POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione
Corso di Laurea Specialistica in Ingegneria Informatica

APPROCCIO BASATO SU COMPLEX EVENT PROCESSING PER L'ADATTIVITÀ NEI SISTEMI INFORMATIVI

Relatore: Ing. Pierluigi Plebani

Tesi di Laurea di:
Alessandro Miracca
Matricola n° 750270

Anno Accademico 2010-2011

A Katty,

Ringraziamenti

Questa tesi segna, con molta probabilità, l'epilogo della mia carriera universitaria e, più in generale, quella di studente. Negli ultimi anni ho sempre pensato a questo momento come alla fine di una parte della mia vita e all'inizio di una nuova. Una profonda spaccatura, dunque, tra un mondo in cui sono considerato un "vecchio" e uno in cui ricoprirò di nuovo il ruolo della matricola, di colui che "non sa niente".

I sentimenti dominanti davanti ad una prospettiva di questo tipo sono un misto di eccitazione e paura. Da un certo punto di vista, sono contento di potermi finalmente mettere alla prova nel mondo del lavoro e vedere cosa sono in grado di ottenere in un periodo così difficile, dal punto di vista economico, per il Paese tramite le conoscenze ottenute e maturate in questi anni di studi. Mi sono sempre ritenuto una persona determinata e le sfide non mi hanno mai spaventato; i valori che le persone a me vicine mi hanno trasmesso mi impongono di tentare di fare sempre il massimo in qualunque situazione, dalla più banale alla più complessa. In tutto ciò, però, rimane un pizzico di paura, come quella che si prova a camminare al buio in un luogo sconosciuto. Essa, tuttavia, non è per forza un qualcosa di negativo, anzi può essere interpretata come un ulteriore stimolo.

Ci sono molte persone che vorrei ringraziare, ma la prima è sicuramente l'Ing. Plebani. Ho avuto il piacere di fare la sua conoscenza già durante la laurea triennale, a Cremona, per poi ritrovarlo nella mia esperienza a Milano. Sin dalla prima volta che ci siamo visti, mi è sembrata una persona, per certi tratti, molto diversa da tutte le altre figure professionali incontrate all'interno del Politecnico di Milano: ha sempre mostrato un contatto sincero con gli studenti e una disponibilità rara. Nell'arco dei nove mesi in cui questa tesi è stata concepita e realizzata, ho sempre ricevuto delle risposte rapide e concrete ad ogni mio dubbio e un aiuto importante nella fase di progettazione del software e stesura della tesi, indipendentemente dall'ora e dal giorno.

Rimanendo nell'ambito universitario, devo confessare che questi ultimi due anni trascorsi a Milano sono stati molto diversi dai precedenti tre: ho trovato un ambiente molto diverso da quello a cui ero abituato e persone che vivevano l'esperienza della laurea specialistica come una vera e propria competizione senza regole, non facendosi scrupoli ad ostacolare l'operato degli altri. Tutto questo mi ha impedito, anche a causa del mio carattere, di legarmi a qualcuno come mi era successo nella precedente esperienza nella Sede di Cremona. Ciò nonostante, ci sono alcune persone che vorrei ringraziare per il loro aiuto, a volte anche fondamentale, in alcuni progetti. Un grazie, quindi, a Iacopo, Maddalena, Marco, Matteo, Nicholas, Nicola e Zubair.

Un altro grazie sincero va alla mia famiglia, che, anche in questa seconda avventura, non mi ha mai fatto mancare nulla. Se sono arrivato fin qui è soprattutto grazie a loro che mi hanno permesso di studiare, facendo a volte anche dei sacrifici, ma senza mai farmeli pesare. Non sono una persona che esterna facilmente le proprie emozioni, specialmente quando si tratta della sfera privata; loro lo sanno e capiranno che, anche se non mi dilungo nel ringraziarli uno ad uno, hanno tutti avuto un ruolo fondamentale nel mio processo di crescita, dai miei nonni materni, che come piace dire a loro "mi hanno cresciuto", a mia zia Barbara, dalla quale ho avuto sempre più di quanto ci si potrebbe aspettare, i nonni paterni, che per ragioni geografiche non vedo tanto quanto gli altri, ma mi hanno ugualmente trasmesso valori importanti, i miei cugini e, ovviamente, i miei genitori, che meriterebbero una pagina intera ciascuno. Grazie anche a mia sorella Giulia per i suoi consigli, di cui ormai sono diventato dipendente; ora che ho terminato gli studi ho finalmente il tempo di accettare la sua richiesta di amicizia su Facebook (lei sa cosa significa).

In ultimo ci tengo a ringraziare alcuni amici, nel senso più vero del termine. Un ringraziamento speciale va a Nicola: ci conosciamo dai tempi dell'asilo, abbiamo affrontato insieme gli anni del liceo e ancora oggi siamo profondamente legati. Oltre a lui, vorrei menzionare anche tutti i ragazzi con cui mi diverto a "grindare" in giro per l'Italia: grazie Davide M., Davide R., Emanuele, Federico, Luca C., Luca Z., Matteo B., Matteo T., Niccolò, Paolo, Samuele, Silvio e Stefano per tutte le volte che avete dovuto sopportare me e le mie patte.

Alessandro

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'efficienza energetica nei data centre e, più in generale il concetto di Green IT, sono argomenti di grande attualità e rappresentano alcune delle più affascinanti e cruciali sfide a livello ambientale dei prossimi anni a causa del sempre più crescente trend di digitalizzazione dei processi di business. Online banking, e-Government, e-Health e intrattenimento digitale sono solo alcuni dei nomi di questi processi. A titolo di esempio, si noti che l'emissione di CO2 dei data centre a livello mondiale è pari a circa la metà di quella prodotta da tutte le compagnie aree del mondo messe insieme, ed è maggiore di quella di alcuni paesi come Argentina e Paesi Bassi [GAM10].

Come riportato dalla European Climate Foundation, nei prossimi quarant'anni si prevede che il consumo energetico aumenterà del 40% e i relativi costi dell'energia cresceranno fino a circa 90 miliardi di dollari [GAM10].

Se si guarda con attenzione al panorama IT, e ai data centre in particolare, solo il 60% dell'energia consumata da un data centre è utilizzata da server, processori e software. Infatti, gli sprechi sono frequenti e buona parte dell'energia consumata non dipende dal servizio o, in generale, dall'output offerto. Un'altra buona percentuale viene utilizzata da sistemi di raffreddamento, gruppi di continuità e sistemi per la distribuzione di energia, tutti componenti che risultano necessari all'interno di una logica di insieme del sistema, anche se non sono direttamente collegati all'output.

I responsabili IT, di conseguenza, se supportati da strumenti efficaci ed efficienti per il calcolo dei valori critici del sistema, possono giocare un ruolo strategico importante nel comprendere il peso specifico della spesa in energia, all'interno dei costi operazionali delle infrastrutture IT, e fare scelte migliori basate su compromessi tra le politiche ambientali dell'impresa e i costi.

In questo ambito si colloca GAMES (Green Active Management of Energy in IT Service centres): un progetto europeo facente parte del 7th Framework Programme of Research and Technological Development che ha come obiettivo lo sviluppo di un insieme di metodologie, metriche, strumenti e servizi software innovativi per aumentare l'efficienza energetica, sia a livello di design che a livello di gestione, nei data centre della nuova generazione.

Come illustrato nei prossimi capitoli, questo progetto punta a definire una nuova metodologia di sviluppo degli applicativi che verranno gestiti all'interno di data centre e di un sistema di monitoraggio in tempo reale attraverso i classici indicatori di prestazione, noti come KPI

(Key Performance Indicator), e i più innovativi indicatori di greenness, chiamati GPI (Green Performance Indicator), attenti alle caratteristiche del programma, o dell'hardware sul quale esso agisce, che potrebbero avere un impatto negativo sull'ambiente.

Ogni applicazione che fornisce informazioni relative agli indicatori KPI e GPI che essa influenza, oltre alla serie di requisiti, funzionali e non, necessari alla sua esecuzione, viene etichettata come applicazione GAMES-enabled.

Lo scopo di questo lavoro di tesi è fornire uno strumento a supporto della progettazione e controllo di applicazioni a servizi adattivi, e quindi integrato all'interno della metodologia GAMES, con lo scopo di diminuire il loro consumo energetico. A tal proposito sfruttati alcuni elementi propri del progetto GAMES, quali la capacità di monitorare il sistema attraverso indicatori GPI e KPI e un modello Goal-based sul quale ci si baserà per risolvere i comportamenti scorretti che si presenteranno di volta in volta.

L'innovatività nella soluzione proposta riguarda la capacità di monitoring dell'impianto, che è stata arricchita con una funzionalità di predizione, così da poter provare ad anticipare le eventuali violazioni delle soglie massime e minime da parte dei GPI e KPI. Utilizzando come elementi di partenza la teoria delle serie temporali e alcune basi di statistica, è possibile ottenere dei modelli di predizione per gli indicatori di interesse, semplicemente osservando il loro andamento passato. La quantità e la qualità dei dati a disposizione saranno entrambe caratteristiche determinanti per definire l'attendibilità e la bontà della predizione. Non è sempre vero, in realtà, che ad un maggior numero di dati corrisponde un modello di predizione più accurato; bisogna anche guardare alla conformazione delle serie dei dati per capire se essa può essere modellata facilmente o meno.

L'analisi e la predizione dei dati di monitoraggio producono una serie di eventi che danno luogo alla necessità di adattare il sistema.

Tutti questi eventi vengono poi esaminati da un engine CEP (Complex Event Processing), uno strumento software, nato negli anni Sessanta, molto utile per trattare con essi. L'event processing si basa sul meccanismo di *continuous query*, un modo di operare sostanzialmente molto diverso dal più conosciuto metodo di interrogazione delle base di dati. Esso, di fatti, non esegue query su un database nel quale sono state salvate, sotto forma di tuple, le informazioni relative ad ogni evento occorso nel sistema, bensì esamina "on-the-fly" il flusso di eventi ricercando particolari condizioni codificate in un insieme di query predefinito. Tale approccio si rende necessario per via dell'enorme numero di eventi che è possibile riscontrare in un sistema di questo genere; il continuo accesso ad un database per portare a termine le varie operazioni di lettura e scrittura non farebbe altro che rallentare l'esecuzione del programma, e non è da escludere che potrebbe anche causare problemi di univocità e di accesso ai dati.

Ogniqualvolta l'engine CEP incontra un evento che corrisponde ad un comportamento scorretto, sia che esso si stia verificando concretamente oppure che sia frutto di una predizione, avvia una procedura, detta di adattamento, la quale ha il compito di suggerire all'utente incaricato di sorvegliare il sistema la migliore azione di riparazione possibile tra quelle implementate. Un'azione di riparazione è una modifica strutturale o logica della macchina fisica che ospita l'applicazione da cui è nato il problema che si sta tentando di risolvere; alcuni esempi di azioni

di riparazione sono rappresentati dalla possibilità di migrare una Virtual Machine (VM) da un server ad un altro, oppure dall'ibernazione di un server.

Il presente lavoro di tesi ha, quindi, riguardato l'analisi, la progettazione e la realizzazione di un sistema di predizione di eventi e di analisi degli stessi. Tutto ciò con l'obiettivo di monitorare ed adattare applicazioni eseguite in un data centre con il fine di loro consumo energetico.

L'elaborato di tesi è strutturato come segue. Nel Capitolo 2 verrà presentata la definizione formale di GPI (Green Performance Indicator) e una rapida introduzione al progetto GAMES, con particolare attenzione agli aspetti più utili a questa trattazione.

Nel Capitolo 3 si affronterà il tema del Complex Event Processing, fornendo per prima cosa una definizione di "evento" e di "Complex Event Processing", per poi passare alle sue caratteristiche principali. Si distingueranno i vari attori di una rete di event processing, anche detti "building blocks", e le molteplici specializzazioni che un *event processing agent* può assumere. Successivamente, si farà riferimento alle più note implementazioni presenti sul mercato, con un particolare occhio di riguardo al miglior prodotto in circolazione, Progress Apama, e a quello che verrà utilizzato per lo sviluppo del progetto in questione, Esper, scelto per la sua caratteristica open-source e le sue comunque notevoli doti di flessibilità ed efficacia.

Il Capitolo 4 introduce il problema dell'anticipazione di una violazione. Qui verranno forniti gli strumenti necessari per comprendere i concetti di serie temporale e di predizione nelle serie temporali.

Il dettaglio di ciascuna delle fasi che compongono il software associato a questo elaborato di tesi è contenuta nel Capitolo 5. Oltre al codice implementato, si farà riferimento alle tecnologie fornite da parti terze, utilizzate per collegare tutte le strutture necessarie, come database, strumenti di calcolo distribuito e parser JSON.

I dettagli relativi allo sviluppo dell'interfaccia client-server, alla messa in funzione della rete locale per lo scambio di messaggi e ai risultati ottenuti dai test effettuati sono contenuti nel Capitolo 6.

Infine, nel Capitolo 7, sono contenute alcune considerazioni finali sul lavoro e su come sarebbe ulteriormente possibile migliorarlo.

Capitolo 2

GPI & KPI

Dopo aver fornito una esaustiva definizione di Green Performance Indicator (GPI), si esaminerà l'utilizzo di tale concetto all'interno del progetto europeo GAMES. In particolare, si vedrà come un indicatore GPI o KPI viene definito, come viene calcolato e il suo legame con un altro elemento di notevole importanza per questa trattazione,

2.1 Che cos'è un Green Performance Indicator?

Un GPI (Green Performance Indicator) è un estensione del generico concetto di KPI (Key Performance Indicator)[1] per la misurazione di caratteristiche di processo, a diversi livelli, nei sistemi informativi ed è utilizzato come criterio di base per la definizione di SLA (Service Level Agreements)[2] e contratti con utenti in cui è necessario soddisfare una certa quale efficienza energetica garantita [JKC+10]. Spesso i GPI vengono definiti come l'elemento fondamentale per cui si fa raccolta e analisi dei dati: l'ideale consumo energetico a loro associato, infatti, è necessario per la comparazione con i dati reali provenienti dal sistema e per ottenere il grado di efficienza energetica dell'intero IT Service Centre.

