabili si può scrivere: $A_k(N) = Q_k(N-1)$.
Per ottenere la soluzione del modello con *N* clienti, si deve utilizzare un procedimento iterativo: con un solo cliente le code sono tutte vuote e quindi si può scrivere: $A_k(1) = Q_k(1-1) = 0$. Partendo da questo risultato, applicando le formule viste in precedenza si possono calcolare: il tempo di residenza ai singoli centri $R_k(1)$, il traffico $X(1)$ e le code ai diversi centri $Q_k(1)$. A questo punto conoscendo il valore di $A_k(2) = Q_k(1)$ si può procedere in modo iterativo fino al numero di clienti desiderato N.
Si noti come questo procedimento iterativo utilizzato per il calcolo del modello per N clienti, comporti anche la risoluzione del modello per 1,2,…,N-2,N-1 clienti.
La soluzione completa del modello per N clienti comporta quindi l'applicazione delle formule viste sopra per N clienti e per K centri, quindi la complessità dell'algoritmo cresce come il prodotto di N per K ovvero la complessità computazionale è dell'ordine di NK: $\Theta(NK)$. Mentre la complessità spaziale è dell'ordine di K: $\Theta(K)$, in quanto le misure ottenute per *n* clienti possono essere scartate quando sono state utilizzate per calcolare le misure per *n* +1 clienti.

Vediamo di seguito un esempio, e l'algoritmo per il calcolo della soluzione esatta di un modello di reti chiuse in forma-prodotto.

Esempio:

Consideriamo la seguente rete di code chiusa:



Sia Z = 15 sec per il delay center TERMINALS, e siano dati:

k	CPU	DISK1	DISK2
V_k	121	70	50
S_k	0,005	0,030	0,027

da $D_k = V_k \times S_k$ si ricava:

k	CPU	DISK1	DISK2
D_k	0,605	2,100	1,350

la seguente tabella mostra le varie fasi della soluzione del modello per N=3 clienti:

	k	***N=0***	***N=1***	***N=2***	***N=3***
$R_k(N) = \begin{cases} D_k \\ D_k(1+A_k(N)) \end{cases}$	**CPU**	-	0,605	0,624	**0,644**
	DISK1	-	2,100	2,331	**2,605**
	DISK2	-	1,350	1,446	**1,551**
$R(N) = \sum_{k=1}^{K} R_k(N)$	**SYSTEM**	-	4,055	4,401	**4,800**
$X(N) = \frac{N}{Z + R(N)}$	**SYSTEM**	-	0,0525	0,1031	**0,1515**
$Q(N) = N - (Z \cdot X(N))$	**SYSTEM**	-	0,2125	0,4535	**0,7275**
$X_k(N) = X(N)V_k$	**CPU**	-	6,353	12,475	**18,332**
	DISK1	-	3,675	7,217	**10,605**
	DISK2	-	2,625	5,155	**7,575**
$U_k(N) = X(N)D_k$	**CPU**	-	0,032	0,062	**0,092**
	DISK1	-	0,110	0,217	**0,318**
	DISK2	-	0,071	0,140	**0,205**
$A_k(N) = Q_k(N-1)$ dove $Q_k(N) = X(N) \cdot R_k(N)$	**CPU**	0	0,0318	0,0643	**0,0976**
	DISK1	0	0,1102	0,2403	**0,3947**
	DISK2	0	0,0708	0,1490	**0,2350**

Algoritmo Exact MVA, per reti di code chiuse in forma-prodotto a singola classe di clienti:

```
Algorithm ClosedSingleClassProductFormQN_Exact(N,K,Sk,Vk,Tk,Z) alias ExactMVA
    Input parameters:
        N ∈ I+ with N > 0; // is the number of job in the network
        K ∈ I+ with K > 0; // is the number of service center
        Sk ∈ R+ for 1 ≤ k ≤ K; // is the service time of k-th service center
        Vk ∈ I+ for 1 ≤ k ≤ K; // is the number of visits at the k-th service center
        Tk ∈ String for 1 ≤ k ≤ K; // is the service centers scheduling policy
        Z ∈ R+ for Z ≥ 0; // is the Think Time of the unique job class
    Local variables:
        k ∈ I+; // used in service center loops
        n ∈ I+; // used in customer loops
        Dk ∈ R+ for 1 ≤ k ≤ K; // store the computed service demand for each service center
        Xk ∈ R+ for 1 ≤ k ≤ K; // store the computed throughput for each service center
        Uk ∈ R+ for 1 ≤ k ≤ K; // store the computed utilization for each service center
        Rk ∈ R+ for 1 ≤ k ≤ K; // store the computed residence time for each service center
        Qk ∈ R+ for 1 ≤ k ≤ K; // store the computed queue length for each service center
        X ∈ R+; // store the computed system throughput
        R ∈ R+; // store the computed system response time
        Q ∈ R+; // store the computed average number of customer in system
begin
    // Compute the Service Demand for all K service centers
    for k:=1 to K do
        Dk := Vk × Sk;
        Output "Service Demand for the ",k,"-th service center = ",Dk;
    endfor;
    // Initialize Qk for step 0
    for k := 1 to K do
              Qk := 0;
    endfor;
    // Compute all model solution up to N customer
    for n := 1 to N do
        Output "Computing solution for the ",n," customers case"
        // Compute residence time at each service center
        // also compute the system global response time
        R := 0;
        for k:=1 to K do
              if (Tk="IS") then
                    Rk := Dk;
              elseif (Tk="FCFS") or (Tk="LCFSPR") or (Tk="PS") then
                    Rk := Dk × (1 + Qk);
              endif
              R := R + Rk;
              Output "Residence time at ",k,"-th service center = ",Rk;
        endfor;
        Output "System Response Time = ",R;
        // Compute system throughput
        X := n / (Z + R);
        Output "System Throughput = ",X;
        for k:=1 to K do
              // Compute throughput for all the K service centers
              Xk := X × Vk;
              Output "Throughput for the ",k,"-th service center is = ",Xk;
              // Compute utilization for all the K service centers
              Uk := X × Dk;
              Output "Utilization for the ",k,"-th service center is = ",Uk;
              // Compute mean queue length at each service center for the next step
              Qk := X × Rk;
              Output "Mean Queue length at ",k,"-th service center = ",Qk;
        endfor;
        // Compute the average number of customer in system
        Q := n - (Z × X);
        Output "System average number of customer = ",Q;
    endfor; // customer loop
end.
```

A.3.2.2 Soluzione approssimata per modelli chiusi a singola classe di clienti

La chiave dell'algoritmo MVA esatto è l'equazione:

$$A_k(\mathrm{N}) = Q_k(\mathrm{N\text{-}1})$$

che ci fornisce la lunghezza della coda all'istante d'arrivo per una popolazione di N clienti, dicendoci che è uguale alla lunghezza media della coda nel periodo di osservazione per una popolazione di N-1, N-2, N-3,…,2,1 clienti. Il calcolo può perciò risultare "pesante" per valori di N molto elevati. In questi casi ci si può "accontentare" di una soluzione approssimata, che richieda una quantità inferiore di calcoli.

Negli algoritmi MVA approssimati, si effettua un'approssimazione di $A_k(\mathrm{N})$ mediante una funzione di approssimazione h:

$$A_k(N) = h\left[Q_k(N)\right]$$

con lo scopo di ottenere un algoritmo più efficiente dal punto di vista computazionale.

Ovviamente l'accuratezza dell'algoritmo dipende dalla scelta della funzione h di cui in letteratura esistono svariate implementazioni. Un compendio di queste approssimazioni può essere trovato in [41]. Nella nostra trattazione useremo la seguente approssimazione:

$$A_k(N) = Q_k(N-1) \approx \frac{N-1}{N} Q_k(N)$$

Nota anche come approssimazione "Proportional Estimation PE", di Bard-Schweitzer [41][42]. Questa approssimazione deriva dall'idea che, per popolazioni molto grandi di clienti (N→∞), è plausibile attendersi che:

$$\frac{Q_k(N)}{N} \cong \frac{Q_k(N-1)}{N-1}$$

questa ipotesi è asintoticamente corretta, in quanto risulta:

$$\lim_{N\to\infty} \frac{Q_k(N)}{N} = \lim_{N\to\infty} \frac{Q_k(N-1)}{N-1}$$

Per innescare l'algoritmo MVA approssimato si può stabilire un qualsiasi valore iniziale per la lunghezza della coda ma, generalmente, per velocizzare la convergenza è meglio impostare:

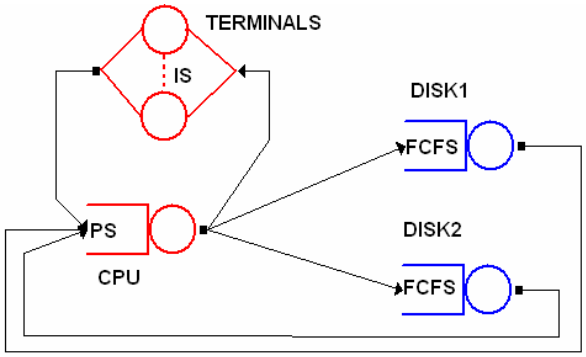
$$Q_k(N) = \frac{N}{K} \text{ per } 1 \le k \le K$$

cioè, si suppone che i clienti siano distribuiti uniformemente nei vari centri di servizio.

Successivamente si usano le solite formule per il calcolo del tempo di residenza dei singoli centri R_k e del sistema R, del throughput di sistema X e dei singoli centri X_k e per i nuovi valori (attuali) delle lunghezze delle code Q_k. Questi valori vengono confrontati con i precedenti, e se lo scostamento è superiore ad una certa soglia fissata arbitrariamente (ad esempio entro lo 0,1%), il procedimento viene ripetuto utilizzando i nuovi valori di Q_k. Se invece lo scostamento è inferiore, allora i valori ottenuti rappresentano la soluzione del modello.

Esempio:

Consideriamo la seguente rete di code chiusa:



Sia Z = 15 sec per il delay center TERMINALS, e siano dati:

k	CPU	DISK1	DISK2
V_k	121	70	50
S_k	0,005	0,030	0,027

da $D_k = V_k \times S_k$ si ricava:

k	CPU	DISK1	DISK2
D_k	0,605	2,100	1,350

Le formule che ci serviranno per il calcolo sono, nell'ordine:

$$1°)\ R_k(N) = \begin{cases} D_k \\ D_k\left(1 + A_k(N)\right) \end{cases} \text{ con } A_k(N) = \frac{N-1}{N} Q_k(N) \qquad (Q_k(N) \text{ del passo n-1})$$

$$2°)\ R(N) = \sum_{k=1}^{K} R_k(N)$$

$$3°)\ X(N) = \frac{N}{Z + R(N)}$$

$$4°)\ Q(N) = N - \left(Z \cdot X(N)\right)$$

$$5°)\ Q_k(N) = X(N) \cdot R_k(N)$$

Per l'innesco dell'algoritmo si userà $Q_k(N) = \frac{N}{K} = 1{,}0000 \quad \forall k \in K$. *La seguente tabella mostra le varie fasi della soluzione del modello per N=3 clienti e K=3 centri:*

Ciclo	R_{CPU}	R_{DISK1}	R_{DISK2}	R	X	Q	Q_{CPU}	Q_{DISK1}	Q_{DISK2}
0	-	-	-	-	-	-	1,0000	1,0000	1,0000
1	1,0083	3,5000	2,2500	6,7583	0,1379	0,9318	0,1390	0,4826	0,3102
2	0,6611	2,7756	1,6292	5,0659	0,1495	0,7574	0,0988	0,4150	0,2436
3	0,6449	2,6810	1,5692	4,8950	0,1508	0,7381	0,0972	0,4043	0,2366
4	0,6442	2,6660	1,5630	4,8732	0,1510	0,7356	0,0972	0,4024	0,2359
5	0,6442	2,6634	1,5623	4,8700	0,1510	0,7353	0,0973	0,4021	0,2359
Esatta	0,6440	2,6050	1,5510	4,8020	0,1515	0,7275	0,0976	0,3947	0,2350
Differenza	0,0002	0,0584	0,0113	0,0680	-0,0005	0,0078	-0,8787	0,0074	0,0009

Il calcolo dei valori della tabella procede dall'alto verso il basso e da sinistra a destra. La conoscenza di Q_k al passo n ci consente di calcolare R_k del passo n+1, poi R, X, Q ed infine la nuova stima di Q_k. Si verifica se la differenza tra i valori precedenti e attuali è inferiore ad una certa soglia e, se lo è, quella attuale è la soluzione cercata.