Analogamente ai KPI, che sono un modo di misurare quanto bene l'organizzazione aziendale, uno specifico individuo o un processo all'interno di tale organizzazione, gestiscono un'attività di tipo strategico, operazionale o direzionale che risulta essere critica per il presente e futuro successo dell'impresa, i GPI sono uno strumento per misurare il livello di attenzione all'ambiente del sistema IT.

In definitiva, i GPI ambiscono, quindi, a misurare il consumo di risorse da parte di una applicazione. L'ipotesi di base per fare ciò è che essa venga monitorata su una macchina fisica per un periodo di osservazione minimo dato dall'intervallo di tempo $T = [t_i, t_k)$, dove con t_i si intende il valore di inizio e con t_k il lasso di tempo minimo che deve trascorrere per ottenere dei dati significativi.

[1] Un indicatore chiave di prestazione (in inglese Key Performance Indicator o KPI) è un indice che monitora l'andamento di un processo aziendale. Solitamente i KPI vengono determinati da un analista, che esegue un'analisi top-down dei processi, a partire quindi dall'esigenza dei vertici oppure dall'analisi del problema.

[2] I Service Level Agreement, in sigla SLA, sono strumenti contrattuali attraverso i quali si definiscono le metriche di servizio che devono essere rispettate da un fornitore di servizi.

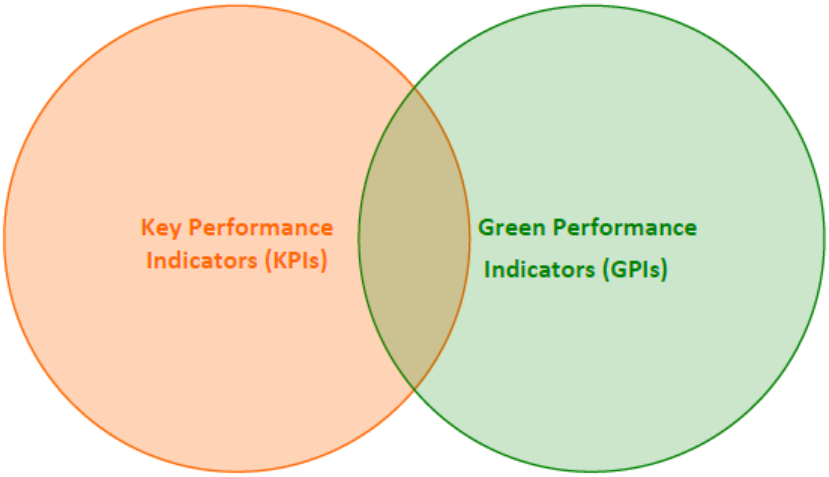

Figura 2.1: Relazione tra GPI e KPI

In un IT Service Centre, la maggior parte degli indicatori di prestazione sono raggruppati in KPI e GPI. Per esempio, il costo di un'applicazione, incluso quello per la progettazione, lo sviluppo del codice e il deploy, è un Key Performance Indicator, dato che, come è chiaro, non ha nessuna relazione diretta con il livello di rispetto dell'ambiente da parte della stessa. Al contrario, l'efficienza infrastrutturale del data centre, solitamente indicata con l'acronimo inglese DCiE, è un Green Performance Indicator, poiché direttamente riferita al consumo energetico e alla greenness dell'applicazione e dell'IT Service Centre.

Ciò nonostante, alcuni KPI possono avere conseguenze su uno o più GPI, così come mostrato dalla Figura 2.1. Generalmente, in un'organizzazione sufficientemente complessa, il livello di utilizzo di risorse è visto come un indicatore di costo, quindi come un KPI, ma nel contesto GAMES, tale utilizzo di risorse, quali ad esempio l'utilizzo della CPU o l'utilizzo di memoria, viene visto come un GPI. Per questo motivo, la Figura 2.1 riporta una zona di intersezione tra l'insieme dei KPI e quello dei GPI [JKC+10].

2.2 Il concetto di GPI all'interno del progetto GAMES

2.2.1 Introduzione al progetto GAMES

GAMES (Green Active Management of Energy in IT Service centres) è un progetto europeo facente parte del 7th Framework Programme e ha come obiettivo lo sviluppo di un insieme di metodologie, metriche, strumenti e servizi software innovativi per aumentare l'efficienza energetica, sia a livello di design che a livello di gestione, all'interno dei data centre della nuova generazione, a cui spesso ci si riferisce con il nome di IT Service Centre.

Esso si impone di migliorare in maniera consistente l'efficienza energetica di questi data centre e di ridurre le loro emissioni, garantendo, comunque, delle performance generali soddisfacenti, sia

per i produttori che per i fruitori dei servizi. Tutto ciò può essere fatto combinando e integrando il consumo energetico misurato ai vari livelli applicativi e il carico di lavoro dei componenti IT, come processori e memorie, per ottenere un unico framework, funzionante a real-time, in grado di monitorare il sistema a diversi livelli di granularità.

In GAMES vengono calcolati GPI relativi sia al data centre in generale, un qualcosa che esiste già da tempo in letteratura, sia ad una specifica applicazione, poiché l'obiettivo primario rimane quello di realizzare applicazioni di tipo green, ovvero rispettose dell'ambiente; per questo motivo, è importante avere dei parametri che misurino sempre la greenness[3] dell'applicazione.

Ogni applicazione che è in grado di fornire informazioni relative agli indicatori KPI e GPI che essa influenza, alle azioni necessarie da intraprendere nel caso qualcuno di questi risulti violato e alle relazioni tra il codice che la compone e l'infrastruttura IT sulla quale è in funzione, viene indicata come un'applicazione GAMES-enabled.

All'interno del progetto GAMES, è possibile classificare GPI e KPI nelle seguenti quattro categorie:

- IT Resource Usage GPIs (CPU Usage, Storage Usage, ecc)
- Application Lifecycle KPIs (Response Time, Throughput, Availability, ecc)
- Energy Impact GPIs (IOPS/Watt, DCiE, PUE, ecc)
- Organizational Factors GPIs (Carbon Credit, RoGI[4], ecc)

La definizione di ciascuno di questi indicatori fa parte dell'innovativa metodologia di progettazione di GAMES che verrà presentata nella Sezione seguente.

2.2.2 Co-design methodology

Uno dei principali obiettivi del progetto GAMES è la riduzione del consumo di energia nei data centre, focalizzandosi prima di tutto sulle applicazioni funzionanti all'interno dello specifico data centre. Il progetto ambisce, dunque, a fornire una modo per misurare e migliorare l'efficienza energetica a livello applicativo. Così facendo, i responsabili dei vari Green IT Service Centre potranno essere in grado di conoscere quali sono le applicazioni che richiedono più energia e, di conseguenza, mettere sotto esame tali applicazioni per migliorarle.

Il raggiungimento di questo scopo richiede di rivedere non solo la fase di esecuzione di queste applicazioni, ma anche e soprattutto quella di sviluppo. Diversi elementi possono essere determinanti ai fini del calcolo del consumo energetico di un'applicazione: basti pensare alle risorse utilizzate dall'applicazione, alla struttura del processo e al numero di istanze in funzione. Non meno, l'hardware su cui l'applicazione viene fatta funzionare influenza non poco il calcolo del consumo di energia.

Per questi motivi, è stata introdotta una metodologia energy-aware per lo sviluppo di applicazioni che guarda sia ai requisiti dell'applicazione in termini di risorse impiegate e struttura

[3] Greenness è un termine di origine inglese, alquanto intraducibile. Indica il livello di attenzione all'ambiente del prodotto a cui fa riferimento.

[4] Il periodo di tempo necessario per recuperare l'investimento in risorse umane, monetarie e altro sborsato per implementare la soluzione di Green IT in questione.

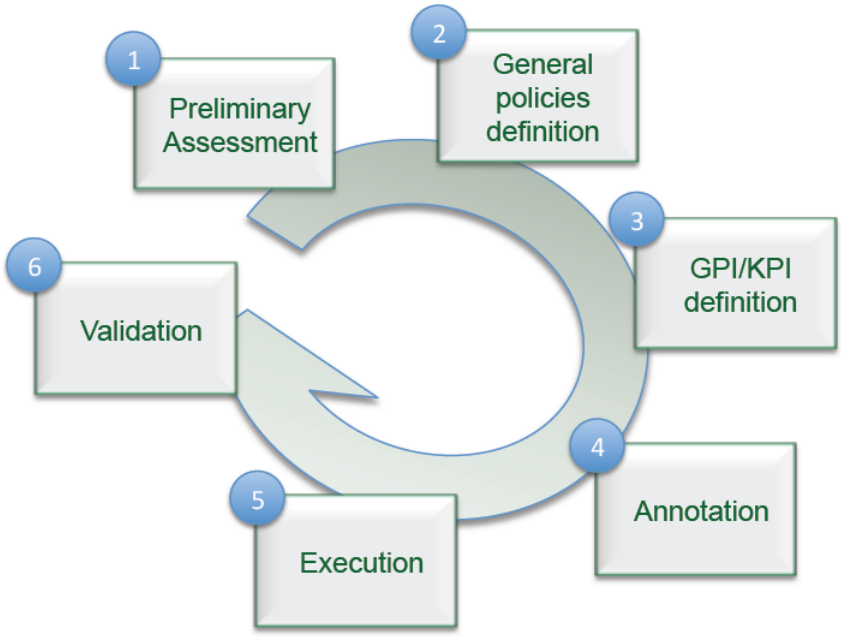

Figura 2.2: Co-design methodology

della stessa, sia all'infrastruttura necessaria per farla operare correttamente. Essa prende il nome di "co-design methodology".

Tale metodologia si compone di sei fasi, come è possibile vedere in Figura 2.2, ognuna delle quali svolge dei compiti ben precisi:

1. *Preliminary assessment*: si svolge un'analisi della situazione corrente nel data centre relativamente alle apparecchiature, l'hardware e il software installato e il consumo energetico. Dopo questa fase, il responsabile del data centre è in grado di identificare quale sia il livello di maturità del data centre guardando al Data Centre Maturity Model[5].

2. *General polices definition*: basandosi sui dati raccolti nella fase precedente, l'amministratore del data centre definisce gli obiettivi da raggiungere in modo da aumentare il livello di maturità. Questa fase è solitamente eseguita a livello strategico, dato che gli indicatori adottati per identificare gli obiettivi desiderati non sono ancora molto precisi.

3. *GPI/KPI definition*: a partire dagli obiettivi prima definiti, in questa fase vengono messi in evidenza i vincoli da applicare sugli indicatori misurabili.

4. *Annotation*: essa si concentra sull'applicazione e arricchisce la classica fase di design richiedendo che ad ogni applicazione vengano aggiunte annotazioni utili per valutare il proprio consumo energetico.

[5]Il Data Centre Maturity Model (DCMM), proposto da The Green Grid Consortium, un consorzio di aziende accomunate dal loro interesse nella Green IT, introduce una scala di sei livelli di maturità per i data centre relativamente alla loro efficienza energetica e sostenibilità: il livello 0 è il più basso e indica la totale mancanza di progresso o attenzione verso l'ambiente, mentre il livello 5, il più alto, viene assegnato ai data centre più innovativi che ricercano continuamente soluzioni più performanti.

5. *Execution*: l'applicazione viene messa in funzione e allo stesso monitorata all'interno del data centre. L'infrastruttura di monitoraggio sarà in grado di raccogliere informazioni riguardanti il consumo di energia e l'utilizzo di risorse.

6. *Validation*: i dati monitorati vengono usati per calcolare i valori correnti dei GPI e KPI rilevanti ai fini dell'applicazione. I valori ottenuti vengono poi confrontati con gli obiettivi definiti nella fase preliminare in modo da poter dire se essi sono stati rispettati. Se non è così, si richiedono azioni di riparazione o adattamento che possono avvenire sia a design-time, che a run-time [CFP+11].

2.2.3 Architettura generale

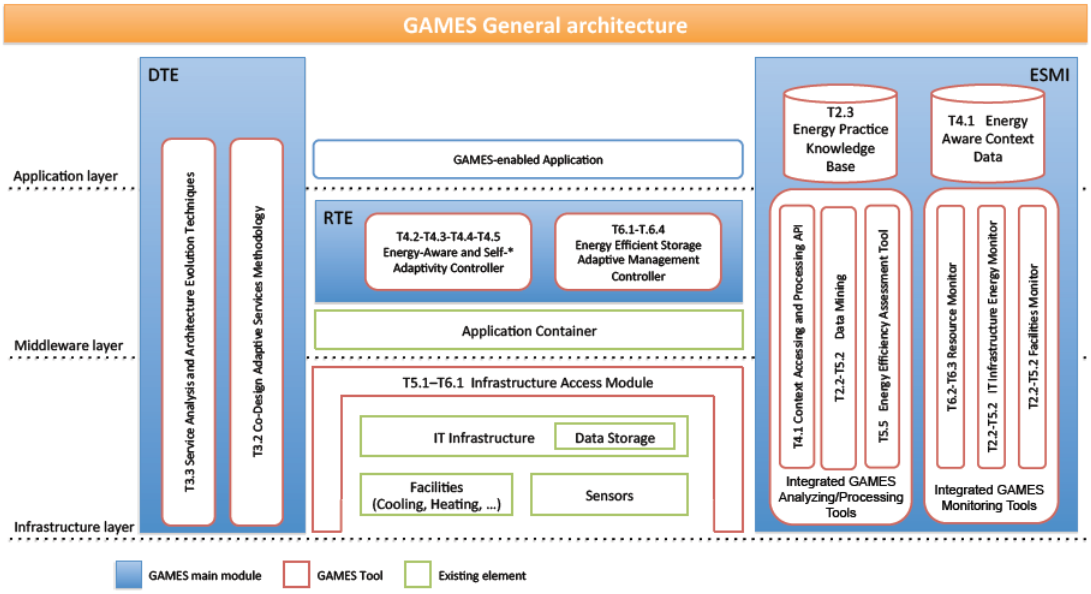

Figura 2.3: Architettura generale del progetto GAMES

La Figura 2.3 mostra l'architettura generale del progetto GAMES [JKC+10]. Per garantire una soluzione flessibile ed efficace, essa è stata disegnata come un insieme di moduli interconnessi tra di loro, ognuno specializzato in uno specifico aspetto della metodologia proposta e discussa nella Sezione 2.2.2.

In generale, il DTE (Design Time Environment) fornisce gli strumenti necessari a supportare il designer durante la definizione dei GPI e KPI rilevanti e la fase di annotazione, insomma le fasi 3 e 4 della "co-design methodology". Queste informazioni guideranno la configurazione del IAM (Infrastructure Access Module), per far sì che l'applicazione possa essere monitorata a run-time.

L'IGMT (Integrated GAMES Monitoring Tools), incluso nell'ESMI (Energy Sensing and Monitoring Infrastructure), è incaricato di raccogliere i dati necessari al calcolo dei vari GPI e KPI che poi l'IGAPT (Integrated GAMES Analyzing/Processing Tools) analizzerà per identifi-

care possibili violazioni. L'ESMI è, dunque, il sottosistema GAMES responsabile del monitoring e processing dei dati provenienti dall'infrastruttura IT del data centre.

L'RTE (Run-Time Environment) è responsabile dell'adattamento delle applicazioni in funzione in modo da migliorare la loro efficienza energetica. Sfruttando le operazioni fornite dall'ESMI, esso può investigare sul grado di soddisfazione dei vari GPI e programmare un'azione di riparazione ogni volta che è necessario. In questo modo, il DTE può ottenere nuove informazioni utili a ridisegnare l'applicazione e fare in modo che esse raggiungano i risultati attesi in termini di performance di greenness [CFP+11].

L'architettura del progetto è organizzata in tre livelli, o per meglio dire layer.

Del layer infrastrutturale fanno parte tutti gli strumenti software che hanno direttamente a che fare con macchine e dispositivi fisici. Tutto ciò che è usato per monitorare l'infrastruttura IT e l'ambiente, come sensori, misuratori di potenza, di temperatura e molto altro, si suppone già esistente e non direttamente implementato nel progetto. L'architettura prevede anche un modulo di accesso, l'IAM, che garantisce, attraverso una API[6], le operazioni necessarie per interrogare o configurare l'infrastruttura dell'IT Service Centre, i sensori ambientali e le altre apparecchiature.

Al livello di middleware, il modulo RTE sfrutta uno Standard Application Container[7], ipotizzando che esso sia presente, per garantire i servizi di adattamento necessari alle applicazioni GAMES-enabled.

Per essere in grado di adattarsi a run-time, tenendo sempre conto dell'ambiente hardware, del contenitore software e della struttura dell'applicazione stessa, le applicazioni GAMES-enabled devono essere progettate in accordo con la metodologia di progettazione fornita dal modulo DTE, nota come "co-design methodology". Questa metodologia consiste di un insieme di modelli che descrivono come il sistema dovrebbe reagire nel caso in cui qualche GPI o KPI venisse violato.

Come conseguenza di tutto questo, oltre alla classica descrizione funzionale e non, il descrittore di una applicazione GAMES-enabled porta con sé un insieme di regole che definiscono le strategie di adattamento e le relazioni tra i vari livelli che compongono l'architettura GAMES.

L'application container tiene conto solamente della descrizione funzionale e di quella non-funzionale, mentre le strategie di adattamento vengono esaminate dal modulo RTE, in modo da selezionare l'azione di riparazione adeguata per rafforzare il predetto GPI. Per questo motivo, l'RTE può essere visto come una possibile estensione dell'usuale application container.

Il modulo ESMI contiene una serie di strumenti software per:

- raccogliere dati a run-time sul consumo energetico e sulle prestazioni del sistema, come Nagios[8];

[6] Con il termine API (Application Programming Interface) si indica un insieme di procedure finalizzato ad ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra il software a basso e quello ad alto livello, semplificando così il lavoro di programmazione.

[7] Il termine "Standard Application Container" si riferisce a quella parte del middleware che offre un insieme di servizi condivisi (ad esempio di tipo transazionale o di sicurezza) che le applicazioni deployate possono utilizzare. JADE container per la telefonia mobile o Sun Glassfish per i servizi sono esempi di questi container.

[8] Nagios è una nota applicazione open source per il monitoraggio di computer e risorse di rete. La sua funzione base è quella di controllare nodi, reti e servizi specificati, avvertendo quando questi non garantiscono il loro servizio o quando ritornano attivi (<http://www.nagios.org/>).

- analizzare e processare questi dati con strumenti di data mining, come Weka/SpagoBI o Pellet reasoner.

I dati raccolti a run-time e processati dall'ESMI, ai quali solitamente ci si riferisce come a dati di contesto, fanno riferimento a valori di consumo energetico e prestazionali dell'infrastruttura IT, valori riguardo l'ambiente del sistema e, infine, valori dello stato delle applicazioni GAMES-enabled che risultano funzionanti all'interno dell'IT Service Centre. Questi dati di contesto vengono poi sfruttati, sempre a run-time, dal modulo RTE per decidere l'azione di riparazione più adeguata.

Per poter far sì che questi ultimi vengano compresi dall'RTE, è stato specificato un apposito modello di contesto. Per prima cosa, gli elementi di tale modello vengono istanziati a run-time assieme ai dati raccolti dagli strumenti dell'ESMI. Queste istanze vengono poi immagazzinate nel repository denominato Energy Aware Context Data e, successivamente, esaminate per determinare lo stato di consumo energetico e prestazionale del data centre al fine di poter dedurre nuove informazioni di contesto che potrebbero essere rilevanti per il processo di decisione.

I dati di contesto sono disponibili a tutti i moduli dell'architettura GAMES attraverso la Context Accessing and Processing API, che può essere usata sia con un approccio push, basato su eventi, sia con quello pull, su richiesta.

La Energy Practice Knowledge Base (EPKB) mantiene al suo interno le informazioni relative al consumo di energia e alle performance del sistema ottenute tramite l'esecuzione di algoritmi di mining sulla serie storica dei dati raccolti dagli strumenti ESMI.

I moduli di adattamento a run-time dell'architettura GAMES sono organizzati rispettando la metodologia MAPE dei sistemi semi-adattativi:

- Strumenti di **M**onitoring (per la raccolta di dati di contesto riguardanti l'infrastruttura IT, le apparecchiature e la sensoristica);
- Strumenti di **A**nalisi e Processing (strumenti per la rappresentazione programmatica dei dati di contesto, il reasoning, il learning e il data mining);
- **P**ianificazione e Decisione su strategie di adattamento a run-time (Energy-Aware and Self-* Adaptivity Controller e Energy Efficient Storage Adaptive Maanagement Controller);
- **E**secuzione di azioni adattative (Energy-Aware and Self-* Adaptivity Controller, Energy Efficient Storage Adaptive Management Controller and the Infrastructure Access Module).

Nel layer più alto risiedono le applicazioni sviluppate rispettando la metodologia proposta dal DTE.

Una prima semplice definizione di applicazione GAMES-enabled è stata fornita nella Sezione 2.2.1, ma ora, grazie ai nuovi elementi forniti, è finalmente possibile darne una più precisa. Si definisce, dunque, una applicazione GAMES-enabled come un processo composto da attività solitamente definite in termini dei propri requisiti funzionali e non. Tali requisiti, per una certa qual attività, possono essere soddisfatti da un insieme di servizi che agiscono su macchine fisiche o Virtual Machine.

A run-time, le istanze di processo, delle attività e dei vari servizi sono definite secondo il loro stato di esecuzione. Assieme alla più canonica definizione di un processo in termini di attività, flusso di dati, flusso di controllo e KPI, una applicazione GAMES-enabled viene descritta anche anche attraverso le informazioni riguardanti i suoi GPI e le azioni di riparazione che possono essere eseguite in caso di violazione di uno di questi GPI.

2.2.4 Goal-based model

L'adozione della "co-design methodology" proposta da GAMES ha come obiettivo dichiarato quello di rendere possibile l'esecuzione flessibile di tutte le applicazioni influenzate dai valori dei loro GPI e KPI.

Qualora si verificasse la violazione di uno di questi indicatori, sarebbe necessario individuare l'inefficienza registrata e abilitare un'azione di riparazione. Per poter comprendere le cause di queste inefficienze e supportare la selezione dell'azione di riparazione, si è deciso di modellare i requisiti del data centre usando un modello Goal-based, naturale estensione del framework TROPOS[9]. Questo framework definisce tre concetti: i goal, gli event e i treatment, i quali sono organizzati in altrettanti layer a cui danno loro il nome.

Nel modello in Figura 2.4, i goal (raffigurati dagli ovali) rappresentano gli interessi strategici che gli stakeholder, o più in generale, gli attori del sistema intendono raggiungere per generare profitto. Gli event (raffigurati come pentagoni) simboleggiano circostanze incerte, tipicamente al di fuori del controllo degli attori, le quali possono avere un impatto, positivo o negativo, sul conseguimento degli obiettivi. I treatment, infine, talvolta identificati anche con il nome di task (raffigurati come esagoni) sono sequenze di azioni usate per soddisfare i propri obiettivi o per trattare gli event [AGM10].

All'interno di GAMES, è possibile modellare gli obiettivi come requisiti sui GPI o KPI definiti a diversi livelli di granularità, a partire dall'intero data centre fino al singolo componente. TROPOS permette di definire relazioni tra gli indicatori; questo significa che un indicatore può dipendere da un altro ad esso correlato. Per reagire alla violazione di un vincolo, è possibile prendere in considerazione tutte le azioni che influenzano l'indicatore relativo al vincolo in questione e conseguentemente, ma in maniera indiretta, anche quelli associati. Per ciascuno di questi è necessario definire una soglia ad ogni livello di granularità.

Gli event sono circostanze incerte. Per questo motivo, si è deciso di modellare le possibili violazioni dei GPI e KPI e tutte le altre occorrenze che richiedono un'azione di riparazione come una riconfigurazione della macchina virtuale. Nel modello è, inoltre, possibile descrivere come gli event abbiano un impatto positivo o negativo sugli indicatori.

I task, o treatment, sono la rappresentazione di tutte le azioni di riparazione considerate nell'architettura GAMES. Una delle più interessanti caratteristiche di un framework TROPOS è la possibilità di descrivere l'impatto delle azioni sia sugli event, dato che un'azione di riparazione potrebbe prevenire un event dal verificarsi, sia sulle relazioni event-goal, poiché la stessa

[9] TROPOS è una metodologia di sviluppo software, in cui i concetti del paradigma agente vengono usati durante l'intero processo di sviluppo del software. Le nozioni di agente, goal, task e dipendenza (sociale) sono usate per modellare e analizzare requisiti software necessari per l'implementazione finale del sistema.

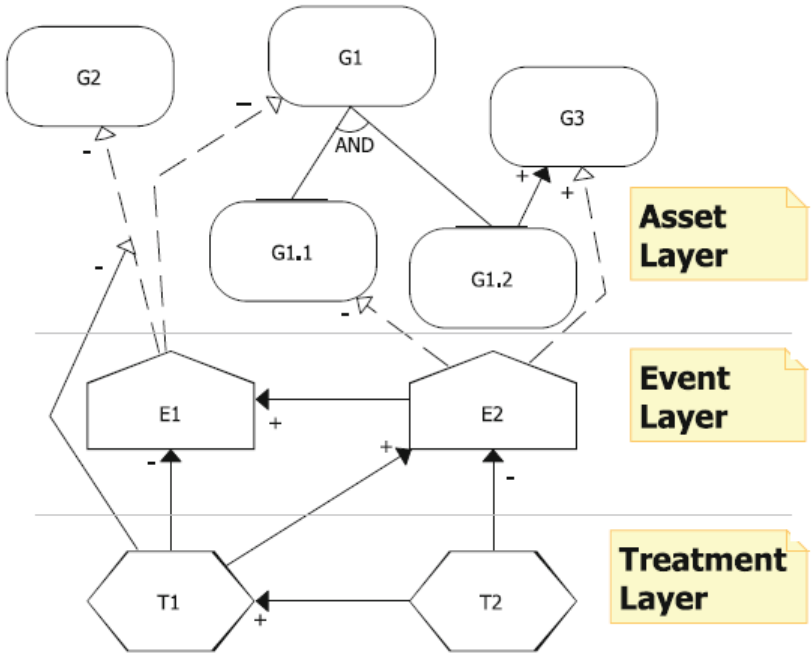

Figura 2.4: Goal-Risk framework

azione potrebbe aumentare o diminuire il rischio di verificarsi di una certa relazione. Questa caratteristica è stata ovviamente importata nel framework utilizzato da GAMES.

Questo modello giocherà un ruolo fondamentale nelle fasi seguenti del progetto di tesi. Ogni informazione da esso esplicitata graficamente sarà codificata all'interno di un database che verrà utilizzato ogniqualvolta un indicatore GPI o KPI risulterà non conforme ai vincoli stabiliti. Partendo semplicemente dal nome di questo indicatore, sarà possibile risalire agli eventi che con maggiore probabilità hanno portato al malfunzionamento del sistema e, di conseguenza, all'azione di riparazione più indicata per riportare il tutto alla normalità.

Capitolo 3

Complex Event Processing

In questo Capitolo si parlerà di Complex Event Processing, un concetto alla base dell'approccio proposto da questo lavoro di tesi.

3.1 Che cos'è il Complex Event Processing?

3.1.1 Breve storia del Complex Event Processing

Alcune delle principali innovazioni tecnologiche degli ultimi cinquant'anni dipendono da varie forme di event processing, come si può vedere dalla Figura 3.1 [Luc07]. Tra queste ci sono senza dubbio la simulazione ad eventi discreti, le reti di computer, i database e le tecnologie di middleware.