Algoritmo Approx MVA, per reti di code chiuse in forma-prodotto a singola classe di clienti:

```
Algorithm ClosedSingleClassProductFormQN_Approx(N,K,Sk,Vk,Tk,Z,h) alias ApproxMVA
    Input parameters:
        N ∈ I+ with N > 0; // is the number of job in the network
        K ∈ I+ with K > 0; // is the number of service center
        Sk ∈ R+ for 1 ≤ k ≤ K; // is the service time of k-th service center
        Vk ∈ I+ for 1 ≤ k ≤ K; // is the number of visits at the k-th service center
        Tk ∈ String for 1 ≤ k ≤ K; // is the service centers scheduling policy
        Z ∈ R+ for Z ≥ 0; // is the Think Time of the unique job class
    Local variables:
        k ∈ I+; // used in service center loops
        tol ∈ R+; // used as stop condition
        diff ∈ R+; // difference between previous and current computed values
        Dk ∈ R+ for 1 ≤ k ≤ K; // store the computed service demand for each service center
        Xk ∈ R+ for 1 ≤ k ≤ K; // store the computed throughput for each service center
        Uk ∈ R+ for 1 ≤ k ≤ K; // store the computed utilization for each service center
        Rk ∈ R+ for 1 ≤ k ≤ K; // store the computed residence time for each service center
        Qk ∈ R+ for 1 ≤ k ≤ K; // store the computed queue length for each service center
        X ∈ R+; // store the computed system throughput
        R ∈ R+; // store the computed system response time
        Q ∈ R+; // store the computed average number of customer in system
begin
    // Compute the Service Demand for all K service centers
    for k:=1 to K do
        Dk := Vk × Sk;
        Output "Service Demand for the ",k,"-th service center = ",Dk;
    endfor;
    // Initialize Qk for step 0
    for k := 1 to K do
            Qk := N / K;
    endfor;
    // Set the tolerance to agree to within 0.1%
    tol := 0.001;
    Do // Main Loop
        // Compute residence time at each service center
        R := 0; // also compute the system global response time
        for k:=1 to K do
            if (Tk="IS") then
                Rk := Dk;
            elseif (Tk="FCFS") or (Tk="LCFSPR") or (Tk="PS") then
                Rk := Dk × (1 + h(N,Qk));
                //for example in PE algorithm: h(N,Qk)::=((N-1)/N)×Qk
            endif
            R := R + Rk;
            Output "Residence time at ",k,"-th service center = ",Rk;
        endfor;
        Output "System Response Time = ",R;
        X := N / (Z + R); // Compute system throughput
        Output "System Throughput = ",X;
        Q := N - (Z × X); // Compute the average number of customer in system
        Output "System average number of customer = ",Q;
        diff := 0; // initialize difference at a low value
        for k:=1 to K do
            // Compute throughput for all the K service centers
            Xk := X × Vk;
            Output "Throughput for the ",k,"-th service center is = ",Xk;
            // Compute utilization for all the K service centers
            Uk := X × Dk;
            Output "Utilization for the ",k,"-th service center is = ",Uk;
            // Compute the max difference between actual and future values of Qk
            if abs((Qk - X×Rk)/Qk) > diff then
                diff := abs((Qk - X×Rk)/Qk);
            endif
            // Compute mean queue length at each service center for the next step
            Qk := X × Rk;
            Output "Mean Queue length at ",k,"-th service center = ",Qk;
        endfor;
    while (diff > tol); // if diff > tol then loop else exit
end.
```

A.3.3 Sistemi aperti (Transazionali) con più classi di clienti

Sia C il numero di classi di clienti presenti nel sistema. $\forall\ c$: $1 \le c \le C$, c è una classe di clienti aperta con una frequenza di arrivo λ_c.

Definiamo: Vettore delle frequenze d'arrivo, il vettore:

$$\vec{\lambda} \equiv (\lambda_1, \lambda_2, \ldots, \lambda_{C-1}, \lambda_C)$$

Dato che nei modelli aperti, il throughput delle classi è $X_c = \lambda_c$ per l'assunzione di bilanciamento del flusso, la risoluzione del modello risulta abbastanza agevole.

Prima di tutto, come per i modelli a singola classe, bisogna valutare se il modello è saturo, e per fare ciò andiamo a studiare la sua capacità di elaborazione (*processing capacity*).

Definiamo: Capacità di processo del sistema

> Si dice che un sistema ha la capacità di processare un dato carico di lavoro $\vec{\lambda}$, se è capace di farlo quando viene sottoposto a quel carico per un lungo periodo di tempo senza diventare saturo. Per modelli a classi multiple, la sufficienza è data dalla soddisfazione della seguente disuguaglianza:
>
> $$\max_{1 \le k \le K} \left\{ \sum_{c=1}^{C} \lambda_c D_{c,k} \right\} < 1$$
>
> la quale ci assicura che nessun centro di servizio è saturo, avendo valutato il carico combinato di tutte le classi.

Nei risultati che seguiranno assumeremo che la disuguaglianza vista sopra sia sempre soddisfatta.
Possiamo calcolare:

- il **throughput** della classe c al centro k, per la legge del flusso forzato viene calcolato in funzione di $\vec{\lambda}$:

$$X_{c,k}(\vec{\lambda}) = \lambda_c V_{c,k}$$

- l'**utilizzazione**, si calcola sfruttando la legge dell'utilizzazione:

$$U_{c,k}(\vec{\lambda}) = X_{c,k}(\vec{\lambda}) S_{c,k} = \lambda_c V_{c,k} S_{c,k} = \lambda_c D_{c,k}$$

- il **tempo di residenza**, come nei modelli a singola classe è dato da:

$$R_{c,k}(\vec{\lambda}) = \begin{cases} D_{c,k} & \text{per disciplina IS} \\ D_{c,k}\left(1 + A_{c,k}(\vec{\lambda})\right) & \text{per discipline FCFS, PS o LCFSPR} \end{cases}$$

  dove $A_{c,k}(\vec{\lambda})$ è il numero medio di clienti visti da un cliente della classe c in arrivo al centro k. L'intuizione alla base di questa formula è la stessa del caso di modelli a singola classe di clienti. Per i centri di ritardo (IS), il residence time equivale interamente al service demand $D_{c,k} = V_{c,k} S_{c,k}$. Per i centri di accodamento (FCFS,PS,LCFSPR) la giustificazione della formula dipende dalla disciplina. Per centri FCFS, il tempo di residenza è dato dal tempo di servizio $D_{c,k} = V_{c,k} S_{c,k}$ più il tempo di servizio dei clienti già in coda all'istante di arrivo $V_{c,k}\left(A_{c,k}(\vec{\lambda}) S_{c,k}\right)$, da cui $V_{c,k} S_{c,k} + V_{c,k}\left(A_{c,k}(\vec{\lambda}) S_{c,k}\right) = V_{c,k} S_{c,k}\left(1 + A_{c,k}(\vec{\lambda})\right) = D_{c,k}\left(1 + A_{c,k}(\vec{\lambda})\right)$. Per centri con disciplina PS, il tempo di residenza è dato dal tempo di servizio $D_{c,k} = V_{c,k} S_{c,k}$ inflazionato da un fattore $\left(1 + A_{c,k}(\vec{\lambda})\right)$, che rappresenta il "degrado" del service rate causato

dagli altri clienti in servizio che competono per accaparrarsi la risorsa di elaborazione. La giustificazione dell'espressione per centri LCFS non ha una spiegazione intuitiva, ma non per questo è meno valida delle precedenti.
Come per il caso di modelli a singola classe, aperti e chiusi, resta il problema di stimare $A_{c,k}(\vec{\lambda})$. Anche qui sfruttiamo la proprietà delle reti separabili vista in precedenza per cui, la lunghezza media vista da un cliente in arrivo deve essere uguale alla lunghezza media delle coda nel periodo di osservazione. E quindi possiamo stimare il tempo di residenza per i centri di accodamento mediante:

$$R_{c,k}(\vec{\lambda}) = D_{c,k}\left(1+Q_k(\vec{\lambda})\right)$$

dove con $Q_k(\vec{\lambda})$ si indica la lunghezza media della coda al centro *k* nel periodo di osservazione (come somma di $Q_{c,k}(\vec{\lambda})$ su tutte le classi *C*).
Applicando la legge di Little *N =XR* si ottiene:

$$R_{c,k}(\vec{\lambda}) = D_{c,k}\left(1+Q_k(\vec{\lambda})\right) = D_{c,k}\left(1+\sum_{j=1}^{C}\lambda_j R_{j,k}(\vec{\lambda})\right)$$

notare come il lato destro dell'equazione precedente dipenda da *c* solo per $D_{c,k}$.
Chiamiamo la parte indipendente da *c*, $\Delta = \left(1+\sum_{j=1}^{C}\lambda_j R_{j,k}(\vec{\lambda})\right)$, e consideriamo due classi di utenti generici *c* e *j*, risulterà:

$$\frac{R_{c,k}(\vec{\lambda})}{D_{c,k}} = \Delta$$

e

$$\frac{R_{j,k}(\vec{\lambda})}{D_{j,k}} = \Delta$$

perciò

$$\frac{R_{c,k}(\vec{\lambda})}{D_{c,k}} = \frac{R_{j,k}(\vec{\lambda})}{D_{j,k}}$$

da cui:

$$\frac{R_{c,k}(\vec{\lambda})}{R_{j,k}(\vec{\lambda})} = \frac{D_{c,k}}{D_{j,k}} \Rightarrow \frac{R_{c,k}(\vec{\lambda})}{R_{j,k}(\vec{\lambda})} = \frac{D_{c,k}}{D_{j,k}} \Rightarrow R_{j,k}(\vec{\lambda}) = \frac{D_{j,k}}{D_{c,k}} R_{c,k}(\vec{\lambda})$$

sostituendo questa espressione in quella del tempo di residenza si trova:

$$\begin{aligned} R_{c,k}(\vec{\lambda}) &= D_{c,k}\left(1+\sum_{j=1}^{C}\lambda_j R_{j,k}(\vec{\lambda})\right) = D_{c,k}\left(1+\sum_{j=1}^{C}\lambda_j \frac{D_{j,k}}{D_{c,k}} R_{c,k}(\vec{\lambda})\right) = \\ &= D_{c,k} + \frac{D_{c,k}}{D_{c,k}} R_{c,k}(\vec{\lambda})\sum_{j=1}^{C}\lambda_j D_{j,k} = D_{c,k} + R_{c,k}(\vec{\lambda})\sum_{j=1}^{C}\lambda_j D_{j,k} = \\ &= D_{c,k} + R_{c,k}(\vec{\lambda})\sum_{j=1}^{C} U_{j,k}(\vec{\lambda}) \end{aligned}$$

quindi riordinando i termini dell'equazione si ottiene:

$$R_{c,k}(\vec{\lambda}) = D_{c,k} + R_{c,k}(\vec{\lambda})\sum_{j=1}^{C} U_{j,k}(\vec{\lambda})$$

$$D_{c,k} = R_{c,k}(\vec{\lambda}) - R_{c,k}(\vec{\lambda})\sum_{j=1}^{C} U_{j,k}(\vec{\lambda})$$

$$D_{c,k} = R_{c,k}(\vec{\lambda})\left(1 - \sum_{j=1}^{C} U_{j,k}(\vec{\lambda})\right)$$

$$R_{c,k}(\vec{\lambda}) = \frac{D_{c,k}}{1 - \sum_{j=1}^{C} U_{j,k}(\vec{\lambda})}$$

che è l'espressione da utilizzare per il calcolo del tempo di residenza per centri di servizio FCFS, PS e LCFSPR. In sintesi risulta:

$$R_{c,k}(\vec{\lambda}) = \begin{cases} D_{c,k} & \text{per disciplina IS} \\ \dfrac{D_{c,k}}{1 - \sum_{j=1}^{C} U_{j,k}(\vec{\lambda})} & \text{per discipline FCFS, PS o LCFSPR} \end{cases}$$

- la **lunghezza della coda**, si ottiene applicando la legge di Little all'espressione del tempo di residenza appena trovata:

$$Q_{c,k}(\vec{\lambda}) = \lambda_c R_{c,k}(\vec{\lambda}) = \begin{cases} U_{c,k} & \text{per disciplina IS} \\ \dfrac{U_{c,k}}{1 - \sum_{j=1}^{C} U_{j,k}(\vec{\lambda})} & \text{per discipline FCFS, PS o LCFSPR} \end{cases}$$

- il **tempo di risposta del sistema per i clienti della classe *c***, si calcola sommando i tempi di risposta presso tutti i service center *K*:

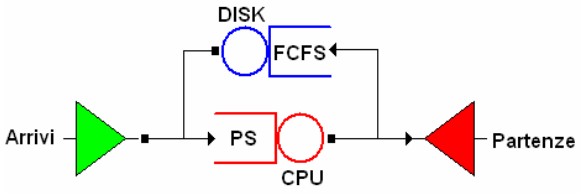
$$R_c(\vec{\lambda}) = \sum_{k=1}^{K} R_{c,k}(\vec{\lambda})$$

- il **numero medio di clienti della classe *c* nel sistema**, può essere calcolato usando la legge di Little, o sommando le lunghezze delle code dei clienti della classe *c* presso tutti i *K* service center:

$$Q_c(\vec{\lambda}) = \lambda_c R_c(\vec{\lambda}) = \sum_{k=1}^{K} Q_{c,k}(\vec{\lambda})$$

Esempio:

Consideriamo la seguente rete di code aperta:



Consideriamo due classi di clienti A e B tali che: λ_A*=3/19 job/sec e* λ_B*=2/19 job/sec. Con i seguenti valori di input:*

c	A		B	
k	CPU	DISK	CPU	DISK
$V_{c,k}$	10	9	5	4
$S_{c,k}$	1/10	1/3	2/5	1

da $D_{c,k} = V_{c,k} \times S_{c,k}$ *si ricava:*

c	A		B	
k	CPU	DISK	CPU	DISK
$D_{c,k}$	1	3	2	4

Calcoliamo la capacità di processo del sistema:

$$\max_{1\le k\le K}\left\{\sum_{c=1}^{C}\lambda_c D_{c,k}\right\} = \max_{1\le k\le K}\left\{\frac{3}{19}1+\frac{2}{19}2,\frac{3}{19}3+\frac{2}{19}4\right\} = \max_{1\le k\le K}\left\{\frac{7}{19},\frac{17}{19}\right\} = \max_{1\le k\le K}\{0.368421, 0.894737\}$$

ovvero:

$$\max_{1\le k\le K}\left\{\sum_{c=1}^{C}\lambda_c D_{c,k}\right\} = 0.894737 < 1$$

il sistema è stabile, si può proseguire.

La tabella seguente mostra il calcolo delle misure di performance; il calcolo avviene dall'alto verso il basso.

c	A		B	
k	CPU	DISK	CPU	DISK
$X_c(\vec{\lambda}) = \lambda_c$	0,1579		0,1053	
$X_{c,k}(\vec{\lambda}) = \lambda_c V_{c,k}$	1,5789	1,4211	0,5263	0,4211
$U_{c,k}(\vec{\lambda}) = \lambda_c D_{c,k}$	0,1579	0,4737	0,2105	0,4211
$R_{c,k}(\vec{\lambda}) = \begin{cases} D_{c,k} \\ \dfrac{D_{c,k}}{1-\sum_{j=1}^{C} U_{j,k}(\vec{\lambda})} \end{cases}$	1,5833	28,5171	3,1666	38,0228
$R_c(\vec{\lambda}) = \sum_{k=1}^{K} R_{c,k}(\vec{\lambda})$	30,1004		41,1894	
$Q_{c,k}(\vec{\lambda}) = \lambda_c R_{c,k}(\vec{\lambda})$	0,2500	4,5027	0,3333	4,0024
$Q_c(\vec{\lambda}) = \lambda_c R_c(\vec{\lambda})$	4,7527		4,3357	

Algoritmo per reti di code in forma-prodotto aperte con più classi di clienti:

```
Algorithm OpenMultipleClassProductFormQN(λc ,C,K,Sc,k ,Vc,k ,Tk )

    Input parameters:
        λc ∈ R+ for 1 ≤ c ≤ C with λc > 0; // is the external arrival rate for all classes
        C ∈ I+ with C > 0; // is the number of classes
        K ∈ I+ with K > 0; // is the number of service center
        Sc,k ∈ R+ for 1 ≤ k ≤ K , 1 ≤ c ≤ C;//is the service time of k service center of c class
        Vc,k ∈ I+ for 1 ≤ k ≤ K , 1 ≤ c ≤ C;//is the number of visits at the k service center of c class
        Tk ∈ String for 1 ≤ k ≤ K; // is the service centers scheduling policy

    Local variables:
        k, c, j ∈ I+; // used in loops
        MaxUtil ∈ R+; // store the system processing capacity
        psum ∈ R+; // temp variable
        Dc,k ∈ R+ for 1 ≤ k ≤ K , 1 ≤ c ≤ C;//service demand for each service center and class
        Uc,k ∈ R+ for 1 ≤ k ≤ K , 1 ≤ c ≤ C;//utilization for each service center and class
        Rc,k ∈ R+ for 1 ≤ k ≤ K , 1 ≤ c ≤ C;//residence time for each service center and class
        Qc,k ∈ R+ for 1 ≤ k ≤ K , 1 ≤ c ≤ C;//queue length for each service center and class
        Xc ∈ R+ for 1 ≤ c ≤ C; // system throughput for c class
        Rc ∈ R+ for 1 ≤ c ≤ C; // system response time for c class
        Qc ∈ R+ for 1 ≤ c ≤ C; // average number of customer in system for c class

begin

    // Compute the Service Demand for all K service centers over all the C classes
    for c:=1 to C do
        Output "Class ",c;
        for k:=1 to K do
                Dc,k := Vc,k × Sc,k;
                Output "Service Demand for the ",k,"-th service center = ",Dc,k;
        endfor k;
    endfor c;

    // check if system have sufficient processing capacity
    MaxUtil := 0;
    for k:=1 to K do
        psum := 0;
        for c:=1 to C do
                psum := psum + (λc × Dc,k);
        endfor c;
        if psum > MaxUtil then
                MaxUtil := psum;
        endif;
    endfor k;

    if MaxUtil < 1 then
        Output "System processing capacity = ",MaxUtil," < 1 The system is stable";
    else
        Output "System processing capacity = ",MaxUtil," >= 1 The system is not stable";
        Stop Execution;
    endif

    // Compute throughput for all the C classes
    for c:=1 to C do
        Xc := λc;
        Output "Throughput for ",c, " class = ",Xc;
    endfor c;

    // Compute utilization for all the C class, and K service centers
    for c:=1 to C do
        for k:=1 to K do
                Uc,k := λc × Dc,k;
                Output "Utilization for ",c, " class, ",k," serv.c. = ",Uc,k;
        endfor k;
    endfor c;
```

```
    // Compute residence time at each service center for all class
    // also compute the system response time for each class
    for c:=1 to C do
        Rc := 0;
        for k:=1 to K do
                if (Tk="IS") then
                        Rc,k := Dc,k;
                elseif (Tk="FCFS") or (Tk="LCFSPR") or (Tk="PS") then
                        psum := 0;
                        for j:=1 to C do
                                psum := psum + Uj,k
                        endfor j;
                        Rc,k := Dc,k / (1 - psum);
                endif
                Rc := Rc + Rc,k;
                Output "Residence time for ",c, " class, ",k," service center = ",Rc,k;
        endfor k;
        Output "System Response Time for ",c, " class = ",Rc;
    endfor c;

    // Compute mean queue length at each service center for all class
    // also compute the average number of customer in system for each class
    for c:=1 to C do
        Qc := 0;
        for k:=1 to K do
                Qc,k := λc × Rc,k;
                Qc := Qc + Qc,k;
                Output "Mean Queue length for ",c, " class, ",k," service center = ",Qc,k;
        endfor k;
        Output "Average number of customer in system for ",c, " class = ",Rc;
    endfor c;

end.
```

A.3.4 Sistemi chiusi (Batch o Interattivi) con più classi di clienti

Consideriamo una rete di code chiusa separabile con C classi di clienti, e K centri di servizio indipendenti dal carico (detti: *load independent service center*, o *fixed-rate service center*, o *single-server service center*).

Definiamo: il vettore delle popolazioni dei clienti come:

$$\vec{N} = (N_1, N_2, \ldots, N_C)$$

dove N_c è il numero di clienti che appartengono alla classe C per $c = 1, 2, \ldots, C$.

Si ha, che il numero totale di clienti nella rete di code è dato da:

$$N = N_1 + N_2 + \ldots + N_C$$

Come per i modelli aperti:

Definiamo: la domanda media di servizio (*service demand*) di un cliente della classe c al centro di servizio k come:

$$D_{c,k} \quad \text{dove } c = 1, 2, \ldots, C \text{ e } k = 1, 2, \ldots, K.$$

La domanda di servizio è definita come prodotto di due fattori: il tempo medio di servizio (*service time*) di un cliente della classe c durante ogni visita al centro k, e il numero medio di visite (*number of visits*) di un cliente della classe c al centro k:

$$D_{c,k} = S_{c,k} \times V_{c,k}$$

dove $c = 1, 2, \ldots, C$ e $k = 1, 2, \ldots, K$.

Quindi conoscendo due dei fattori è sempre possibile ricavare il terzo mediante una delle seguenti formule:

$$D_{c,k} = S_{c,k} \times V_{c,k}$$

$$V_{c,k} = \frac{D_{c,k}}{S_{c,k}}$$

$$S_{c,k} = \frac{D_{c,k}}{V_{c,k}}$$

Definiamo: Z_c il tempo di riflessione (*think time*) della classe c, come la somma delle domande di servizio (*service demand*) dei centri di ritardo (*delay center* anche detti *infinite-server*) visitati dai clienti della classe c. Ovviamente risulterà $Z_c = 0$ per classi di tipo batch.

Definiamo: $R_{c,k}(\vec{N})$ come il tempo medio di residenza (*average residence time*) dei clienti della classe c al centro k, dato il vettore della popolazione della rete $\vec{N}$.

Definiamo: $R_c(\vec{N})$ come il tempo medio di risposta (*average response time*) dei clienti della classe c, dato il vettore della popolazione della rete $\vec{N}$.

Definiamo: $X_c(\vec{N})$ come il tempo medio di attraversamento (*throughput*) della classe c, dato il vettore della popolazione della rete $\vec{N}$.

Definiamo: $Q_{c,k}(\vec{N})$ come la lunghezza media della coda (*mean queue length*) della classe *c* al centro *k*, dato il vettore della popolazione della rete $\vec{N}$.

Definiamo: $Q_k(\vec{N})$ come la lunghezza media totale della coda (*mean total queue length*) al centro *k*, dato il vettore della popolazione della rete $\vec{N}$.

La tecnica di soluzione, per questa classe di reti di code, è un'estensione dell'algoritmo MVA usato nel caso di reti a singola classe di clienti.

Tutto ruota intorno alle seguenti tre equazioni:

1) Per ogni classe, la legge di Little applicata all'intera rete di code ci da:
$$X_c(\vec{N}) = \frac{N_c}{Z_c + \sum_{k=1}^{K} R_{c,k}(\vec{N})}$$
2) Per ogni classe, la legge di Little applicata al singolo centro di servizio ci da:
$$Q_{c,k}(\vec{N}) = X_c(\vec{N}) R_{c,k}(\vec{N})$$
da cui si ricava la lunghezza della coda al centro *k*:
$$Q_k(\vec{N}) = \sum_{c=1}^{C} Q_{c,k}(\vec{N})$$
3) Per ogni classe, si ha la seguente espressione del tempo di residenza:
$$R_{c,k}(\vec{N}) = \begin{cases} D_{c,k} & \text{per discipline IS (delay centers)} \\ D_{c,k}\left(1 + A_{c,k}(\vec{N})\right) & \text{per discipline FCFS, LCFSPR, PS (queueing centers)} \end{cases}$$

Analizzando le tre equazioni si deduce che il flusso di calcolo prevede che si trovi:
$$A_{c,k}(\vec{N}) \rightarrow R_{c,k}(\vec{N}) \rightarrow X_c(\vec{N}) \rightarrow Q_{c,k}(\vec{N})$$
Quindi, come nei casi precedenti, le tre equazioni non possono essere valutate se prima non si da una stima di $A_{c,k}(\vec{N})$.

Anche per le reti chiuse con più classi di clienti esistono due approcci alla risoluzione del modello, uno esatto ed uno approssimato.

A.3.4.1 Soluzione esatta per modelli chiusi con più classi di clienti

Per ottenere la soluzione esatta di un modello chiuso multiclasse, si deve calcolare il valore esatto di $A_{c,k}(\vec{N})$. Trovato questo valore si possono trovare le equazioni 1, 2 e 3 e, quindi, la soluzione del modello. La chiave della tecnica di soluzione esatta MVA è la generalizzazione, al caso multiclasse, della relazione vista per il caso a singola classe:

$$A_{c,k}(\vec{N}) = Q_k(\overrightarrow{N-1_c})$$

dove il vettore:

$$\begin{aligned}\overrightarrow{N-1_c} &= \vec{N} - \vec{1_c} = \\ &= (N_1, N_2, \ldots, N_{c-1}, N_c, N_{c+1}, \ldots, N_C) - (0,0,\ldots,0,1,0,\ldots 0) = \\ &= (N_1, N_2, \ldots, N_{c-1}, N_c - 1, N_{c+1}, \ldots, N_C)\end{aligned}$$

ovvero $\overrightarrow{N-1_c}$ è il vettore popolazione $\vec{N}$ a cui è stato sottratto un cliente della classe *c*.
Intuitivamente questa relazione ci dice che la lunghezza della coda ad un centro *k*, vista da un cliente della classe *c* in arrivo, è uguale alla lunghezza media della coda nel periodo di osservazione senza il cliente in arrivo.