Il concetto di event processing nasce con la simulazione ad eventi discreti negli anni Cinquanta. L'idea principale era che il comportamento di un sistema, sia esso un hardware, un sistema di controllo, una catena di produzione di una fabbrica o un fenomeno naturale come il meteo, potesse essere controllato tramite un programma per computer scritto in un "linguaggio di simulazione". Presi dei dati in input, il programma avrebbe dovuto generare eventi che riproducevano le interazioni tra i componenti del sistema. Ogni evento doveva registrarsi ad un orario fissato da un orologio, e, ovviamente, alcuni eventi potevano verificarsi nello stesso istante. Alla fine l'orologio avrebbe aumentato la sua lettura tramite "ticks" discreti, che rappresentavano lo scorrere del tempo reale. Il simulatore, quindi, doveva registrare il flusso di eventi tra i componenti, l'esecuzione dei componenti, e lo scorrere del tempo dell'orologio. Le simulazioni ad eventi discreti di quel tempo stavano certamente svolgendo event processing e i modelli, infatti, erano architetture event-driven.

Un altro tipo di event processing fu coinvolto nello sviluppo delle reti di computer, a partire dagli ultimi anni Sessanta con la rete ARPANET[10]. L'obiettivo era realizzare una comunicazione affidabile tra computer attraverso reti, tramite l'uso di eventi contenenti sequenze di dati binari,

[10]ARPANET (acronimo di Advanced Research Projects Agency NETwork) venne studiata e realizzata nel 1969 dal DARPA, l'agenzia del Dipartimento della Difesa degli Stati Uniti responsabile per lo sviluppo di nuove tecnologie ad uso militare. Si tratta della forma embrionale dalla quale poi nel 1983 nascerà Internet. ARPANET fu pensata per scopi militari statunitensi durante la Guerra Fredda.

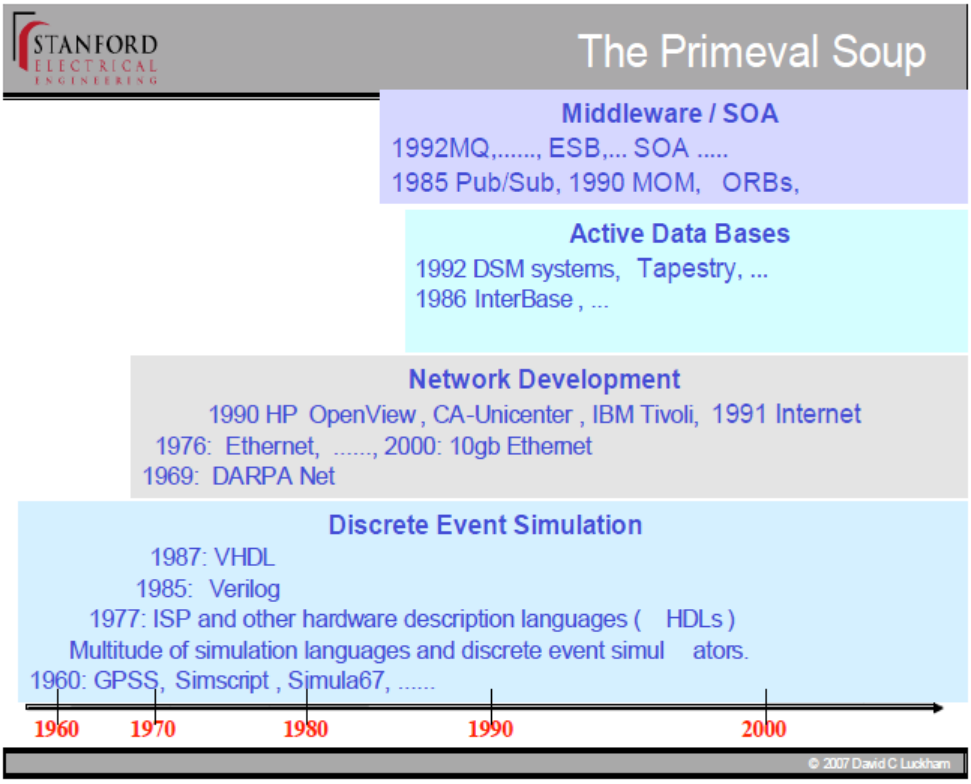

Figura 3.1: Le quattro maggiori aree interessate dall'Event Processing negli ultimi cinquant'anni

i cosiddetti pacchetti. Trasmettere o ricevere un pacchetto corrispondeva ad un evento. La parte più difficile fu senza dubbio lo sviluppo dei protocolli per poter trasmettere una sequenza di pacchetti in maniera affidabile, anche se la rete stessa era totalmente insicura e soggetta ad errori. Dagli sforzi fatti per mettere in piedi tale rete si deve la definizione di standard per i protocolli di rete che utilizzano il concetto di livello di eventi. Due esempi sono il protocollo TCP/IP[11] e il modello ISO/OSI[12] a sette livelli.

Lo sviluppo della tecnologia inerente ai cosiddetti active database ha inizio nei tardi anni Ottanta, come un miglioramento dei database tradizionali per poter venire incontro alla sempre più crescente domanda di elaborazione a real time. Essi furono estesi con capacità di event processing che permettevano loro di invocare applicazioni in risposta ad eventi. Per fare ciò, un livello di event processing fu implementato al di sopra di ogni database tradizionale: questo permetteva la definizione di eventi e di semplici tipi di schemi di eventi che potevano essere

[11] In telecomunicazioni e informatica il Transmission Control Protocol (TCP), anche chiamato Transfer Control Protocol, è un protocollo di rete a pacchetto di livello di trasporto, appartenente alla suite di protocolli Internet, che si occupa di controllo di trasmissione.

[12] In telecomunicazioni e informatica l'Open Systems Interconnection (meglio conosciuto come modello ISO/OSI) è uno standard per reti di calcolatori stabilito nel 1978 dall'International Organization for Standardization, il principale ente di standardizzazione internazionale, che stabilisce per l'architettura logica di rete un'architettura a strati composta da una pila di protocolli suddivisa in 7 livelli, i quali insieme espletano in maniera logico-gerarchica tutte le funzionalità della rete.

utilizzati come trigger[13] per le regole reattive, anche chiamate regole ECA (Event - Condition - Action).

3.1.2 Che cosa si intende per "evento"?

Prima di procedere con una vera e propria definizione di che cosa sia esattamente il Complex Event Processing, bisogna chiarire cosa si intende esattamente con il termine "evento". Un evento è un'occorrenza all'interno di un particolare sistema o dominio: esso è qualcosa che è avvenuto, o che si prevede che possa avvenire in quel dominio. La parola "evento", inoltre, è usata per rappresentare in un sistema computazionale l'occorrenza di una specifica entità di programmazione[14], come, ad esempio, la pressione di un pulsante all'interno di un'applicazione può portare allo scatenarsi dell'evento "mouse clicked" [EN11].

In altre parole, vengono dati due significati alla parola "evento". Il primo si riferisce ad un'occorrenza, a ciò che è avvenuto nel mondo reale o in un qualunque altro sistema. Il secondo, invece, si avvicina molto di più al mondo della computazione e conduce al concetto di event processing, dove la parola "evento" è usata per indicare un'entità di programmazione che mostra una sua occorrenza.

Sarebbe molto facile aprire una discussione sulle differenze tra queste due interpretazioni, ma, ai fini di questa trattazione, si può facilmente attribuire entrambe le definizioni alla parola "evento" e dedurre la più corretta tra le due semplicemente guardando al contesto. E', inoltre, interessante notare che una singola occorrenza di un evento può essere rappresentata da più entità, e, anche, che una data entità potrebbe essere in grado di catturare solo alcune specifiche sfaccettature di una particolare occorrenza di un evento.

Esistono eventi che si potrebbero definire come rumore di fondo e che, quindi, non richiedono alcuna reazione. Gli eventi che, invece, ne richiedono vengono spesso chiamati "situazioni". Si definisce una situazione come l'occorrenza di un evento che potrebbe richiedere una reazione. Una delle tematiche principali legate all'event processing è l'identificazione e il riscontro di situazioni, così che possano essere garantire le eventuali reazioni. Una reazione potrebbe essere, semplicemente, rispondere al telefono o aggiungere un oggetto alla lista della spesa, oppure potrebbe consistere in qualcosa di più complesso: nel caso in cui una persona perda l'aereo esistono svariate alternative di reazione a seconda dell'ora e del giorno, dell'aeroporto in cui si trova, delle regole della compagnia e del numero di altri passeggeri nelle sue stesse condizioni.

3.1.3 Che cosa si intende per "event processing"?

L'event processing è un processo di computazione che compie operazioni su eventi [EN11]. Le più comuni sono l'analisi, la creazione, la trasformazione e la cancellazione. Esistono due tematiche principali all'interno di quest'area:

[13] Per trigger, nelle basi di dati, si intende una procedura che viene eseguita in maniera automatica in coincidenza di un determinato evento.

[14] Questo termine viene volutamente usato per rendere chiaro che non si sta necessariamente parlando di oggetti, così come vengono definiti nell'ambito della programmazione ad oggetti: in alcuni contesti, infatti, è possibile incontrare eventi come oggetti di programmazione object-oriented, ma è anche possibile che appaiano in forme diverse, come record di un database, strutture in un linguaggio come C o COBOL, o messaggi trasmessi tra sistemi.

- Il design, la struttura del codice e il funzionamento delle applicazioni che utilizzano gli eventi, sia direttamente che indirettamente. Ci si riferisce a ciò come programmazione event-based, sebbene più spesso viene utilizzata la dicitura architettura event-driven.

- Le operazioni di elaborazione che è possibile eseguire su eventi come parti di una certa applicazione. Questo include il filtraggio di certi tipi di eventi, il cambiare l'istanza di un evento da un tipo ad un altro ed esaminare un insieme di eventi al fine di riscontrare un modello specifico. Queste operazioni spesso fanno sì che nuove istanze di eventi vengano generate.

E' possibile, ovviamente, scrivere programmi event-based senza ricorrere esplicitamente ad operazioni di event processing. Le tre principali differenze che distinguono l'event processing dalla semplice programmazione event-based, e che aprono ad un ricco insieme di possibilità, sono:

- Astrazione - Le operazioni che compongono la logica dell'event processing possono essere separate dalla logica applicativa, permettendone così la modifica senza toccare le applicazioni che producono e consumano gli eventi.

- Disaccoppiamento - Gli eventi riscontrati e prodotti da una specifica applicazione possono essere utilizzati e consumati da applicazioni totalmente differenti. Non c'è alcun bisogno per le applicazioni che producono e quelle che consumano eventi di essere a conoscenza ognuna dell'esistenza dell'altra; non a caso esse possono essere dislocate in qualunque parte del mondo. Un evento emesso da una singola applicazione produttrice può essere utilizzato da più macchine consumatrici di eventi e, viceversa, si può far sì che una applicazione consumi eventi prodotti da più applicazioni generatrici di eventi.

- Obiettivo incentrato sul mondo reale - L'event processing ha spesso a che fare con eventi che si verificano, o che dovrebbero verificarsi, nel mondo reale. La relazione dell'event processing con il mondo può avvenire in due modi: in maniera deterministica, ovvero tramite un perfetto mapping tra la situazione del mondo reale e la sua implementazione nel sistema di event processing, oppure in maniera approssimata, dove, appunto, viene approssimata nel sistema una situazione reale.

3.1.4 Il concetto di event processing nell'ambito della tecnologia IT

Ora che è più chiaro cosa si intende per Complex Event Processing, è bene inquadrare tale concetto all'interno dell'ambiente IT. Verranno, quindi, approfondite le seguenti aree: Business Process Management (BPM), Business Activity Monitoring (BAM), Business Intelligence (BI), Business Rule Management Systems (BRMSs), Network and Systems Management (NSM), Message-oriented Middleware (MOM) e Stream Computing.

Il Business Process Management si occupa di supporto computerizzato per la modellazione, gestione, orchestrazione ed esecuzione di alcuni o tutti i processi di business di un'impresa. I software BPM si sono evoluti da sistemi di workflow quali erano in precedenza, e ora sono spesso inclusi in piattaforme con architettura service-oriented. Le sinergie tra BPM e l'event processing consistono nel fatto che:

- Il sistema BPM può servire da produttore, generando eventi che riportano lo stato dei cambiamenti all'interno del sistema stesso, i quali vengono poi analizzati da un sistema di event processing così che possano essere ritornati al sistema, oppure inviati alle altre applicazioni, gli eventuali eventi derivati;

- Il sistema BPM può agire come un consumatore di eventi, reagendo a situazioni individuate dal sistema di event processing dopo che questo ha analizzato gli eventi fuori dal sistema BPM. Un evento inviato da un sistema di event processing può interagire con il sistema BPM in molti modi: l'evento può generare una nuova istanza del processo di business, può influenzare il punto di decisione all'interno del flusso di di un processo di business già attivo, o può causare la terminazione di un'istanza di processo.

Al momento esistono già molti sistemi BPM che possiedono al loro interno un proprio sistema di event processing. Questa caratteristica è destinata a diffondersi sempre di più visto l'importanza crescente dell'event processing.

Business Activity Monitoring (BAM) è un termine coniato dalla Gartner Inc[15]. Il concetto dietro questa tecnologia è, tuttavia, più grande del semplice tener traccia dei KPI, una delle operazioni più comuni per un processo di questo genere, e può includere ogni tipo di evento di business. Non a caso i software BAM contengono alcune funzionalità di event processing: i sistemi che si concentrano sul monitoraggio dei KPI sono in grado di compiere operazioni di filtraggio, trasformazione e, cosa più importante di tutte, aggregazione di eventi.

I Business Rule Management Systems (BRMS) sono sistemi software che eseguono regole, tipicamente in forma di "condizione-azione", o "if-then", le quali sono tenute separate dalla parte principale della logica dell'applicazione. Questo significa che tali regole possono essere modificate senza dover cambiare il codice dell'applicazione. Esse sono espresse tramite linguaggi dichiarativi e sono gestite da piattaforme software dedicate.

I BRMS e i sistemi di event processing hanno differenze sostanziali:

- L'event processing viene invocato attraverso l'occorrenza di eventi, mentre le regole di business sono invocate tramite le richieste fatte dalla logica applicativa;

- Le regole di business operano sugli stati; l'event processing, invece, opera su eventi, ma può consultare lo stato;

- Un sistema basato su regole di business inferisce a partire da una base nota di regole di business, mentre la funzionalità principale di un sistema di event processing è il filtraggio, la trasformazione e l'identificazione di modelli.