Si inizia con il trovare la soluzione banale per la rete di code con il vettore popolazione $\vec{N} = \vec{0}$, cioè:

$$Q_k(\vec{0}) = 0 \qquad \forall k : 1 \le k \le K$$

a questo punto possiamo usare ripetutamente le equazioni 3,1,2 e 3 per costruire iterativamente la soluzione, incrementando la popolazione ad ogni passo fino a trovare la soluzione per il vettore popolazione $\vec{N}$ desiderato.
È da notare come, in generale, per trovare la soluzione per una generica popolazione $\vec{n}$, siano richieste *C* soluzioni, una per ogni $\overrightarrow{n-1_c}$ con *c* = 1,2,…,*C*. Ad esempio se consideriamo una rete con 2 classi di clienti (A e B, la prima con 3 clienti e la seconda con 2 clienti), per trovare la soluzione di (3A,2B) prima si devono trovare tutte le soluzioni intermedie, come mostrato nella Figura A.12:

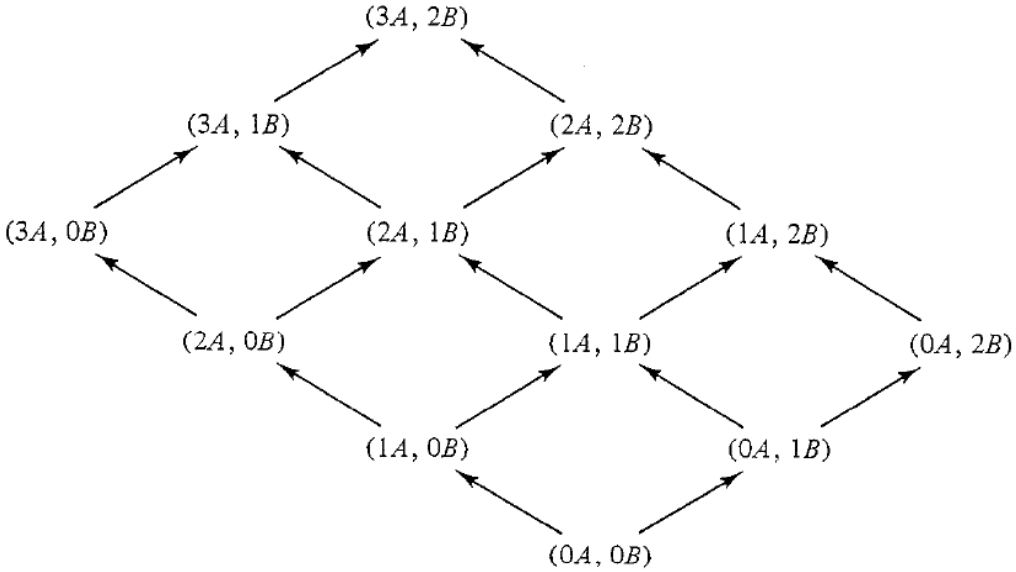


Figura A.12 - Precedenza delle soluzioni intermedie per Exact MVA multiclasse

Una conseguenza di questa dipendenza è l'elevata complessità dell'algoritmo Exact MVA nel caso di più classi di clienti, rispetto al caso a singola classe.

La complessità dell'algoritmo è infatti pari a:

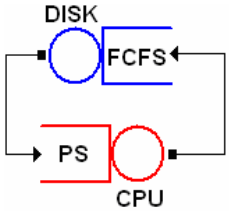
$$\text{tempo:} \quad \Theta\left(CK \prod_{c=1}^{C} (N_c + 1) \right) \quad \text{operazioni aritmetiche}$$

$$\text{spazio:} \quad \Theta\left(K \prod_{\substack{c=1 \\ c \neq c_{\max}}}^{C} (N_c + 1) \right) \quad \text{locazioni di memoria}$$

dove $c_{\max}$ è l'indice della classe con il maggior numero di clienti.
Una diretta conseguenza, di questa complessità computazionale, è che può risultare improponibile calcolare la soluzione esatta per una rete di code che abbia un numero elevato di classi e di clienti.
Ad esempio la soluzione esatta per una rete di code con K = 10 centri, C = 5 classi con 10 clienti l'una, richiede più di 8.000.000 di operazioni aritmetiche e circa 145.000 locazioni di memoria.

Esempio:

Consideriamo la seguente rete di code chiusa:



Consideriamo due classi di clienti A e B tali che: $N_A=1$ *job e* $N_B=1$ *job, cioè* $\vec{N}=(1_A,1_B)$*. Con i seguenti valori di input:*

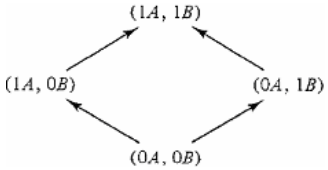
c	A		B	
k	CPU	DISK	CPU	DISK
$V_{c,k}$	10	9	5	4
$S_{c,k}$	1/10	1/3	2/5	1

da $D_{c,k} = V_{c,k} \times S_{c,k}$ *si ricava:*

c	A		B	
k	CPU	DISK	CPU	DISK
$D_{c,k}$	1	3	2	4

Trattandosi di classi batch risulta $\vec{Z} = (0_A,0_B)$*.*

Il procedimento di calcolo prevede di dover calcolare le soluzioni intermedie:



Si ha la seguente tabella:

$\vec{N}$	$(0_A,0_B)$	$(1_A,0_B)$	$(0_A,1_B)$	$(1_A,1_B)$
$R_{A,CPU}$	-	*1*	-	*4/3*
$R_{A,DISK}$	-	*3*	-	*5*
$R_{B,CPU}$	-	-	*2*	*5/2*
$R_{B,DISK}$	-	-	*4*	*7*
X_A	-	*1/4*	-	*3/19*
X_B	-	-	*1/6*	*2/19*
$Q_{A,CPU}$	*0*	*1/4*	-	*4/19*
$Q_{A,DISK}$	*0*	*3/4*	-	*15/19*
$Q_{B,CPU}$	*0*	-	*1/3*	*5/19*
$Q_{B,DISK}$	*0*	-	*2/3*	*14/19*

il calcolo dei valori nella tabella, avviene dall'alto verso il basso e da sinistra verso destra.

Algoritmo di risoluzione esatta per reti di code in forma-prodotto chiuse con più classi di clienti:

```
Algorithm ClosedMultipleClassProductFormQN(Nc , Zc , C , K , Sc,k ,Vc,k ,Tk ) alias ExactMVA
    Input parameters:
        Nc ∈ I+ for 1 ≤ c ≤ C with Nc > 0; // is the number of job for each class
        Zc ∈ I+ for 1 ≤ c ≤ C with Zc ≥ 0; // is the c class think time
        C ∈ I+ with C > 0; // is the number of classes
        K ∈ I+ with K > 0; // is the number of service center
        Sc,k ∈ R+ for 1 ≤ k ≤ K , 1 ≤ c ≤ C;//is the service time of k service center of c class
        Vc,k ∈ I+ for 1 ≤ k ≤ K , 1 ≤ c ≤ C;//is the number of visits at the k service center of c class
        Tk ∈ String for 1 ≤ k ≤ K; // is the service centers scheduling policy
    Local variables:
        k, c, n ∈ I+; // used in loops
        N ∈ I+; // total network population
        nc ∈ R+ for 1 ≤ c ≤ C;//used in loop for increasing population
        Dc,k ∈ R+ for 1 ≤ k ≤ K , 1 ≤ c ≤ C;//service demand for each service center and class
        Rc,k ∈ R+ for 1 ≤ k ≤ K , 1 ≤ c ≤ C;//residence time for each service center and class
        Xc ∈ R+ for 1 ≤ c ≤ C; // system throughput for c class
        Rc ∈ R+ for 1 ≤ c ≤ C; // system response time for c class
        Qk ∈ R+ for 1 ≤ k ≤ K; // average number of customer in the k service center for all classes
begin
    // Compute the Service Demand for all K service centers over all the C classes
    N := 0; // also use C loop to compute total population N
    for c:=1 to C do
        Output "Class ",c;
        for k:=1 to K do
                Dc,k := Vc,k × Sc,k;
                Output "Service Demand for the ",k,"-th service center = ",Dc,k;
        endfor k;
        N := N + Nc;
    endfor c;
    // Initialize queue length for empty solution N=0
    for k:=1 to K do
        Qk := 0;
    endfor k;
    // Main loop for increasing population
    for n:=1 to N do
        foreach feasible population vector [n1,…,nc] with ∑nc(1 ≤ c ≤ C) = n do
                for c:=1 to C do
                        Rc := 0;
                        for k:=1 to K do
                                if (Tk="IS") then
                                        Rc,k := Dc,k;
                                elseif (Tk="FCFS") or (Tk="LCFSPR") or (Tk="PS") then
                                        Rc,k := Dc,k × (1 + Qk);
                                endif
                                Rc := Rc + Rc,k;
                        endfor k;
                        Xc = nc / (Zc + Rc);
                endfor c;
                for k:=1 to K do
                        Qk := 0;
                        for c:=1 to C do
                                Qk := Qk + ( Xc × Rc,k )
                        endfor c;
                endfor k;
        endforeach;
    endfor n;
    // when you are here all system, per class, metrics are available:
    for c:=1 to C do
        Output "Class ",c," system throughput       = ",Xc;
        Output "Class ",c," system response time    = ", Nc / ( Xc - Zc );
        Output "Class ",c," average number of jobs = ", Nc - ( Xc × Zc );
        for k:=1 to K do
                Output "Class ",c," throughput at dev. ",k," = ", Xc × Vc,k;
                Output "Class ",c," utilization at dev. ",k," = ", Xc × Dc,k;
                Output "Class ",c," queue length at dev. ",k," = ", Xc × Rc,k;
                Output "Class ",c," residence time at dev. ",k," = ", Rc,k;
        endfor k;
    endfor c;
end.
```

A.3.4.2 Come generare tutti i possibili vettori popolazione

Nell'algoritmo precedente c'è un ciclo di enumerazione di tutti i possibili vettori popolazione. Questo non è un compito facile da effettuare, e potrebbe creare qualche difficoltà a chi volesse implementare l'algoritmo. Per questo si riporta di seguito un frammento di codice PHP che genera tutti i possibili vettori popolazione, dati C ed $\vec{N}$.
Il principale vantaggio di PHP è che supporta in modo nativo i vettori dinamici (senza dimensione fissata a priori), e che dispone di una funzione per la somma degli elementi di un vettore. Tuttavia per un bravo programmatore non dovrebbe essere difficile convertire questo codice in qualche altro linguaggio:

```
<?php
// Questa funzione stampa un vettore passato come parametro in formato comprensibile
function mypv($v)
{
	echo " [ ";
	foreach ($v as $ve) { echo $ve," "; };
	echo "] = ",array_sum($v),"\n";
};
// Inputs del programma:
$C = 2; // Numero massimo di classi
$N = array(1=> 3 ,2=> 2 ); // Vettore con il massimo numero di Job per classe
// Il vettore popolazione di Output è un vettore di vettori dinamico
$P = array(array());
// il vettore $tp è un vettore di lavoro temporaneo inizializzato a zero
$tp = array();
for ($i=1; $i<=$C; $i++) { $tp[$i] = 0; };
// Calcolo la popolazione totale
$pop = array_sum($N);
echo "Max POP: ",$pop,"\n";
// Calcolo il numero atteso di vettori popolazione
$maxvect = 1;
for ($i=1; $i<=$C; $i++) { $maxvect = $maxvect * ($N[$i]+1); };
echo "Numero atteso di vettori: $maxvect\n\n";
// Questo è il ciclo principale di calcolo
$i = 1;
while(array_sum($tp)<=array_sum($N))
{
	$P[array_sum($tp)][]=$tp;
	mypv($tp);
	$tp[$i]++;
	if ($tp[$i]>$N[$i])
	{
		while (($tp[$i]>$N[$i]) and ($i<$C) and (array_sum($tp)<=array_sum($N)))
		{
			$tp[$i] = 0;
			$i++;
			$tp[$i]++;
		};
		$i = 1;
	};
};
// All'uscita da questo ciclo $P contiene tutti i
// possibili vettori popolazione ordinati per lunghezza

// Stampo $P a video
$j=0;
for ($i=0; $i<=$pop; $i++)
{
	echo "Vettori di lunghezza $i:\n";
	foreach ($P[$i] as $v)
	{
		mypv($v);
		$j++;
	};
	echo "\n";
};
echo "Totale vettori stampati: $j\n";

?>
```

Vediamo di seguito alcuni esempi di esecuzione per classi e popolazioni differenti:

Input:

```
$C = 2;
$N = array(1=> 3 ,2=> 2 );
```

Output:

```
Max POP: 5
Numero atteso di vettori: 12
Vettori di lunghezza 0:
   [ 0 0 ] = 0
Vettori di lunghezza 1:
   [ 1 0 ] = 1
   [ 0 1 ] = 1
Vettori di lunghezza 2:
   [ 2 0 ] = 2
   [ 1 1 ] = 2
   [ 0 2 ] = 2
Vettori di lunghezza 3:
   [ 3 0 ] = 3
   [ 2 1 ] = 3
   [ 1 2 ] = 3
Vettori di lunghezza 4:
   [ 3 1 ] = 4
   [ 2 2 ] = 4
Vettori di lunghezza 5:
   [ 3 2 ] = 5
Totale vettori stampati: 12
```