Queste differenze portano dunque a diverse funzionalità, differenti meccanismi di esecuzione e differenti tipi di ottimizzazione.

Le applicazioni di tipo Network and System Managemenent (NSM) sono guidate da eventi. Uno dei principali obiettivi di questo tipo di applicazioni è quello di monitorare eventi di errore, talvolta chiamati "sintomi", così da poterli analizzare e risalire alle cause dell'errore. Gli NSM

[15]Gartner Inc. è una società multinazionale leader mondiale nella consulenza strategica, ricerca e analisi nel campo dell'Information Technology.

usano una tecnica chiamata "correlazione" per esaminare i sintomi e identificare gruppi con una radice comune. Questi sistemi sono storicamente più vecchi di molti software general-purpose di event processing, ma hanno continuato ad evolversi in parallelo con essi.

I sistemi di Middleware Message-Oriented (MOM) complementano l'event processing, e in un alcuni casi vanno anche a sovrapporsi a questo. MOM garantisce un livello di trasporto che l'event processing può utilizzare come infrastruttura per implementare canali di eventi. La funzionalità di filtraggio tipica dell'event processing è simile a quella di MOM, così come la funzionalità di trasformazione. Ma ci sono anche delle differenze:

- Mentre l'istanza di un evento può essere rappresentata con un messaggio, un messaggio non rappresenta necessariamente un evento. Per esempio, inviare un'immagine tramite un messaggio usando MOM non rappresenta esattamente un evento;
- Anche se i messaggi nei sistemi di middleware message-oriented possono avere svariati tipi di semantica temporale, come ad esempio timestamp, scadenza e ordinamento, essi non possono essere riconosciuti come requisiti forti. Al contrario, le proprietà temporali sono fondamentali nell'event processing;
- MOM tipicamente gestisce ogni messaggio separatamente, mentre l'event processing solitamente include funzioni che operano su insiemi di eventi, come per esempio, le funzioni di aggregazioni e di riconoscimento di schemi o modelli.

L'event processing può essere implementato usando una combinazione di MOM e componenti di event processing dedicati, usando MOM per instradare i messaggi di eventi verso i componenti di event processing e poter così garantire il filtraggio.

Il termine "event stream processing" è qualcosa che viene utilizzato come sinonimo di event processing, ma il più delle volte risulta un soggetto molto più ampio, comprendendo i flussi che contengono i dati che di solito non vengono visti come eventi, quali i flussi video e audio. L'event processing si relaziona e interagisce con lo Stream Computing in vari modi: ad esempio, si può utilizzare una piattaforma di stream processing per implementare la funzionalità di event processing desiderata.

3.2 Caratteristiche principali del Complex Event Processing

3.2.1 Principi fondamentali del Complex Event Processing

Rispetto all'interazione request-response, gli eventi differiscono nella seguente maniera: un evento è l'indicazione di qualcosa che è già accaduto, mentre una richiesta, come suggerisce il nome, esprime la volontà del richiedente che qualcosa di specifico si verifichi nel futuro.

Come risultato si usa una terminologia leggermente diversa per i partecipanti ad una interazione che coinvolge eventi. Invece di parlare di richiedente di servizi e di service provider, si parlerà di event producer, o generatore di eventi, e event consumer, o consumatore di eventi.

Guardando a due semplici esempi, se si consulta un servizio per la prenotazione online di un volo aereo per cercare quali voli sono disponibili in data specificata, si sta inviando al servizio una richiesta, ma quando l'aereo effettivamente decolla, allora si parla di evento; supponendo, invece, che si voglia vendere qualcosa attraverso un sistema di trading online, la propria interazione con il sistema può essere vista sia come una richiesta, perché ci si aspetta di vendere l'oggetto messo in asta, sia come un evento, poiché è stata effettivamente creata un'asta. E' chiaro, dunque, come sia implicitamente permesso che vi siano più consumatori per un unico evento.

In alcuni casi, è possibile far sì che un evento mostri le occorrenze sottostanti in maniera completa. Si consideri, ad esempio, un sensore di temperatura che invia un evento ogni minuto. L'evento potrebbe contenere tutto ciò che c'è da sapere: la posizione del sensore, l'ora del giorno e il valore della temperatura. Tuttavia, in altri casi, esistono molte variabili di una occorrenza che si vorrebbero osservare, ma che l'evento in questione non è in grado di mostrare. Si pensi ad un sistema di monitoraggio di un paziente in un ospedale che deve essere in grado di effettuare molteplici misurazioni sulle condizioni del paziente, ma che, quando viene interrogato, riporta solo le misurazioni che sono rilevanti per il trattamento del paziente.

Un'altra possibilità è che una singola occorrenza possa essere rappresentata da più di una istanza di evento. Ciò si verifica quando più applicazioni sono interessate a diversi aspetti della medesima occorrenza. Si consideri, a tale scopo, l'occorrenza di un evento come l'assunzione di un nuovo impiegato. L'applicazione che gestisce il libro paga dovrebbe essere interessata a cose come il nome dell'impiegato, il suo identificativo, il livello gerarchico all'interno dell'impresa e il salario di partenza, mentre l'applicazione che si occupa delle polizze assicurative è più interessata alla sua anamnesi familiare e al suo stato di salute corrente.

Gli eventi vengono talvolta utilizzati per indicare cambiamenti di stato, spesso all'interno di sistemi di monitoraggio. Un sistema monitorato viene rappresentato come un insieme di risorse, che possono essere fisiche, come sensori o altri tipi di apparecchiature di rilevazione dati, o logiche, quali processi di business, e ad ognuno dei quali è associata una informazione di stato. Solitamente esiste un modo per interrogare ogni risorsa direttamente, così da poter leggere il suo stato, ma è anche possibile che le risorse agiscano da generatori di eventi e ne inviino uno ogniqualvolta uno o più dei loro stati interni subisce un cambiamento. Questo permette alle applicazioni di monitoring di essere immediatamente avvisate se succede qualcosa, senza così dover effettuare costantemente polling[16] sulla risorsa desiderata.

Un evento è per definizione totalmente indipendente dal suo generatore e dal suo consumatore, e questo permette che gli event producer e gli event consumer siano totalmente disaccoppiati gli uni dagli altri. L'idea di utilizzare l'evento stesso come mezzo per disaccoppiare l'event producer e l'event consumer è una sostanziale differenza tra la programmazione event-based e il design applicativo basato su interazioni di request-response, a cui si è già accennato nella Sezione 3.1.3. Esiste chiaramente un certo grado di disaccoppiamento anche nelle interazioni request-response; tuttavia, il richiedente del servizio dipende sempre dal service provider che

[16] Il polling è la verifica ciclica di tutte le unità di input/output da parte del sistema operativo di un personal computer tramite test dei bit di busy associati ad ogni periferica, seguita da un'eventuale interazione (scrittura o lettura). Questa attività impegna molto del tempo di funzionamento del processore (CPU), rallentando di conseguenza l'intero sistema. Nel contesto di questa trattazione la verifica avviene, ovviamente, sullo stato della risorsa.

garantisce la funzionalità concordata. Nell'event processing, invece, si può avere mutuo disaccoppiamento del produttore e del consumatore. Tutto ciò si può riassumere nel *principio di disaccoppiamento*[EN11], il quale sostiene che:

> "In un sistema di event processing disaccoppiato, un event producer non dipende da un particolare flusso di processo o corso di azioni che sono state intraprese dall'event consumer. In più, un event consumer non dipende dall'attività di processing eseguita da un event producer, se non da quella della produzione dell'evento in sé."

In un sistema disaccoppiato ci può essere, dunque, più di consumatore di un evento e l'azione intrapresa, se ce n'è una, può variare significativamente a seconda del consumatore o del ciclo di vita dell'applicazione. Dato che un event producer non conosce ciò che l'event consumer farà con l'evento, o quanti consumer riceveranno tale evento, non ha senso per lui aspettare una risposta per esso. Sarebbe bene, però, far notare che, nonostante l'event producer non si aspetti nessuna risposta da o dagli event consumer, in alcune situazioni questo è parte di una applicazione più grande che si aspetta di ricevere eventi in ingresso.

Un corollario del principio di disaccoppiamento è che un evento può, e di solito deve, avere significato al di fuori del contesto di una particolare interazione tra il suo producer e il suo o i suoi consumer [EN11].

3.2.2 Architettura di event processing

Non tutte le applicazioni di event processing sono uguali tra di loro, ovviamente, ma la maggior parte di queste hanno una struttura riconducibile a quella in Figura 3.2 [EE11].

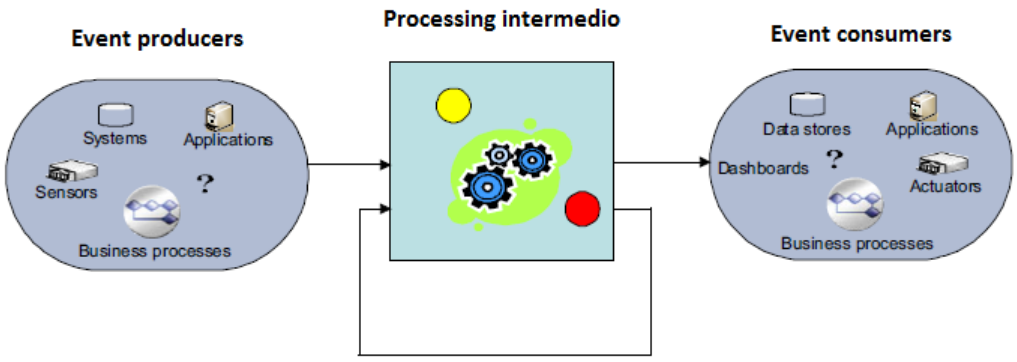

Figura 3.2: Architettura generale di event processing

Un'applicazione può contenere uno o più generatori di eventi, anche chiamati, in linea con quanto detto fino ad ora, event producer. Gli event producer possono apparire in una vasta gamma di forme e dimensioni: per esempio, sensori hardware che producono eventi quando rilevano un certo tipo di occorrenza, ma anche semplici bit di strumentazione software che produce eventi quando alcune condizioni di errore vengono segnalate o, ancora, bit di controllo di una applicazione.

La controparte degli event producer sono gli event consumer. Questi componenti sono coloro che ricevono gli eventi e, tipicamente, agiscono su di essi. Le loro funzionalità possono variare: essi sono in grado, ad esempio, di immagazzinare gli eventi per un uso futuro, mostrarli in una interfaccia utente, o decidere eventuali comportamenti reattivi dopo averli ricevuti.

Gli event producer e gli event consumer sono collegati a meccanismi di distribuzione degli eventi, identificati dalle frecce in Figura 3.2 e, solitamente, qualche meccanismo di event processing intermedio viene posto tra i suddetti. La distribuzione è garantita mediante l'uso di una tecnologia di instradamento dei messaggi asincrona; per questo motivo, spesso con un leggero abuso di notazione, si parla di un producer che invia un evento o di un consumer che riceve un evento. Ciò nonostante, si possono utilizzare altri meccanismi, come far sì che l'event producer scriva semplicemente le varie occorrenze degli eventi su un file di log, il quale viene poi letto successivamente dagli event consumer. Il meccanismo di distribuzione degli eventi è solitamente one-to-many per far sì che ogni evento, dopo essere stato inviato, possa essere ricevuto da più event consumer.

Per quel che riguarda i sistemi di event processing intermedi, nei casi più semplici il processing può semplicemente essere ricondotto ad una attività di routing con o senza filtraggio.

Una delle caratteristiche più importanti nelle architetture event-driver è il fatto che i sistemi intermedi sono in grado di generare eventi addizionali. Tali eventi possono essere distribuiti ai vari event consumer, ma possono anche essere anche soggetti ad ulteriori sistemi di event processing, così come è suggerito dalla freccia di feedback nella Figura 3.2. Il diagramma mostra il sistema intermedio come un componente monolitico. Nella pratica comune, invece, i sistemi intermedi sono composti da una serie di sotto-componenti, a cui ci si riferirà con il termine "event processing agent" [EN11], o agenti di event processing.

E' importante notare il disaccoppiamento, di cui si è già parlato, tra gli event producer e gli event consumer. Gli eventi sono il centro dell'attenzione di tutto il sistema. L'event producer ha una relazione con ogni evento che produce, invece di una relazione con gli event consumer. Non è importante per esso sapere quanti event consumer ci sono per i suoi eventi, e non gli interessano nemmeno le azioni che questi ultimi potrebbero prendere una volta ricevuti gli eventi. Allo stesso modo gli event consumer reagiscono all'evento in sè, e non all'atto della ricezione di un evento da uno specifico event producer, anche se in alcuni casi tale informazione può rivelarsi fondamentale per la corretta reazione del componente.

Un event producer si può definire come una entità posta ai margini del sistema di event processing che introduce eventi nel sistema. Questa definizione può sembrare poco calzante ad una prima lettura, specie per il fatto che sono state omesse alcune altre importanti azioni che un event producer dovrebbe compiere, come la rilevazione dell'occorrenza che da origine ad un evento, la creazione di un oggetto che andrà a rappresentare tale evento e l'introduzione di dati in tale oggetto. Essa è la più generale possibile semplicemente perché non è ovvio che tutte queste operazioni elencate debbano svolgersi all'interno dell'event producer; basti pensare al fatto che l'event producer potrebbe essere un proxy che trasmette eventi a qualche altra entità. Inoltre, e forse ancora più importante, il resto del sistema non può sapere come un event producer genera i suoi eventi: ciò che è importante è che esso li generi e basta.

Una definizione formale di event consumer, richiamando quella appena proposta per l'event producer, potrebbe essere quella di un'entità ai margini del sistema di event processing che riceve messaggi da questo.

Un event processing agent, invece, si può definire come un modulo software che compie operazioni sugli eventi. Esso riceve e invia eventi, o può generarne degli altri, perciò, ad un certo modo, si può dire che produce e consuma eventi. Tuttavia, sarebbe sbagliato riferirsi ad un event processing agent come ad un event producer unito ad un event consumer, dato che questa terminologia è riservata alle entità che si trovano ai margini del sistema di event processing vero e proprio.