Input:

```
$C = 2;
$N = array(1=> 2 ,2=> 3 );
```

Output:

```
Max POP: 5
Numero atteso di vettori: 12
Vettori di lunghezza 0:
   [ 0 0 ] = 0
Vettori di lunghezza 1:
   [ 1 0 ] = 1
   [ 0 1 ] = 1
Vettori di lunghezza 2:
   [ 2 0 ] = 2
   [ 1 1 ] = 2
   [ 0 2 ] = 2
Vettori di lunghezza 3:
   [ 2 1 ] = 3
   [ 1 2 ] = 3
   [ 0 3 ] = 3
Vettori di lunghezza 4:
   [ 2 2 ] = 4
   [ 1 3 ] = 4
Vettori di lunghezza 5:
   [ 2 3 ] = 5
Totale vettori stampati: 12
```

Input:

```
$C = 3;
$N = array(1=> 3 ,2=> 2, 3=> 1);
```

Output:

```
Max POP: 6
Numero atteso di vettori: 24
Vettori di lunghezza 0:
   [ 0 0 0 ] = 0
Vettori di lunghezza 1:
   [ 1 0 0 ] = 1
   [ 0 1 0 ] = 1
   [ 0 0 1 ] = 1
Vettori di lunghezza 2:
   [ 2 0 0 ] = 2
   [ 1 1 0 ] = 2
   [ 0 2 0 ] = 2
   [ 1 0 1 ] = 2
   [ 0 1 1 ] = 2
Vettori di lunghezza 3:
   [ 3 0 0 ] = 3
   [ 2 1 0 ] = 3
   [ 1 2 0 ] = 3
   [ 2 0 1 ] = 3
   [ 1 1 1 ] = 3
   [ 0 2 1 ] = 3
Vettori di lunghezza 4:
   [ 3 1 0 ] = 4
   [ 2 2 0 ] = 4
   [ 3 0 1 ] = 4
   [ 2 1 1 ] = 4
   [ 1 2 1 ] = 4
Vettori di lunghezza 5:
   [ 3 2 0 ] = 5
   [ 3 1 1 ] = 5
   [ 2 2 1 ] = 5
Vettori di lunghezza 6:
   [ 3 2 1 ] = 6
Totale vettori stampati: 24
```

Input:

```
$C = 3;
$N = array(1=> 2 ,2=> 1, 3=> 2);
```

Output:

```
Max POP: 5
Numero atteso di vettori: 18
Vettori di lunghezza 0:
   [ 0 0 0 ] = 0
Vettori di lunghezza 1:
   [ 1 0 0 ] = 1
   [ 0 1 0 ] = 1
   [ 0 0 1 ] = 1
Vettori di lunghezza 2:
   [ 2 0 0 ] = 2
   [ 1 1 0 ] = 2
   [ 1 0 1 ] = 2
   [ 0 1 1 ] = 2
   [ 0 0 2 ] = 2
Vettori di lunghezza 3:
   [ 2 1 0 ] = 3
   [ 2 0 1 ] = 3
   [ 1 1 1 ] = 3
   [ 1 0 2 ] = 3
   [ 0 1 2 ] = 3
Vettori di lunghezza 4:
   [ 2 1 1 ] = 4
   [ 2 0 2 ] = 4
   [ 1 1 2 ] = 4
Vettori di lunghezza 5:
   [ 2 1 2 ] = 5
Totale vettori stampati: 18
```

Input:

```
$C = 4;
$N = array(1=> 3 ,2=> 1, 3=> 2, 4=> 1);
```

Output:

```
Max POP: 7
Numero atteso di vettori: 48
Vettori di lunghezza 0:
   [ 0 0 0 0 ] = 0
Vettori di lunghezza 1:
   [ 1 0 0 0 ] = 1
   [ 0 1 0 0 ] = 1
   [ 0 0 1 0 ] = 1
   [ 0 0 0 1 ] = 1
Vettori di lunghezza 2:
   [ 2 0 0 0 ] = 2
   [ 1 1 0 0 ] = 2
   [ 1 0 1 0 ] = 2
   [ 0 1 1 0 ] = 2
   [ 0 0 2 0 ] = 2
   [ 1 0 0 1 ] = 2
   [ 0 1 0 1 ] = 2
   [ 0 0 1 1 ] = 2
Vettori di lunghezza 3:
   [ 3 0 0 0 ] = 3
   [ 2 1 0 0 ] = 3
   [ 2 0 1 0 ] = 3
   [ 1 1 1 0 ] = 3
   [ 1 0 2 0 ] = 3
   [ 0 1 2 0 ] = 3
   [ 2 0 0 1 ] = 3
   [ 1 1 0 1 ] = 3
   [ 1 0 1 1 ] = 3
   [ 0 1 1 1 ] = 3
   [ 0 0 2 1 ] = 3
Vettori di lunghezza 4:
   [ 3 1 0 0 ] = 4
   [ 3 0 1 0 ] = 4
   [ 2 1 1 0 ] = 4
   [ 2 0 2 0 ] = 4
   [ 1 1 2 0 ] = 4
   [ 3 0 0 1 ] = 4
   [ 2 1 0 1 ] = 4
   [ 2 0 1 1 ] = 4
   [ 1 1 1 1 ] = 4
   [ 1 0 2 1 ] = 4
   [ 0 1 2 1 ] = 4
Vettori di lunghezza 5:
   [ 3 1 1 0 ] = 5
   [ 3 0 2 0 ] = 5
   [ 2 1 2 0 ] = 5
   [ 3 1 0 1 ] = 5
   [ 3 0 1 1 ] = 5
   [ 2 1 1 1 ] = 5
   [ 2 0 2 1 ] = 5
   [ 1 1 2 1 ] = 5
Vettori di lunghezza 6:
   [ 3 1 2 0 ] = 6
   [ 3 1 1 1 ] = 6
   [ 3 0 2 1 ] = 6
   [ 2 1 2 1 ] = 6
Vettori di lunghezza 7:
   [ 3 1 2 1 ] = 7
Totale vettori stampati: 48
```

Input:

```
$C = 4;
$N = array(1=> 2 ,2=> 1, 3=> 2, 4=> 2);
```

Output:

```
Max POP: 7
Numero atteso di vettori: 54
   [ 0 0 0 0 ] = 0
   [ 1 0 0 0 ] = 1
   [ 0 1 0 0 ] = 1
   [ 0 0 1 0 ] = 1
   [ 0 0 0 1 ] = 1
   [ 2 0 0 0 ] = 2
   [ 1 1 0 0 ] = 2
   [ 1 0 1 0 ] = 2
   [ 0 1 1 0 ] = 2
   [ 0 0 2 0 ] = 2
   [ 1 0 0 1 ] = 2
   [ 0 1 0 1 ] = 2
   [ 0 0 1 1 ] = 2
   [ 0 0 0 2 ] = 2
   [ 2 1 0 0 ] = 3
   [ 2 0 1 0 ] = 3
   [ 1 1 1 0 ] = 3
   [ 1 0 2 0 ] = 3
   [ 0 1 2 0 ] = 3
   [ 2 0 0 1 ] = 3
   [ 1 1 0 1 ] = 3
   [ 1 0 1 1 ] = 3
   [ 0 1 1 1 ] = 3
   [ 0 0 2 1 ] = 3
   [ 1 0 0 2 ] = 3
   [ 0 1 0 2 ] = 3
   [ 0 0 1 2 ] = 3
   [ 2 1 1 0 ] = 4
   [ 2 0 2 0 ] = 4
   [ 1 1 2 0 ] = 4
   [ 2 1 0 1 ] = 4
   [ 2 0 1 1 ] = 4
   [ 1 1 1 1 ] = 4
   [ 1 0 2 1 ] = 4
   [ 0 1 2 1 ] = 4
   [ 2 0 0 2 ] = 4
   [ 1 1 0 2 ] = 4
   [ 1 0 1 2 ] = 4
   [ 0 1 1 2 ] = 4
   [ 0 0 2 2 ] = 4
   [ 2 1 2 0 ] = 5
   [ 2 1 1 1 ] = 5
   [ 2 0 2 1 ] = 5
   [ 1 1 2 1 ] = 5
   [ 2 1 0 2 ] = 5
   [ 2 0 1 2 ] = 5
   [ 1 1 1 2 ] = 5
   [ 1 0 2 2 ] = 5
   [ 0 1 2 2 ] = 5
   [ 2 1 2 1 ] = 6
   [ 2 1 1 2 ] = 6
   [ 2 0 2 2 ] = 6
   [ 1 1 2 2 ] = 6
   [ 2 1 2 2 ] = 7
Totale vettori stampati: 54
```

L'algoritmo funziona anche per il caso a singola classe $C = 1$ e $\vec{N} = [3]$ con il seguente output:

```
Max POP: 3
Numero atteso di vettori: 4
   [ 0 ] = 0
   [ 1 ] = 1
   [ 2 ] = 2
   [ 3 ] = 3
Totale vettori stampati: 4
```

A.3.4.3 Soluzione approssimata per modelli chiusi con più classi di clienti

Dato che la soluzione esatta richiede tempo e spazio eccessivi per modelli con molte classi di clienti, la soluzione approssimata è spesso l'unica che può essere applicata.
Il principale vantaggio della soluzione approssimata è che non richiede il calcolo di tutte le possibili soluzioni intermedie.
Operativamente la tecnica di soluzione per modelli chiusi con più classi di clienti non è altro che un'estensione del caso a singola classe di clienti.
La soluzione mediante approssimazione si basa sulle stesse equazioni 1,2 e 3 della tecnica esatta, ma con il significativo vantaggio di lavorare sul vettore popolazione completo $\vec{N}$, invece di costruire la soluzione partendo dal vettore popolazione vuoto. Per questo, la tecnica di soluzione approssimata richiede molto meno spazio di quella esatta, in quanto deve memorizzare solo un vettore popolazione $\vec{N}$. In particolare, lo spazio richiesto è proporzionale al prodotto di *C* e *K*.
Per quanto riguarda la velocità di esecuzione è più difficile dare una stima del vantaggio ,tuttavia, da misurazioni empiriche, questo vantaggio risulta essere notevole. Il numero di operazioni richieste per la soluzione del modello è ancora una volta proporzionale a *C* e *K*, mentre il numero di clienti delle singole classi non influiscono sul processo di soluzione.
Come nel caso a singola classe, la soluzione approssimata viene costruita stimando la lunghezza media delle code all'istante d'arrivo ad ogni centro di servizio $A_{c,k}(\vec{N})$.
La chiave dell'algoritmo è data da una funzione di approssimazione h_c:

$$A_{c,k}(\vec{N}) = Q_k(\overrightarrow{N-1_c}) \cong h_c\left(Q_{1,k}(\vec{N}),\ldots,Q_{C,k}(\vec{N})\right)$$

L'accuratezza dell'algoritmo e della soluzione trovata dipende, come nel caso a singola classe, dalla funzione di approssimazione scelta.
Quella che in letteratura ha riscosso il maggior successo è la versione multiclasse dell' approssimazione "Proportional Estimation PE", di Bard-Schweitzer [41][42]:

$$h_c\left(Q_{1,k}(\vec{N}),\ldots,Q_{C,k}(\vec{N})\right) = \left[\frac{N_c - 1}{N_c} Q_{c,k}(\vec{N})\right] + \sum_{\substack{j=1 \\ j\neq c}}^{C} Q_{j,k}(\vec{N})$$

come nel caso a singola classe, essa si basa sull'idea che per popolazioni molto grandi di clienti ($N_c \to \infty \ \forall c$), è plausibile attendersi che:

$$\frac{Q_{c,k}(\vec{N})}{\vec{N}} \cong \frac{Q_{c,k}(\overrightarrow{N-1_c})}{\overrightarrow{N-1_c}}$$

e, questa ipotesi risulta essere corretta in quanto, passando ai limiti risulta:

$$\lim_{\vec{N}\to\vec{\infty}} \frac{Q_{c,k}(\vec{N})}{\vec{N}} = \lim_{\vec{N}\to\vec{\infty}} \frac{Q_{c,k}(\overrightarrow{N-1_c})}{\overrightarrow{N-1_c}}$$

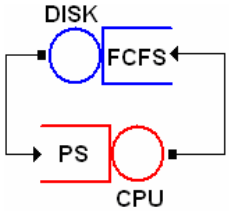
Per innescare l'algoritmo MVA approssimato multiclasse, si può stabilire un qualsiasi valore iniziale per la lunghezza della coda, ma generalmente, per velocizzare la convergenza è meglio impostare:

$$Q_{c,k}(\vec{N}) = \frac{N_c}{K} \text{ per } 1 \le k \le K \text{ e } 1 \le c \le C$$

cioè, si suppone che i clienti delle varie classi siano distribuiti uniformemente nei vari centri di servizio.

Esempio:

Consideriamo la seguente rete di code chiusa:



Consideriamo due classi di clienti A e B tali che: $N_A=1$ *job e* $N_B=1$ *job, cioè* $\vec{N}=(1_A,1_B)$*. Con i seguenti valori di input:*

c	A		B	
k	CPU	DISK	CPU	DISK
$V_{c,k}$	10	9	5	4
$S_{c,k}$	1/10	1/3	2/5	1

da $D_{c,k} = V_{c,k} \times S_{c,k}$ *si ricava:*

c	A		B	
k	CPU	DISK	CPU	DISK
$D_{c,k}$	1	3	2	4

Trattandosi di classi batch risulta $\vec{Z} = (0_A,0_B)$*.*

Si ha la seguente tabella di risoluzione:

Iterazione	Classe	Misure di Performance			
		$Q_{c,CPU}$	$Q_{c,DISK}$	X_c	R_c
0	A	½	½	-	-
	B	½	½	-	-
1	A	0,250	0,750	0,167	6,000
	B	0,333	0,667	0,111	9,000
2	A	0,211	0,790	0,158	6,333
	B	0,263	0,737	0,105	9,500
3	A	0,195	0,805	0,154	6,474
	B	0,253	0,747	0,104	9,579
4	A	0,193	0,807	0,154	6,495
	B	0,249	0,751	0,104	9,610
5	A	0,192	0,808	0,154	6,508
	B	0,248	0,752	0,104	9,614
Soluzione esatta	A	0,211	0,789	0,158	6,333
	B	0,263	0,737	0,105	9,500

il calcolo dei valori della tabella avviene dall'alto verso il basso e da sinistra verso destra.

Algoritmo Approx MVA, per reti di code chiuse in forma-prodotto con più classi di clienti:

```
Algorithm ClosedMultipleClassProductFormQN_Approx(Nc, K, Sc,k, Vc,k, Tk, Zc, hc) alias ApproxMVA
    Input parameters:
        Nc ∈ I+ for 1 ≤ c ≤ C with Nc > 0; // is the number of c class jobs in the network
        K ∈ I+ with K > 0; // is the number of service center
        Sc,k ∈ R+ for 1 ≤ c ≤ C, 1 ≤ k ≤ K; // service time of the k service center for c class jobs
        Vc,k ∈ I+ for 1 ≤ c ≤ C, 1 ≤ k ≤ K; // number of visits at the k service center for c class jobs
        Tk ∈ String for 1 ≤ k ≤ K; // is the service centers scheduling policy
        Zc ∈ R+ for 1 ≤ c ≤ C with Z ≥ 0; // is the Think Time of the c class jobs
    Local variables:
        c ∈ I+; // used in class loops
        k ∈ I+; // used in service center loops
        tol ∈ R+; // used as stop condition
        diffc ∈ R+ for 1 ≤ c ≤ C; // difference between previous and current computed values
        Dc,k ∈ R+ for 1 ≤ c ≤ C, 1 ≤ k ≤ K; // store the computed service demand for each service center
        Xk ∈ R+ for 1 ≤ k ≤ K; // store the computed throughput for each service center
        Uc,k ∈ R+ for 1 ≤ c ≤ C, 1 ≤ k ≤ K; // store the computed utilization for each service center
        Rc,k ∈ R+ for 1 ≤ c ≤ C, 1 ≤ k ≤ K; // store the computed residence time for each service center
        Qc,k ∈ R+ for 1 ≤ c ≤ C, 1 ≤ k ≤ K; // store the computed queue length for each service center
        Xc ∈ R+; // store the computed system throughput
        Rc ∈ R+; // store the computed system response time
        Qc ∈ R+; // store the computed average number of customer in system
begin
    // Compute the Service Demand for all C class and K service centers
    for c:=1 to C do
        for k:=1 to K do
                Dc,k := Vc,k × Sc,k;
                Output "Service Demand for class ",c," at ",k,"-th service center = ",Dc,k;
        endfor k;
    endfor c;
    // Initialize Qc,k for step 0
    for c := 1 to C do
        for k := 1 to K do
                Qc,k := Nc / K;
        endfor k;
    endfor c;
    tol := 0.001; // Set the tolerance to agree to within 0.1%
    Do // Main Loop
        for c := 1 to C do
                Rc := 0; // also compute the c class system global response time
                for k:=1 to K do
                        if (Tk="IS") then
                                Rc,k := Dc,k;
                        elseif (Tk="FCFS") or (Tk="LCFSPR") or (Tk="PS") then
                                Rc,k := Dc,k × (1 + hc(Nc,Qc,k));
                                //for example in PE algorithm: hc(Nc,Qc,k)::=[((Nc-1)/Nc)×Qc,k]+F
                                // where the F factor is computed, given c and k as:
                                //      F := 0;
                                //      for j:=1 to C do
                                //              if j<>c then
                                //                      F:=F+Qj,k;
                                //              endif;
                                //      endfor j;
                        endif
                        Rc := Rc + Rc,k;
                        Output "Residence time at for ",c," class at ",k,"-th service center = ",Rc,k;
                endfor k;
                Output "Class ",c," System Response Time = ",Rc;
                Xc := Nc / (Zc + Rc); // Compute class c system throughput
                Output "Class ",c," System Throughput = ",Xc;
                diffc := 0; // initialize difference at a low value
                for k:=1 to K do
                        // Compute the max difference between actual and future values of Qc,k
                        if abs((Qc,k – (Xc × Rc,k))/Qc,k) > diffc then
                                diffc := abs((Qc,k – (Xc × Rc,k))/Qc,k);
                        endif
                        Qc,k := Xc × Rc,k; // Compute Qc,k for the next step
                endfor k;
        endfor c;
    while (Max(diffc, for c:=1 to C) > tol); // if the max diffc > tol then loop else exit
end.
```

Appendice B: File WSDL di descrizione del Web Service

```
<?xml version="1.0" ?>
<definitions name="QNSolver" targetNamespace="urn:QNSolver"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:tns="urn:QNSolver"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns="http://schemas.xmlsoap.org/wsdl/">

<types xmlns="http://schemas.xmlsoap.org/wsdl/" />

<message name="Copyright_Request" />
<message name="Copyright_Response">
      <part name="copyright_tips" type="xsd:string" />
</message>
<message name="ValidateSyntax_Request">
      <part name="model" type="xsd:string" />
      <part name="model_type" type="xsd:string" />
</message>
<message name="ValidateSyntax_Response">
      <part name="validated" type="xsd:string" />
</message>
<message name="ValidateSemantic_Request">
      <part name="model" type="xsd:string" />
      <part name="model_type" type="xsd:string" />
</message>
<message name="ValidateSemantic_Response">
      <part name="validated" type="xsd:string" />
</message>
<message name="Solve_Request">
      <part name="model" type="xsd:string" />
      <part name="model_type" type="xsd:string" />
      <part name="tool" type="xsd:string" />
      <part name="method" type="xsd:string" />
      <part name="params" type="xsd:string" />
</message>
<message name="Solve_Response">
      <part name="solution" type="xsd:string" />
</message>
<message name="Transform_Request">
      <part name="model" type="xsd:string" />
      <part name="model_type" type="xsd:string" />
      <part name="tool" type="xsd:string" />
      <part name="method" type="xsd:string" />
      <part name="params" type="xsd:string" />
</message>
<message name="Transform_Response">
      <part name="translated_model" type="xsd:string" />
</message>
<message name="GetModelDescription_Request">
      <part name="model" type="xsd:string" />
      <part name="model_type" type="xsd:string" />
</message>
<message name="GetModelDescription_Response">
      <part name="text_model_description" type="xsd:string" />
</message>
<message name="GetToolsList_Request" />
<message name="GetToolsList_Response">
      <part name="available_tools_list" type="xsd:string" />
</message>
<portType name="QNSolverPort">
      <operation name="Copyright_">
            <input message="tns:Copyright_Request" />
            <output message="tns:Copyright_Response" />
      </operation>
      <operation name="ValidateSyntax_">
            <input message="tns:ValidateSyntax_Request" />
            <output message="tns:ValidateSyntax_Response" />
      </operation>
      <operation name="ValidateSemantic_">
            <input message="tns:ValidateSemantic_Request" />
            <output message="tns:ValidateSemantic_Response" />
      </operation>
      <operation name="Solve_">
            <input message="tns:Solve_Request" />
            <output message="tns:Solve_Response" />
      </operation>
      <operation name="Transform_">
            <input message="tns:Transform_Request" />
            <output message="tns:Transform_Response" />
```

```
        </operation>
        <operation name="GetModelDescription_">
            <input message="tns:GetModelDescription_Request" />
            <output message="tns:GetModelDescription_Response" />
        </operation>
        <operation name="GetToolsList_">
            <input message="tns:GetToolsList_Request" />
            <output message="tns:GetToolsList_Response" />
        </operation>
    </portType>
    <binding name="QNSolverBinding" type="tns:QNSolverPort">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="Copyright_">
            <soap:operation soapAction="http://schemas.xmlsoap.org/soap/envelope/#QNSolver#Copyright_" />
            <input>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
        <operation name="ValidateSyntax_">
            <soap:operation soapAction="http://schemas.xmlsoap.org/soap/envelope/#QNSolver#ValidateSyntax_" />
            <input>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
        <operation name="ValidateSemantic_">
            <soap:operation soapAction="http://schemas.xmlsoap.org/soap/envelope/#QNSolver#ValidateSemantic_" />
            <input>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
        <operation name="Solve_">
            <soap:operation soapAction="http://schemas.xmlsoap.org/soap/envelope/#QNSolver#Solve_" />
            <input>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
        <operation name="Transform_">
            <soap:operation soapAction="http://schemas.xmlsoap.org/soap/envelope/#QNSolver#Transform_" />
            <input>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
        <operation name="GetModelDescription_">
            <soap:operation soapAction="http://schemas.xmlsoap.org/soap/envelope/#QNSolver#GetModelDescription_" />
            <input>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
        <operation name="GetToolsList_">
            <soap:operation soapAction="http://schemas.xmlsoap.org/soap/envelope/#QNSolver#GetToolsList_" />
            <input>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
```

```
            </input>
            <output>
                <soap:body use="encoded" namespace="http://schemas.xmlsoap.org/soap/envelope/"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
</binding>
<service name="QNSolverService">
        <documentation />
        <port name="QNSolverPort" binding="tns:QNSolverBinding">
            <soap:address location="http://localhost/Queueing_Network_Solver_Service.php" />
        </port>
</service>
</definitions>
```