E' possibile distinguere anche due tipologie di eventi: eventi grezzi e eventi derivati. Si definisce evento grezzo un evento introdotto in un sistema di event processing da un event producer. Tale definizione si basa solamente sulla sorgente di tale evento e non sulla sua struttura: un evento grezzo può o meno essere composto da altri eventi. Un evento derivato, invece, si definisce come un evento generato dal sistema di event processing come risultato di una computazione. Un oggetto evento può essere generato in un sistema di event processing, e perciò esso sarà un evento derivato in quel sistema, dopodiché può essere passato ad un altro sistema di event processing dal quale viene considerato un evento grezzo.

3.2.3 Reti di event processing

Come è già stato detto in passato, l'architettura del sistema intermedio di event processing, in Figura 3.2, di solito non è monolitica; piuttosto, è composta da un certo numero di event processing agent. Tali agent vengono implementati attraverso uno specifico linguaggio di event processing, di cui ne esistono svariati tipi. Alcuni di questi sono:

- Linguaggi rule-oriented che usano regole di produzione
- Linguaggi rule-oriented che usano regole attive
- Linguaggi rule-oriented che usano regole logiche
- Linguaggi di programmazione imperativi
- Linguaggi stream-oriented che sono estensioni di SQL
- Altri linguaggi stream-oriented

Tornando a come i vari agent sono collegati tra di loro, si definisce rete di event processing (EPN) un insieme di event processing agent, producer, consumer e elementi di stato, di cui si parlerà a breve, connessi tramite un insieme di canali.

Una rappresentazione grafica di rete di event processing è proposta in Figura 3.3 [EN11]. Essa mostra tutti i componenti introdotti sino ad ora. Ai margini è possibile notare gli event producer e gli event consumer, con gli event processing agent posti nel mezzo. Le linee continue mostrano il flusso di eventi tra i diversi agent, e si riferiscono a canali impliciti, ma è anche modellarli esplicitamente. Ogni agent accetta in ingresso un flusso di uno o più event di input

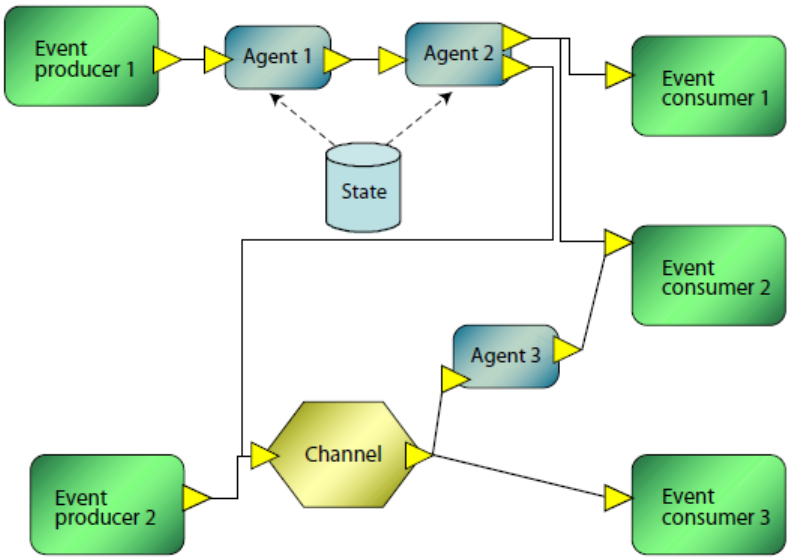

Figura 3.3: Esempio di rete di event processing

ricevuti tramite un canale e, a sua volta, emette eventi che verranno trasportati attraverso uno o più canali verso agent successivi o event consumer.

Una rete di event processing può essere usata per descrivere una singola applicazione, ma può anche servire per rappresentare più applicazioni collegate tra di loro. Supponendo che la rete di event processing in Figura 3.3 venga costruita per descrivere una singola applicazione, quest'ultima conterrà due agent (*Agent 1* e *Agent 2*) che prendono gli eventi dall'*Event producer 1* e li elaborano per produrre un flusso di nuovi eventi derivati. Supponendo, invece, di voler sviluppare una seconda applicazione di event processing che richiede lo stesso insieme di eventi derivati, invece di duplicare *Agent 1* e *Agent 2* nella seconda applicazione, è possibile estendere la rete per fare in modo che l'output di *Agent 2* venga inviato in ingresso ad altri agent in grado di compiere le elaborazioni necessarie alla seconda applicazione.

3.2.4 Tipi di sistemi intermedi

Come è già stato più volte ripetuto, una delle caratteristiche più importanti dell'event processing è l'astrazione, o separazione, della logica dell'event processing dalla logica applicativa, in modo tale che essa possa essere posizionata tra gli event producer e gli event consumer, invece di essere incorporata in essi; in questa sezione verrà presentata un'idea di rete di event processing contenente degli event processing agent con il preciso compito di implementare tale livello di astrazione.

La più semplice forma di event processing prende gli eventi dagli event producer e li distribuisce agli event consumer. Ciò include spesso operazioni di filtraggio di eventi, poiché non tutti gli eventi sono rilevanti per un specifico event consumer, oppure perché potrebbero esserci eventi

contenenti dati sensibili che un certo event consumer non è autorizzato a ricevere. Il filtraggio e l'instradamento sono garantiti dalla stessa infrastruttura che provvede alla distribuzione degli eventi, specialmente se tale infrastruttura ha capacità di publish/subscribe[17]. Accanto a queste semplici operazioni, l'elaborazione intermedia potrebbe includere la registrazione di eventi per scopi di verifica. Filtraggio, instradamento e registrazione sono tutti esempi di event processing di tipo stateless.

Un event processing agent, quindi, si definisce stateless se il modo in cui elabora un evento non influenza il modo in cui elaborerà ogni evento successivo [EN11].

Guardando a strutture più complesse, il processing intermedio potrebbe essere in grado di tradurre gli eventi, sia cambiando la loro rappresentazione sia aggiungendo informazioni, arricchendoli quindi, oppure rimuovendone, operazione spesso identificata con il termine proiezione. Le tecniche qui elencate sono simili a quelle utilizzate per l'integrazione di applicazioni d'impresa e possono essere sviluppate usando approcci e strumenti simili.

Le operazioni di cui si è appena discusso prendono un evento in ingresso e restituiscono un evento in uscita, ma è anche possibile avere event processing agent che accettano in ingresso insiemi di eventi o creano uno o più eventi come output:

- Un evento in ingresso può essere diviso in più eventi, ognuno contenente un sottoinsieme delle informazioni dell'evento originale.
- Un flusso di più eventi in ingresso può essere aggregato per produrre un flusso di eventi derivati in output. Gli eventi derivati sono tipicamente il risultato di più eventi del flusso di input: ad esempio, un evento di output può contenere il totale aggiornato di alcuni valori contenuti negli eventi di input.
- Due flussi di eventi in ingresso possono essere accorpati per produrre un unico flusso di eventi derivati in uscita.

Come si può facilmente capire, in questi esempi è stato introdotto un nuovo concetto, quello di flusso di eventi.

Un flusso di eventi si definisce come un insieme di eventi associati. Esso risulta essere spesso un insieme totalmente ordinato a livello temporale, il che significa che esiste un ordinamento basato su timestamp ben definiti. Un flusso in cui tutti gli eventi appartengono alla stessa tipologia si definisce flusso omogeneo di eventi; al contrario, un flusso in cui gli eventi sono di tipi diversi viene definito flusso eterogeneo di eventi [EN11].

I flussi di eventi possono essere un utile strumento per pensare e modellare un'applicazione di event processing. Alcuni sistemi di event processing fanno dei flussi la loro massima forma di astrazione: risulta, infatti, più naturale pensare ad un event processing agent come operante su un intero flusso di eventi, piuttosto che su un singolo evento alla volta. Il concetto di flusso è particolarmente utile in applicazioni che coinvolgono serie temporali di eventi, come la lettura periodica di un sensore.

[17] L'espressione "publish/subscribe" si riferisce a un design pattern, o stile architetturale, utilizzato per la comunicazione asincrona fra diversi processi, oggetti o altri agenti.

Agent che aggregano o compongono flussi di eventi fanno event processing di tipo stateful. Perciò, un event processing agent è detto stateful se il modo in cui elabora eventi è influenzato da più di un evento in input [EN11].

Come esempio, si supponga di avere un'applicazione che riceve un evento ogniqualvolta una certa quantità di un dato prodotto viene venduta, e si è interessati a sapere la quantità totale che è stata venduta. E' possibile usare un event processing agent per calcolare il totale corrente. Ogni volta che si riceve un evento di acquisto, l'agent emette un nuovo evento contenente il totale aggiornato.

Questo incontra la definizione precedente di event processing agent di tipo stateful, dato che ogni evento emesso da un agent dipende da tutti gli altri eventi prima ricevuti. Si noti che un agent stateful non emette necessariamente un evento derivato ogni volta che riceve un evento in input. Si supponga, ad esempio, che il volume di acquisto sia elevato. Si potrebbe decidere di non emettere un evento quando un ordine di acquisto viene completato, ma di emetterne uno al raggiungimento di una certa soglia di volume di prodotto venduta o, al raggiungimento di un certo numero di ordini di acquisto. Si potrebbe, inoltre, essere interessati a calcolare una media corrente degli ultimi dieci o cento ordini di acquisto in termini di merce venduta. Per permettere ciò, gli agent di tipo stateful devono operare solo su un sottoinsieme ridotto di tutti gli ordini di acquisto, spesso identificato dal termine finestra [EN11].

3.2.5 Modellazione di reti di event processing

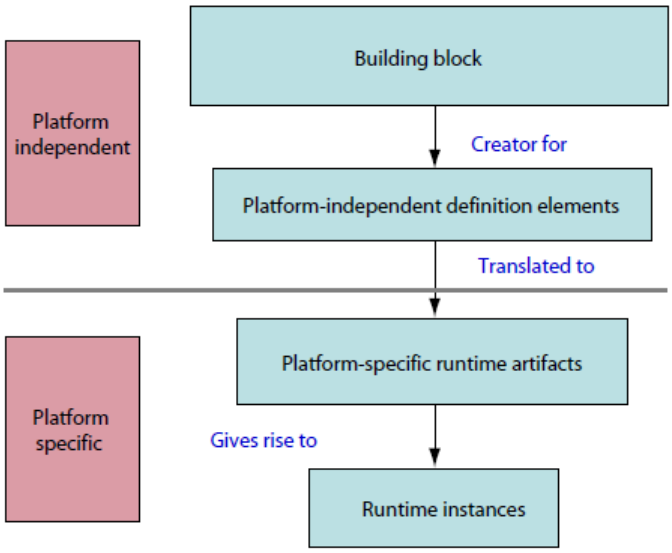

Figura 3.4: Building blocks, elementi descrittivi di rete, istanze runtime e loro correlazione

Per la modellazione di una rete di event processing si utilizzerà il linguaggio proposto dal libro, già più volte citato durante questa trattazione, "Event Processing in Action" di Eztion e

Niblett, grazie al quale è possibile descrivere una EPN come un insieme di elementi platform-independent[EN11]. Ogni elemento è un'istanza di uno dei sette "building blocks" base. Fino ad ora sono stati introdotti event processing agent, event producer, event consumer e canali di eventi; tutti questi sono da considerarsi building blocks del linguaggio di modellazione. L'astrazione dei building blocks permette di concentrarsi sulle caratteristiche realmente importanti dell'applicazione, e non sui dettagli implementativi.

Un building block rappresenta un concetto di event processing ed è usato per rappresentare elementi platform-independent descrittivi della rete, i quali sono istanze non dipendenti dall'implementazione dello stesso building block. Quando un'applicazione viene sviluppata, gli elementi descrittivi della rete devono essere tradotti in uno o più artefatti platform-specific, vale a dire utilizzabili solo da una particolare piattaforma codificata tramite un particolare linguaggio di programmazione, usando gli strumenti messi a disposizione da essa. Una volta completata, l'applicazione sarà eseguita e verranno prodotte le istanza runtime di questi artefatti platform-specific. La Figura 3.4 mostra la relazione tra building blocks, elementi descrittivi della rete e istanze a runtime [EN11].

Nonostante esistano diverse tipologie di building blocks, esse hanno tutte una struttura simile. Ogni building block ha un nome, e questo determina il tipo di elementi descrittivi platform-independent che descrive. Inoltre, possiede un'icona che può essere usata per rappresentare questi elementi in rappresentazioni grafiche.

Per poter creare un elemento descrittivo a partire da un building block, è necessario fornire alcune informazioni. Il building block descrive quali informazioni bisogna fornire, come ad esempio i terminali di input e quelli di output, lo schema di instradamento o, anche, asserzioni sulla qualità del servizio nel caso si tratti di canali.

Questi building block forniscono i componenti essenziali, che messi insieme possono dar vita ad applicazioni event-driven. Come già anticipato, esistono sette tipi di building block ed è possibile vederli tutti in Figura 3.5 [EN11]. Alcuni di questi contengono riferimenti ad altri; queste relazioni sono identificate dalle frecce entranti e uscenti dai vari blocchi.

Ogni applicazione event-driven richiede uno o più tipi di evento e, come il nome suggerisce, il building block *event type* permette di descrivere tali tipi di eventi. Questo building block definisce la struttura di un evento, a volte anche detto schema di un evento, assieme ad un po' della sua semantica.

I building block *event producer* e *event consumer* sono utilizzati per rappresentare i concetti già associati a questi nomi precedentemente. Il primo definisce un'entità che genera eventi e li immette all'interno della EPN, mentre il secondo raffigura un'entità che li riceve. Questi building blocks modellano solo quei bit relativi al comportamento dell'event producer e dell'event consumer che sono visibili agli altri componenti di una rete di event processing. Perciò il building block *event producer* non specifica come un'istanza di event producer generi un evento, e il building block *event consumer* non specifica quali azione compie un'istanza di event consumer quando riceve un evento.