Appendice C: PMIF 2.0 XML Schema: XSD

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Global -->
<xsd:element name="QueueingNetworkModel" type="QNMType"/>
<!-- Complex Type Definitions -->
<xsd:complexType name="QNMType">
		<xsd:sequence>
			<xsd:element name="Node" type="NodeType" maxOccurs="unbounded"/>
			<xsd:element name="Arc" type="ArcType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="Workload" type="WorkloadType" maxOccurs="unbounded"/>
			<xsd:element name="ServiceRequest" type="ServiceRequestType" maxOccurs="unbounded"/>
		</xsd:sequence>
		<xsd:attribute name="Name" type="xsd:string" use="optional"/>
		<xsd:attribute name="Description" type="xsd:string" use="optional"/>
		<xsd:attribute name="Date-Time" type="xsd:dateTime" use="optional"/>
</xsd:complexType>
<xsd:complexType name="WorkloadType">
		<xsd:choice minOccurs="1" maxOccurs="unbounded">
			<xsd:element name="OpenWorkload" minOccurs="0" maxOccurs="unbounded">
				<xsd:complexType>
					<xsd:sequence>
						<xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
					</xsd:sequence>
					 <xsd:attribute name="WorkloadName" type="xsd:ID" use="required"/>
					<xsd:attribute name="ArrivalRate" type="nonNegativeFloat" use="required"/>
					<xsd:attribute name="TimeUnits" type="TimeUnitsType" use="optional"/>
					<xsd:attribute name="ArrivesAt" type="xsd:IDREF" use="required"/>
					<xsd:attribute name="DepartsAt" type="xsd:IDREF" use="required"/>
				 </xsd:complexType>
			</xsd:element>
			<xsd:element name="ClosedWorkload" minOccurs="0" maxOccurs="unbounded">
				<xsd:complexType>
					<xsd:sequence>
						<xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
					</xsd:sequence>
					<xsd:attribute name="WorkloadName" type="xsd:ID" use="required"/>
					<xsd:attribute name="NumberOfJobs" type="xsd:nonNegativeInteger" use="required"/>
					<xsd:attribute name="ThinkTime" type="nonNegativeFloat" use="required"/>
					<xsd:attribute name="TimeUnits" type="TimeUnitsType" use="optional"/>
					<xsd:attribute name="ThinkDevice" type="xsd:IDREF" use="required"/>
				 </xsd:complexType>
			</xsd:element>
		</xsd:choice>
</xsd:complexType>
<xsd:complexType name="NodeType">
		<xsd:choice minOccurs="0" maxOccurs="unbounded">
			<xsd:element name="Server" minOccurs="0" maxOccurs="unbounded">
				 <xsd:complexType>
					<xsd:attribute name="Name" type="xsd:ID" use="required"/>
					<xsd:attribute name="Quantity" type="xsd:nonNegativeInteger" use="required"/>
					<xsd:attribute name="SchedulingPolicy" type="SchedulingType" use="required"/>
				</xsd:complexType>
			</xsd:element>
			<xsd:element name="WorkUnitServer" minOccurs="0" maxOccurs="unbounded">
				<xsd:complexType>
					<xsd:attribute name="Name" type="xsd:ID" use="required"/>
					<xsd:attribute name="Quantity" type="xsd:nonNegativeInteger" use="required"/>
					<xsd:attribute name="SchedulingPolicy" type="SchedulingType" use="required"/>
					<xsd:attribute name="TimeUnits" type="TimeUnitsType" use="optional"/>
					<xsd:attribute name="ServiceTime" type="nonNegativeFloat" use="required"/>
				</xsd:complexType>
			</xsd:element>
			<xsd:element name="SourceNode" minOccurs="0" maxOccurs="unbounded">
				 <xsd:complexType>
					<xsd:attribute name="Name" type="xsd:ID" use="required"/>
				</xsd:complexType>
			</xsd:element>
			<xsd:element name="SinkNode" minOccurs="0" maxOccurs="unbounded">
				<xsd:complexType>
					<xsd:attribute name="Name" type="xsd:ID" use="required"/>
				</xsd:complexType>
			</xsd:element>
		 </xsd:choice>
</xsd:complexType>
<xsd:complexType name="ServiceRequestType">
		<xsd:choice minOccurs="0" maxOccurs="unbounded">
			 <xsd:element name="TimeServiceRequest" type="TimeServType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="DemandServiceRequest" type="DemandServType" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element name="WorkUnitServiceRequest" type="WorkUnitServType" minOccurs="0" maxOccurs="unbounded"/>
		</xsd:choice>
</xsd:complexType>
<xsd:complexType name="DemandServType">
		<xsd:sequence>
			<xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
		</xsd:sequence>
		<xsd:attribute name="WorkloadName" type="xsd:IDREF" use="required"/>
		<xsd:attribute name="ServerID" type="xsd:IDREF" use="required"/>
		<xsd:attribute name="TimeUnits" type="TimeUnitsType" use="optional"/>
		<xsd:attribute name="ServiceDemand" type="nonNegativeFloat" use="required"/>
		<xsd:attribute name="NumberOfVisits" type="xsd:nonNegativeInteger" use="optional"/>
</xsd:complexType>
<xsd:complexType name="TimeServType">
		<xsd:sequence>
			<xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
		</xsd:sequence>
		<xsd:attribute name="WorkloadName" type="xsd:IDREF" use="required"/>
		<xsd:attribute name="ServerID" type="xsd:IDREF" use="required"/>
```

```
		<xsd:attribute name="TimeUnits" type="TimeUnitsType" use="optional"/>
		<xsd:attribute name="ServiceTime" type="nonNegativeFloat" use="required"/>
		<xsd:attribute name="NumberOfVisits" type="xsd:nonNegativeInteger" use="optional"/>
</xsd:complexType>
<xsd:complexType name="WorkUnitServType">
		<xsd:sequence>
			<xsd:element name="Transit" type="TransitType" maxOccurs="unbounded"/>
		</xsd:sequence>
		<xsd:attribute name="WorkloadName" type="xsd:IDREF" use="required"/>
		<xsd:attribute name="ServerID" type="xsd:IDREF" use="required"/>
		<xsd:attribute name="NumberOfVisits" type="xsd:nonNegativeInteger" use="optional"/>
</xsd:complexType>
<xsd:complexType name="TransitType">
		<xsd:attribute name="To" type="xsd:IDREF" use="required"/>
		<xsd:attribute name="Probability" type="nonNegativeFloat" use="required"/>
</xsd:complexType>
<xsd:complexType name="ArcType">
		<xsd:attribute name="Description" type="xsd:string"/>
		<xsd:attribute name="FromNode" type="xsd:IDREF" use="required"/>
		<xsd:attribute name="ToNode" type="xsd:IDREF" use="required"/>
</xsd:complexType>
<!-- Simple Type Definitions -->
<xsd:simpleType name="nonNegativeFloat">
		<xsd:restriction base="xsd:float">
			<xsd:minInclusive value="0.0"/>
		</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TimeUnitsType">
		<xsd:annotation>
			<xsd:documentation>
				If time units are omitted, all specifications are assumed to be the same relative units.
			</xsd:documentation>
		</xsd:annotation>
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="day"/>
			<xsd:enumeration value="Day"/>
			<xsd:enumeration value="hr"/>
			<xsd:enumeration value="Hr"/>
			<xsd:enumeration value="min"/>
			<xsd:enumeration value="Min"/>
			<xsd:enumeration value="sec"/>
			<xsd:enumeration value="Sec"/>
			<xsd:enumeration value="ms"/>
			<xsd:enumeration value="Ms"/>
			<xsd:enumeration value="ns"/>
			<xsd:enumeration value="Ns"/>
		</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="SchedulingType">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="FCFS"/>
			<xsd:enumeration value="IS"/>
			<xsd:enumeration value="PS"/>
		</xsd:restriction>
</xsd:simpleType>
<xsd:annotation>
		<xsd:documentation>
			Entities (Elements):
				Arc: An Arc connects two Nodes in a QueueingNetworkModel. Traversal of an Arc represents completion of a service request at the FromNode
					and a new request for service at the ToNode.
				ClosedWorkload: A ClosedWorkload is a Workload with a fixed population that circulates among the Servers.
				DemandServiceRequest: A DemandServiceRequest specifies the average service demand (service time multiplied by number of visits)
					provided for each workload that visits the Server.
				Node: A Node represents an entity in the QueueingNetworkModel of the execution environment that either provides service or designates
					model topology.
				Non-ServerNode: A Non-ServerNode represents a Node of the execution environment that designates model topology but does not
					provide processing service.
				OpenWorkload: An OpenWorkload is a workload with a potentially infinite population where transactions or jobs arrive from the outside
					world, receive service, and exit. The population of the OpenWorkload at any one time is variable.
				QueueingNetworkModel: A QueueingNetworkModel represents a network of connected servers that provides processing service for Workloads.
				Server: A Server represents a Node of the execution environment that provides some processing service.
				ServiceRequest: A ServiceRequest specifies either the average TimeService or DemandService provided for each workload that visits
					the Server.
				SinkNode: A SinkNode represents a Node of the execution environment that designates where OpenWorkloads terminate.
				SourceNode: A SourceNode represents a Node of the execution environment that designates where OpenWorkloads originate.
				TimeServiceRequest: A TimeServiceRequest specifies the average service time and number of visits provided for each workload
					that visits the Server.
				Workload: A Workload represents a collection of transactions or jobs that make similar service requests from servers
					in the QueueingNetworkModel.
				WorkUnitServer: A WorkUnitServer represents a Server that has the same ServiceTime for all Workloads.
				WorkUnitServiceRequest: A WorkUnitServiceRequest specifies the number of visits to a WorkUnitServer.
			Attributes:
				ArrivalRate: The average rate at which transactions or jobs arrive from the outside world, receive service, and exit.
				ArrivesAt: The Name of the SoruceNode of an OpenWorkload.
				DepartsAt: The Name of the SinkNode of an OpenWorkload.
				FromNode: The Name of the origin Node of an Arc.
				NumberOfJobs: The fixed population that circulates among the Nodes.
				NumberOfVisits: The average number of visits to a Server in a ServiceRequest.
				Quantity: The number of instances of a given Server. Multiple servers have one queue for service requests.
				SchedulingPolicy: The policy used to select the next ServiceRequest to be served from a queue.
				ServiceDemand: The total demand for a service request. Demand is the product of ServiceTime and NumberOfVisits.
				ServiceTime: The amount of time required for a server to perform one unit of service. A unit of service is the amount provided
					for each visit to the Server.
				ThinkTime: The average interval of time that elapses between the completion of a transaction or job and the submission of the next
					transaction or job.
				TimeUnits: The unit of time specified in a ServiceRequest or Workload intensity. If time units are omitted, all specifications are assumed
					to be the same relative units.
				ToNode: The Name of the destination Node of an Arc.
		</xsd:documentation>
</xsd:annotation>
</xsd:schema>
```

15 Bibliografia e Fonti

[1] "**Web Service, Guida Pratica**", I. Venuti, Edizioni Master;

[2] "**XML Web Services Basics**", R. Wolter, Microsoft Corporation, MSDN Library da Internet;

[3] "**Web services protocolli e strumenti interoperabilità ed evoluzioni future**", L. Barbieri, ObjectWay S.p.A. Ottobre 2001, da Internet;

[4] "**Web Services Description Language (WSDL) 1.1**", W3C Note 15 March 2001, E. Christensen (Microsoft), F. Curbera (IBM Research), G. Meredith (Microsoft), S. Weerawarana (IBM Research), W3C http://www.w3.org/TR/2001/NOTE-wsdl;

[5] "**Costruire applicazioni basate su Web Service**", ASPITALIA.COM http://www.aspitalia.com/articoli/webservice.aspx;

[6] "**Using WSDL in a UDDI Registry, Version 1.07**", UDDI Best Practice, May 21, 2002, F. Curbera (IBM), D. Ehnebuske (IBM), D. Rogers (Microsoft), UDDI.ORG http://www.uddi.org/pubs/wsdlbestpractices.pdf;

[7] "**Modelli e metodi per la valutazione delle prestazioni di sistemi**", S. Balsamo, R. Mirandola, Università degli Studi di Pisa, Servizio Editoriale Universitario di Pisa;

[8] "**Quantitative System Performance, Computer System Analysis Using Queueing Network Models**", E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik, Prentice-Hall Inc. Englewood Clifs New Jersey 07632, ISBN 0-13-746975-6;

[9] "**Performance Evaluation Techniques**", Dr. Ing. A. Willing, Hasso-Plattner-Institut Universität Potsdam, June 29, 2004;

[10] "**Queueing Systems Volume I : Theory**", L. Kleinrock, Hoelpi 1990, Wiley New York 1975, biblioteca della Facoltà di Scienze MM.FF.NN. L'Aquila coll. i/D.4/27/1;

[11] "**Reti di code per l'analisi dei sistemi di calcolo V 1.2**", Nello Scarabotto Dicembre 2003, da internet

[12] "**Sistemi di Servizio e Simulazione**", M. Roma A.A. 05/06, Dipartimento di Informatica e Sistemistica, Facoltà di Ingegneria, Università di Roma "La Sapienza".

[13] "**Queueing Systems Volume II: Computer Applications**", L. Kleinrock, Hoelpi 1990, Wiley New York 1975, biblioteca della Facoltà di Scienze MM.FF.NN. L'Aquila coll. i/D.4/27.2;

[14] "**A Web Service for solving Queueing Network Models using PMIF**", J. Rosselló, M. Lladó, R. Puigjaner, Universitat de les Illes Balears, Dep. de Matemàtiques i Informàtica, Palma de Mallorca, Spain; Connie U. Smith, Performance Engineering Services, Santa Fe, New Mexico USA.

[15] "**Towards uniform interchange formats format for performance validation tools**", A. Di Marco, Vittorio Cortellessa, Dipartimento di Informatica, Università degli Studi di L'Aquila.

[16] "**XPRIT: an XML-based tool to traslate UML diagrams into Execution Graphs and Queuing Networks**", Vittorio Cortellessa, Michele Gentile, Marco Pizzuti, Dipartimento di Informatica, Università degli Studi di L'Aquila.

[17] "**Modelling resources in UML-based simulative environment**", Hany H. Ammar, Vittorio Cortellessa, Alaa Ibrahim, Computer Science and Electrical Engineering Department, West Virginia University, Morgantown.

[18] "**Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation Technical Report**", February 2004, Connie U. Smith, Performance Engineering Service, Santa Fe, New Mexico USA,

Catalina M. Lladó, Universitat Illes Balears, Department de Matemàtiques i Informàtica, Palma de Mallorca Spain.

[19] "**A Basic Performance Model Interchange Format**", May 1997, Connie U. Smith, Performance Engineering Service, Santa Fe, New Mexico USA, Lloyd G. Williams, Software Engineering Research, Ridgeview Lane, Boulder, Colorado USA.

[20] "**A Performance Model Interchange Format**", January 1998, Connie U. Smith, Performance Engineering Service, Santa Fe, New Mexico USA, Lloyd G. Williams, Software Engineering Research, Ridgeview Lane, Boulder, Colorado USA.

[21] "**Computational algorithm for closed queueing networks with exponential servers**", 1973, J. P. Buzen, C.ACM 16, 527-531.

[22] "**Mean value analysis of closed multichain queueing networks**", 1980, M. Raiser, S.S Lavenberg, Journal of the ACM, 27, 313-322.

[23] "**Mean value analysis and convolution method for queue dependent servers in closed queueing networks**", 1981, Perf. Eval., 1 (1). 7-18.

[24] "**Queueing Network-Exact Computational Algorithms: A Unified theory based on Decomposition and Aggregation**", 1989, Conway A. E., N. D. Georganas, MIT Press.

[25] "**Mean value analysis by chain of product from queueing networks**", 1989, Conway A. E., De Souza e Silva E., Lavenberg S.S., IEEE Trans on Comp, 38, 3, 432-442.

[26] "**Calculating joint queue length distributions in product form queueing networks**", 1989, De Souza e Silva E., Lavenberg S. S., J. ACM, 36, 1, 194-207.

[27] "**A perspective on queueing models of computer performance**", 1989, Levenberg S. S., Perf. Eval. 10, 53-76.

[28] "**A tree convolution algorithm for the solution of queueing networks**", 1983, Lam S. S., Y. L. Lien, C. ACM 26,3,203-215.

[29] "**The tree MVA algorithm**" 1985, Tucci S., C. H. Sauer, Perf. Eval. 5,3.

[30] "**SISTEMI A CODA, Introduzione alla teoria delle code**", 1992, Leonard Kleinrock, Biblioteca Scientifica Hoelpi. Biblioteca della Facoltà di Ingegneria, Università degli Studi di L'Aquila, coll. ing. 519.82.

[31] "**PHP 5, Guida Completa**", 2005, Andi Gutmans, Stig S. Bakken, Derick Rethans, APOGEO.

[32] "**GRAPH DRAWING, Algorithms for the visualization of graphs**" 1999, Giuseppe Di Battista, Peter Eades, Roberto Tamassia, Ioannis G. Tollis, PRENTICE HALL, disponibile c/o Biblioteca della Facoltà di Scienze coll. i. G. 2. 11.

[33] "**SIMULATION MODELING & ANALYSIS, Second Edition**" 1991, Averill M. Law, W. David Kelton, McGRAW-HILL INTERNATIONAL EDITIONS, Industrial Engineering Series, disponibile c/o Biblioteca della Facoltà di Scienze coll. i.I.6.6.

[34] "**The Practical Performance Analyst**" 1998, Libro associato al Tool/libreria in C "**PDQ Analyzer**", Neil J. Gunther, Authors Choice Press an Imprint of iUniverse.com Inc. Originally published by McGraw Hill.

[35] "**Analyzing Computer System Performance with Perl::PDQ**" 2004, Libro associato al Tool/libreria in PERL "**PDQ Analyzer**", Neil J. Gunther, SPRINGER.

[36] "**Mean Value Analysis for Closed, Separable, Multi Class Queueing Networks with Single Server & Delay Queues**" pubblicazione associata al Tool/Libreria "**MVA Queueing Formalism Parser**" , Chandrakanth

Chereddi, Department of Electrical & Computer Engineering, and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, cchered2@uiuc.edu.

[37] "**QNAP2: A portable environment for queueing system modelling**" 1984, Michel Véran (BULL Sems France), Dominique Potier (INRIA France).

[38] "**SHARPE: Symbolic Hierarchical Automated Reliability and Performance Evaluator, Introduction and Guide for Users**" 1992, Robina A. Sahner, Kishor S. Trivedi (Duke University).

[39] "**Performance And Reliability Analysis of Computer System, An Example-Based Approach Using the SHARPE Software Package**", Robin SAHNER (Urbana, IL), Kishor S. TRIVEDI (Duke University, Durham, N.C.), Antonio PULIAFITO (University of Catania, Catania, Italy), Kluwer Academic Publishers Boston/London/Dordrecht.

[40] "**SHARPE Interface, User's Manual Version 1.01**" 1999, Kishor S. TRIVEDI, Center of Advanced Computing and Communication (CACC), Department of Electrical and Computer Engineering, Duke University, Durham, NC, 27708, USA, kst@ee.duke.edu.

[41] "**Approximate MVA Algorithms For Solving Queueing Network Models**" 1997, Hai Wang, Master of Science, Graduate Department of Computer Science, University of Toronto.

[42] "**Approximate analisys of multiclass closed networks of queues**" 1979, P. J. Schweitzer, Proceedings of International Conference on Stochastic Control and Optimization, 25-29, Amsterdam, Netherlands.

[43] "**Linearizer: A heuristic algorithm for queueing network models of computing systems**" Febbraio 1982, K. M. Chandy and D. Neuse, Communications of the ACM, 25(2):126-134.

[44] "**Queueing networks with multiple closed chains: Theory and computational algorithms**", Maggio 1975, M. Reiser and H. Kobayashi, IBM Journal of Research and Development, 19(3):283-294.

[45] "**Stationary state probabilities of arrival instants for closed queueing networks with multiple types of customers**" Dicembre 1980, S. S. Lavenberg and M. Reiser, Journal of Applied Probability, 17(4):1048-1061.

[46] "**The distribution of queueing network states at input and output instants**" Aprile 1981, K. C. Sevcik and I. Mitrani, Journal of the ACM, 28(2):358-371.

[47] "**A proof of the queueing formula L = λW**" Maggio/Giugno 1961, J. D. C. Little, Operations Research, 9(3):383-387.

[48] "**Performance Model Interchange Format: Semantic Validation Technical Report**", Maggio 2006, D. García, C. M. Lladó, C. U. Smith, R. Puigjaner.

[49] "**PMVA – Perdue Mean Value Analysis Program – User's Guide**", Aprile 1981, Jeff Brumfield, Department of Computer Sciences, Perdue University, West Lafayette, Indiana 47907, CSD-TR-383.

[50] "**QNAP2 version 9.3 User's Guide**", July 1996, Simulog FR.

[51] **Standard Performance Evaluation Corporation**: http://www.spec.org

[52] **TPC**: http://www.tpc.org

[53] **EEMBC**: http://www.eembc.org

[54] **Web Polygraph**: http://www.web-polygraph.org

[55] "**Capacity Planning and Performance Modeling: from Mainframes to client-server systems**", 1994, D. Menasce, V. Almeida, L. Dowdy, Prentice Hall

[56] "**Getting Started with CSIM17 (C Version)**", Mesquite Software, Inc. 3925 West Braker Lane, Austin, TX, info@mesquite.com

[57] "**GUIDE: A Graphical Interface for Specification of Extended Queueing Network Models**", 1986, J. B. Sinclair e S. Madala, Department of Electrical and Computer Engineering, Rice University, Huston, Texas

[58] "**The Queueing Network Analysis Tool (QNAT)**", Hema Tahilramani Kaur of Electrical Computer and Systems Engineering Dept, Rensselaer Polytechnic Institute, Troy NY; D. Manjunath of Dept of Electrical Engg., Indian Institute of Technology, Powai Mumbau, INDIA; Sanjay K. Bose, Dept of Electrical Engg., Indian Institute of Technology, Kampur, INDIA

[59] "**The Research Queueing Package Modeling Environment (RESQME)**", Proceedings of the 1993 Winter Simulation Conference, K. C. Chang, R. F. Gordon, P. G. Loewner, E. A. MacNair, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA

[60] "**Analysis of manufacturing systems by the Research Queueing Package**", IBM J. Res. Develop. Vol 29 No. 4 July 1985, by We-Min Chow, Edward A. MacNair, Charles H. Sauer

[61] "**A Language for Extended Queueing Network Models – RESQ2**", IBM J. Res. Develop., Vol. 24, No. 6, November 1980, by Charles H. Sauer, Edward A. MacNair, Silvio Salza

[62] "**SAM: A Tool for Software Architecture Modeling & Performance Analysis**", Proceeding of the Second International Conference on the Quantitative Evaluation of Systems (QUEST'05), 2005 IEEE Computer Society, by Rajeshwari G. and Santonu Sarkar Software Engineering and Technology Labs, Infosys Technologies Ltd

[63] "**Discrete-event simulation system Delsi 1.1 – Programmer's Guide**", © 1996-2002, Herman Holushko

[64] "**MOSEL: MOdeling Specification and Evaluation Language**", 2002 ESM02, J. Barner, K. Begain, G. Bolch and H. Herold. MOSEL Project Web Site: http://www4.informatik.uni-erlagen.de/Projects/MOSEL/

[65] "**The Performance Evaluation and Prediction System for Queueing NetworkS: PEPSY-QNS**", June 1992, G. Bolch and M. Kirschnick, Computer Science Department, Operating Systems – IMMD IV, Friederich Alexander University FAU, Erlangen Nürnberg, Germany

[66] "**MQNA – Markovian Queueing Networks Analyser**", Proceedings of the 11TH IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS'03) © 2003 IEEE, L. Brenner, P. Fernandes, A. Sales, PUCRS, Av. Ipiranga, Porto Alegre, BRAZIL. http://www.inf.pucrs.br/mqna/.

[67] "**MQNA – Markovian Queueing Networks Analyser**", L. Brenner, P. Fernandes, A. Sales, PUCRS, av. Ipiranga, Porto Alegre, BRAZIL. http://www.inf.pucrs.br/mqna/.

[68] "**Möbius: An Extensible Tool For Performance and Dependability Modeling**", 1999, David Daly, Daniel D. Deavours, Jay M. Doyle, Aaron J. Stillman, Patrick G. Webster, Department of Electrical and Computer Engineering, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana IL, USA

[69] "**The Möbius Modeling Tool**", 2001, G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders and P. Webster, Department of Electrical and Computer Engineering, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana IL, USA

[70] "**The Möbius Modeling Environment: Recent Developments**", Proceedings of the First International Conference on Quantitative Evaluation of Systems (QUEST'04), T. Courtney, D. Daly, S. Derisavi, S. Gaonkar, M. Griffith, V. Lam and W. H. Sanders, Department of Electrical and Computer Engineering, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana IL, USA

[71] "**The Möbius Tool – Overview of Features**": http://www.mobius.uiuc.edu/

[72] "**Object-Oriented Discrete-Event Simulation in C++, C++SIM User's Guide – Public Release 1.6 – Draft Version 1.0**", developed as part of the Arjuna project in 1985, Department of Computing Science, Computing Laboratory, The University Newcastle upon Tyne, NE1 7Ru, UK

[73] "**DESP-C++: A Discrete-Event Simulation Package for C++**", To appear in Software, Practice & Experience, Vol. 30, No. 1-24, 2000, Jerome Darmont, LIMOS, Blaise Pascal University (Clermont-Ferrand II), FRANCE. http://libd2.univ-bpclermont.fr/~darmont/download/desp-c++.tar.gz

[74] "**Dependable LQNS: A Performability Modeling Tool for Layered Systems**", Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN'03), IEEE Computer Society, O. Das, C. M. Woodside, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, CANADA

[75] "**ORSTAT: MCQueue**", 2000-2002, Hnek Tijms and Peter Schram, Dept. of Econometrics and Operational Research, Vrije University, Amsterdam.

[76] "**MINA – A Tool for MSC-Based Performance Analysis and Simulation of Distribuited Systems**", Dissertation zur Erlangung des Grades, Doktor der Naturwissenschaften, von Hesham Kamal Arafat Mohamed, Institut fur Informatik und Wirtschaftsinformatik der Universitat Duisburg-Essen, Gatuachter Prof. Br. Bruno Müller-Clostermann and Prof. Dr. Micheal Göedicke

[77] "**QSOLVE: An Object-Oriented Software Tool for Teaching Queueing Theory**", Fernando C. Castaño Mariño e Paulo R. L. Gondim, System Engineering Department – Instituto Militar de Engenharia – IME (Rio de Janeiro – RJ)

[78] "**Fundamentals of Queueing Theory, Third Edition**" di D. Gross e C. Harris, questo libro contiene il software "**QTS-PLUS per Excel**", il libro è distribuito da John Wiley & Sons Inc. http://www.geocities.com/qtsplus, ftp://ftp.wiley.com/public/sci_tech_med/queueing_theory

[79] "**A User's Guide to RAQS: Rapid Analysis of Queueing Systems**", S. Sivaramakrishnan and G. Shirhatti, Dr. Manjunath Kamath, Developed at the Center for Computer Integrated Manufacturing School of Industrial Engineering and Management, Oklahoma State University, Stillwater, OK 74078

[80] "**OMNet++ Manual**", by Andreás Varga, Omnest Global, Inc. http://www.omnest.com.

[81] "**WinPEPSY**", Department of Computer Science, Engineering Sciences Faculty, Friederich Alexander University FAU, Erlangen Nürnberg, Germany

[82] **XML**: http://www.w3.org/2000/xp/

[83] **SOAP 1.1**: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/