L'elemento descrittivo di rete event producer o event consumer può rappresentare sia una singola istanza di produttore o consumatore di eventi, sia un'intera classe di tali istanze. In

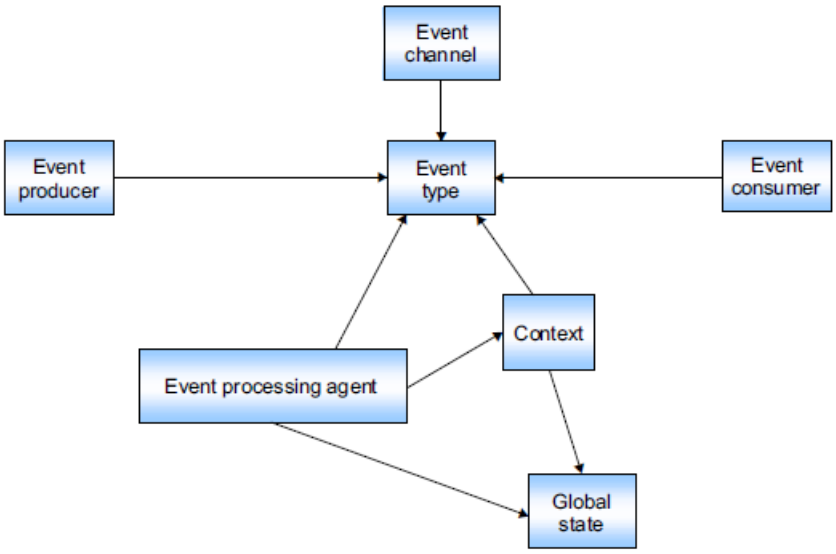

Figura 3.5: I sette tipi fondamentali di building block

alcune applicazioni potrebbe bastare una sola istanza di event producer, come nel caso in cui il produttore di eventi sia un firewall che invia messaggi di allerta; in altri casi ci potrebbero essere più istanze, per esempio nel caso di rilevatori di fumo all'interno di un edificio.

Un elemento di tipo *context* riunisce un insieme di condizioni di varie tipologie (temporali, spaziali, segmentation-oriented e state-oriented), dando così la possibilità di dividere in categorie le istanze dei vari eventi e poterle più facilmente instradare verso le istanze di agent appropriate. Per esempio, è possibile utilizzare un context di tipo segmentation-oriented per far sì gli eventi relativi a diversi clienti vengano gestiti da diverse istanze di un event processing agent.

Un elemento di tipo *global state* si riferisce ai dati che sono disponibili per l'uso sia agli event processing agent sia ai vari elementi di contesto. Questi dati possono essere paragonati a variabili globali di sistema, usate per arricchire eventi vecchi e nuovi.

Le due tipologie di building block rimaste, *event processing agent* e *event channel*, richiedono una trattazione più articolata per via della loro complessità. Per questo motivo sono state volutamente tralasciate fino a questo momento, nonostante i concetti da loro espressi siano già stati esposti nell'arco della trattazione precedente.

Il building block *event processing agent* rappresenta un pezzo di logica intermedia di event processing posta tra gli event producer e gli event consumer. Esistono molti tipi di event processing agent, e per questo esistono molte varianti di building block event processing agent, come si può vedere in Figura 3.6 [EN11]. Essa mostra la gerarchia di ereditarietà che esiste tra i vari EPA. Filtraggio, trasformazione, e rilevazione di schemi sono tutte specializzazioni degli event processing agent:

- Agent di filtraggio - Sono usati per eliminare eventi non interessanti dal punto di vista degli event consumer. L'agent di filtraggio prende un oggetto evento in ingresso e applica

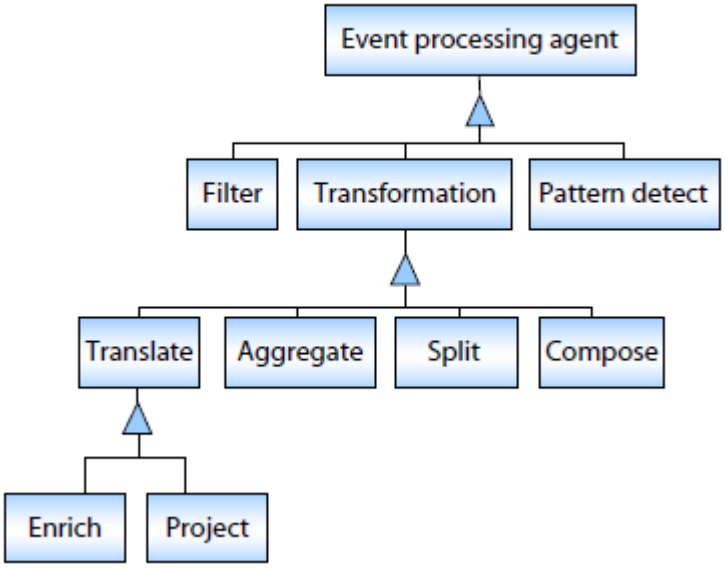

Figura 3.6: I diversi utilizzi di un event processing agent

un test per decidere se scartarlo o se inoltrarlo ad un agent seguente. Il test solitamente è stateless, in altre parole si basa solamente sul contenuto dell'istanza dell'evento. Un esempio potrebbe essere un test che scarta gli eventi in relazione a transazioni con un valore superiore a 100€.

- Agent di rilevazione di schemi - Questi accettano in ingresso insiemi di oggetti evento e li esaminano per vedere se è possibile riconoscere il verificarsi di un particolare schema. Per esempio, si può utilizzare un agent di rilevazione di schemi per scartare gli eventi relativi al primo caso di due logon consecutivi falliti, ma passare tutti i successivi. Questi agent possono generare eventi derivati che descrivono lo schema che hanno rilevato invece di passarlo assieme agli oggetti evento in ingresso.

- Agent di trasformazione - Questi agent modificano il contenuto degli oggetti evento che ricevono.

Gli agent di trasformazione possono essere ulteriormente classificati basandosi sulla cardinalità dei loro ingressi e delle loro uscite:

- Agent di traduzione - Essi prendono ogni oggetto evento in ingresso e operano su di esso, indipendentemente dagli eventi precedenti o successivi. Essi forniscono un'operazione di tipo "single event in - single event out".

- Agent di split - Gli agent di split prendono un singolo evento in ingresso e generano un flusso di più oggetti in uscita, fornendo così un'operazione di tipo "single event in - multiple event out".

- Agent di aggregazione - Con questi agent un flusso di oggetti evento in ingresso produce un evento di output che è funzione degli eventi in ingresso. In questo modo viene fornita un'operazione di tipo "multiple event in - single event out".
- Agent di composizione - Gli agent di composizione prendono due flussi di oggetti evento in ingresso per combinarli in un unico flusso. Questa operazione è simile a quella di join dell'algebra relazionale, tranne per il fatto che tale operazione avviene su flussi di dati, invece che su tabelle di dati.

Due particolari tipi di traduzione appaiono abbastanza frequentemente per far sì che ad essi venga assegnato il proprio building block. Essi sono l'agent di arricchimento, il quale arricchisce, appunto, un evento oggetto con informazioni aggiuntive, come ad esempio, l'indirizzo di un cliente derivato dall'identificativo dello stesso presente nell'evento in ingresso, e l'agent di proiezione, il cui compito è quello di cancellare informazioni dall'evento in ingresso.

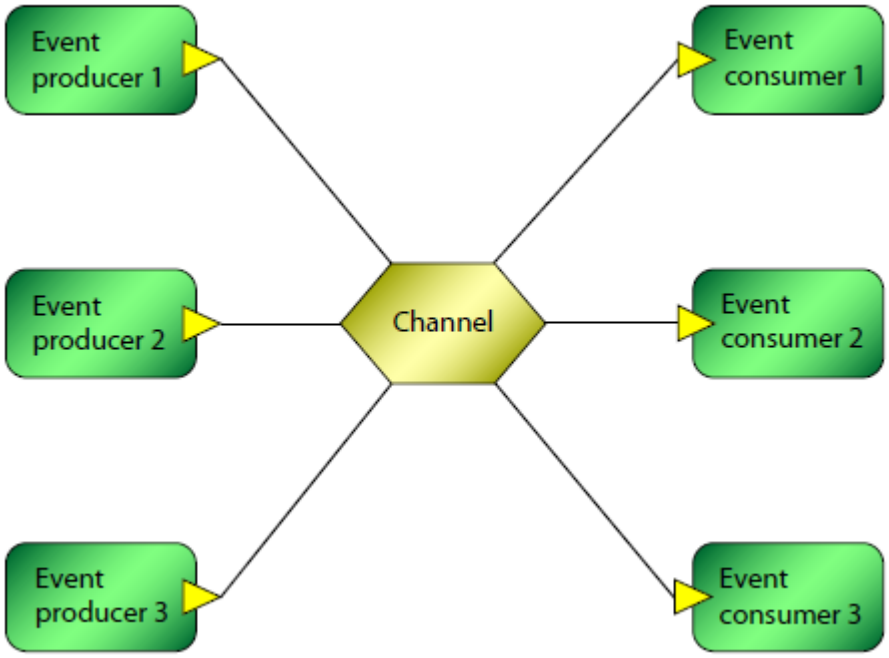

Figura 3.7: Esempio di un canale di eventi esplicitamente modellato

E' facile notare che queste descrizioni si riferiscono ad eventi in ingresso e si è spesso parlato di passare eventi ad agent successivi per ulteriori elaborazioni. Tutto ciò porta naturalmente a chiedersi come i vari event processing agent sono connessi tra di loro per formare applicazioni.

Il compito principale di un building block di tipo *event channel* è di guidare gli eventi nel percorso dagli event producer agli event consumer. Esso può essere lasciato in forma implicita, identificata da una semplice linea retta continua, quando il suo compito non è altro che quello di collegare un terminale di output con un terminale di input, ma può anche essere esplicitato tramite un componente a forma esagonale, così come è rappresentato in Figura 3.7, nel caso in

cui esso vada a migliorare l'architettura generale della rete, in termini di prestazioni, oppure nel caso in cui il canale richieda una particolare operazione al suo interno, quale ad esempio un'operazione di filtraggio [EN11].

In Figura 3.7 l'implementazione esplicita del canale si è resa necessaria per evitare di dover modellare molti più canali con capacità di relazione uno-a-uno. In questo modo è stata snellita la rappresentazione grafica della rete di event processing. Inoltre, nel caso in cui venga successivamente aggiunto un altro event consumer o event producer, l'implementazione esplicita del canale permette di aggiornare la complessità del diagramma di event processing in maniera molto semplice, aggiungendo banalmente un collegamento tra la nuova entità e il canale; la stessa operazione effettuata su una rete con canali impliciti sarebbe costata molto di più, poiché si sarebbero dovuti aggiungere n canali, dove n è il numero di event consumer presenti, nel caso in cui venga creato un event producer o m canali, dove m è il numero di event producer presenti, nel caso in cui sia stato creato un event consumer.

3.3 Implementazioni del Complex Event Processing

Il mercato dell'event processing sta facendo registrare una grande crescita negli ultimi anni. I principali competitor nel campo della Information Technology (IT) hanno ritenuto fondamentale il ruolo dell'event processing nelle applicazioni di domani, specialmente a causa della sua capacità di riconoscere particolari schemi di eventi.

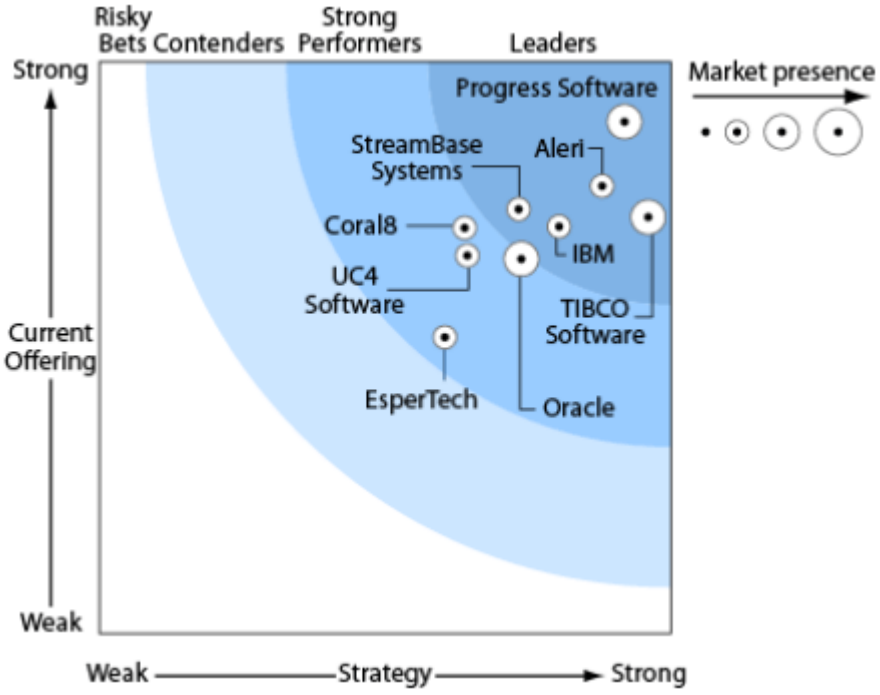

Figura 3.8: Presenza sul mercato dei principali vendor secondo Forrester Research

La Figura 3.8 illustra la presenza nel mercato dell'event processing dei principali vendor, grazie ad un studio effettuato dalla Forrester Research in data 4 Agosto 2009 [GR09]. Essa ha valutato nove piattaforme di complex event processing (CEP) usando centoquattordici criteri di paragone e i risultati evidenziano il prodotto della Progress Software e quello di Aleri come i principali leader del mercato grazie alle loro alte valutazioni, per quel che riguarda le caratteristiche del sistema di event processing e le varie categorie strategiche. Solide caratteristiche e strategie di business adeguate hanno aiutato anche IBM e Tibco Software a trovare la loro posizione di mercato. Più indietro è possibile trovare Oracle, che, secondo gli studi della Forrester Research, "garantisce un'offerta abbastanza importante da potergli garantire una posizione di leadership in futuro, ma le sue strategie complessive sono un passo indietro rispetto a quelle dei leader attuali" [GR09]. Infine, una menzione speciale viene dedicata a EsperTech, l'unica implementazione open source presente in tale classifica.

3.3.1 Progress Software

La piattaforma di event processing Progress Apama è il più ricco ambiente di sviluppo per la creazione di applicazioni di Complex Event Processing ad elevate prestazioni presente sul mercato. Adottata da alcune delle più importanti istituzioni finanziare mondiali, la ricca architettura CEP di Apama garantisce la possibilità di monitorare, analizzare e agire in tempo reale anche su eventi che necessitano di tempi di risposta molto piccoli. Con la sua avanzata capacità di event processing, Apama fornisce sofisticate operazioni a livello temporale, spaziale e logico per rispondere agli aventi in ingresso in archi di tempo nell'ordine dei millisecondi.

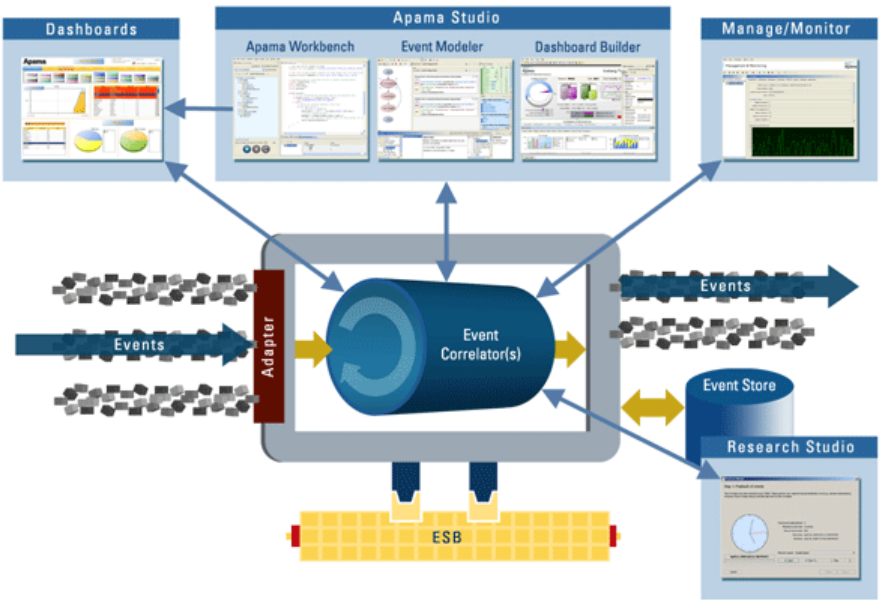

Figura 3.9: Architettura della piattaforma Progress Apama

La piattaforma di event processing Apama include anche:

- Dashboard altamente interattive così che i clienti business possano accedere ad informazioni sensibili e controllare il funzionamento delle operazioni su di esse.
- Un ambiente di integrazione che permette agli eventi di essere riconosciuti da diversi sistemi di middleware, incluso un service bus a livello di impresa, applicazioni e altri endpoint senza interagire con l'infrastruttura IT.
- Strumenti di sviluppo e di testing che permettono all'analista e agli sviluppatori di creare, testare e migliorare applicazioni di Complex Event Processing.
- Un engine[18] CEP scalabile e a bassa latenza che garantisce elevate prestazioni di event processing.

Analizzando la struttura dell'applicazione, riportata in Figura 3.9, si nota come la piattaforma Progress Apama si concentri sull'identificazione di schemi di eventi a real-time operando su flussi in ingresso. L'architettura CEP di Apama elimina la latenza dei tradizionali sistemi che salvano e indicizzano le informazioni relative agli eventi occorsi prima di analizzarle [Pro12].

La piattaforma Apama incorpora completamente tutte le potenzialità del CEP attraverso un linguaggio di event processing, definito ad-hoc, che permette di derivare eventi di business di alto livello da più flussi di eventi di basso livello. Essa è, inoltre, in grado riconoscere vincoli temporali e causali, così come definizioni di eventi composti, che spesso vengono detti "eventi complessi o di business". Disponibile con un ambiente di sviluppo basato sul linguaggio definito da Apama e Java, la piattaforma garantisce un controllo totale e un alto potere espressivo al fine di riuscire a gestire qualunque applicazione di event processing. Una volta create, tali applicazioni vengono eseguite all'interno di uno o più Event Correlator, gli elementi centrali dell'implementazione di Progress Apama, gestiti da un Apama Event Manager. Gli Event Correlator monitorano i flussi di eventi in ingresso cercando di riconoscere gli schemi definiti dall'applicazione CEP Apama. Essi supportano un meccanismo di filtraggio multi-dimensionale che rapidamente setaccia i dati dei flussi di eventi, individua gli schemi ricercati e identifica la misura correttiva appropriata, come specificato dall'applicazione, il tutto nell'arco di pochi millisecondi. Apama supporta configurazioni flessibili, con più istanze di Event Correlator disponibili, così da poter distribuire gli eventi in ingresso e gestire correttamente il load balancing[19] e la fault tolerance[20].

3.3.2 EsperTech

Esper, la soluzione software prodotta da EsperTech, è uno strumento open source di event processing per flussi di dati (ESP) e, allo stesso tempo, un motore di correlazione di eventi

[18] Con il termine "engine" ci si riferisce alle funzionalità primarie di un software, a ciò che a volte viene anche indicato con il termine "core".

[19] Il Load Balancing è una tecnica informatica che consiste nel distribuire il carico di un servizio, ad esempio la fornitura di un sito web, tra più server. Con essa si aumentano la scalabilità e l'affidabilità dell'architettura nel suo complesso.

[20] La tolleranza ai guasti (o fault tolerance, dall'inglese) è la capacità di un sistema di non subire interruzioni di servizio anche in presenza di guasti. La tolleranza ai guasti è uno degli aspetti fondamentali che costituiscono l'affidabilità di un sistema. È importante notare che non garantisce l'immunità da tutti i guasti, ma solo da quelli per cui è stata progettata una protezione.

(CEP). Esiste anche una versione a pagamento, denominata Esper Enterprise Edition che include alcune funzionalità avanzate, quali supporto JMX, una interfaccia grafica web-based detta Esper HQ, Esper JDBC e Esper HA, vale a dire un server endpoint per l'arricchimento dei dati in streaming con dati storicizzati provenienti da una base di dati e una soluzione completa per la gestione in alta affidabilità del server Esper [Esp12].

Esper permette di individuare un gran numero di situazioni che accadono in tempo reale e far scattare azioni definite dall'utente al riconoscimento di alcuni condizioni nello stream di dati. Esso è stato progettato per saper riconoscere correlazioni attraverso un grande numero di eventi, spesso milioni, che altrimenti non sarebbe possibile analizzare per via dell'elevato spazio di memorizzazione necessario per salvarli tutti all'interno di una architettura classica a database presente su disco ed esaminarli in un secondo momento.

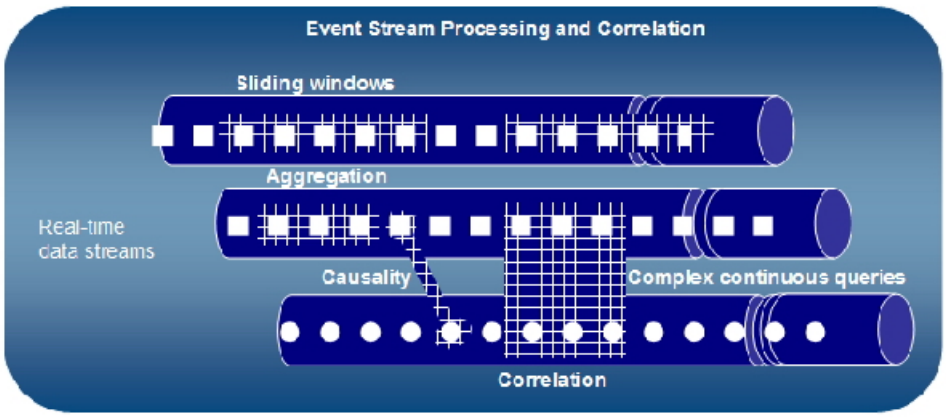

Figura 3.10: Event processing su flussi di dati e possibili correlazioni

ESP e CEP sono stati concepiti dopo anni di ricerche nel campo degli active database[21], delle continuous query[22] e dello stream processing e ora vengono largamente usati in parecchie architetture in cui il fattore critico è la latenza, come ad esempio sistemi di individuazioni di frodi o intrusioni, real-time business intelligente e CRM[23].

Esper fornisce un ricco linguaggio di interrogazione, detto Event Processing Language (EPL), in grado di filtrare, aggregare, fare join, anche su finestre scorrevoli, eventi appartenenti a più flussi. Presenta anche delle caratteristiche utili per il pattern recognition, ovvero il riconoscimento di precise sequenze di eventi a cui si può essere interessati per via del loro effetto complessivo, così da poter esprimere anche vincoli di causalità temporale tra gli eventi.

Esper supporta una vasta gamma di rappresentazioni di eventi, come i Java bean, i documenti XML, classi legacy, o semplici coppie nome-valore. Può facilmente essere incapsulato all'interno

[21] Un Active Database è un database che contiene al suo interno una architettura guidata da eventi che può rispondere alle condizioni presenti sia all'interno che all'esterno del database stesso. Alcuni dei suoi possibili usi comprendono il security monitoring, l'alerting e la raccolta statistica di informazioni.

[22] Una Continuous Query si differenzia da una normale query su database perché, a differenza di quest'ultima, essa non cerca di estrarre dati da un database che contiene informazioni precedente salvate; bensì cerca di cogliere questi dati durante il naturale scorrere del flusso di eventi all'interno del motore CEP su cui essa è stata inserita.

[23] Customer relationship management (CRM) è una strategia ampiamente implementata per la gestione del rapporto con la clientela da parte dell'azienda.

di una qualunque applicazione Java o fare da middleware per aggiungere a piattaforme esistenti la capacità di gestione degli eventi in tempo reale senza introdurre alti costi di serializzazione o latenza di rete per ogni messaggio ricevuto o azione avviata.

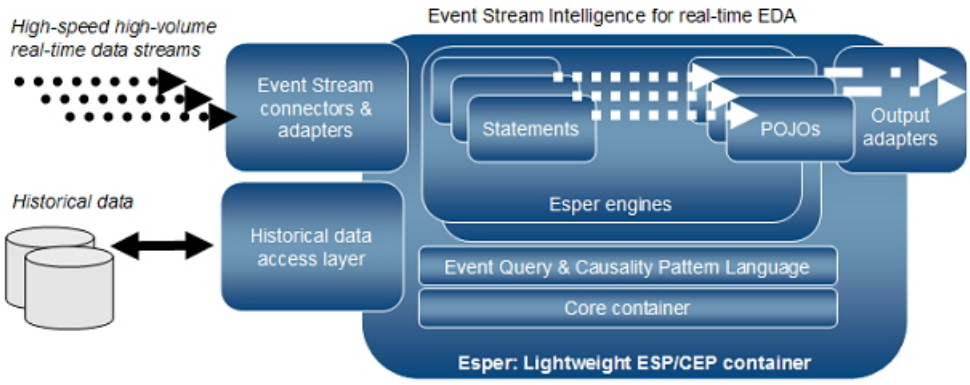

Figura 3.11: Composizione del core Esper

Una volta che le query per l'individuazione di eventi, o pattern di eventi, sono registrate all'interno del core di Esper, gli eventi fluiscono con la loro reale velocità e fanno scattare vincoli logici arbitrariamente definiti all'interno dell'engine sotto forma di Plain Old Java Object[24]. Questo permette di sfruttare qualunque tecnologia Java esistente e, allo stesso tempo, assicurare una facile connessione ai già esistenti building block di tipo SOA [Esp12].

In conclusione, Esper non è un motore stand-alone e non necessità di un application server per essere utilizzato. In sostanza si tratta di una serie di API Java, rilasciate sotto forma di Java ARchive (JAR), in cui la logica di processo deve venir implementata direttamente all'interno del codice dell'applicazione cliente, senza l'ausilio di strumenti esterni per la progettazione. Questo fattore ha l'innegabile vantaggio di rendere Esper estremamente adattabile e low-latency; infatti, gli eventi generati in ingresso e in uscita vengono passati all'engine all'interno dello stesso processo. Di contro, però, lo rendono strettamente dipendente dall'applicazione, in quanto ha bisogno di essere riadattato ogniqualvolta si desidera cambiare il tipo di evento, il numero di stream, la logica delle query, ecc.

La sua logica open source e le infinite possibilità di personalizzazione del prodotto restano comunque i motivi principali per cui è stato scelto come motore di event processing più adatto per la realizzazione di questo progetto di tesi.

[24] Nell'ambito del software computing, POJO è l'acronimo per Plain Old Java Object. Il termine "POJO" viene principalmente usato per descrivere un oggetto Java che non segue nessuno dei principali modelli per oggetti Java, né alcuna convenzione o rappresentazione.

Capitolo 4

Predizione di un indicatore GPI/KPI

In questo Capitolo verrà esplicitato uno degli obiettivi principali della trattazione: anticipare la violazione di un indicatore GPI/KPI durante l'esecuzione di un'applicazione per fare in modo che sia possibile agire su di essa prima che questa violazione venga realmente riscontrata. Così facendo, il sistema potrebbe migliorare la sua efficienza energetica, in quanto la prevenzione del superamento di una soglia di consumo e l'esecuzione di un'azione di riparazione quando ancora il carico di lavoro è considerato accettabile hanno un costo, in termini di risorse impiegate e consumo di potenza, inferiore rispetto alla correzione del sistema una volta riscontrata l'anomalia.

Il concetto di predizione è strettamente legato all'ambito delle serie temporali. Seguirà, dunque, una breve trattazione del significato di serie temporale e di predizione nelle serie temporali, accennando ai principali modelli di rappresentazione di tali serie e ai più comuni metodi di raffinamento della predizione basati sul calcolo di indici di errore nella predizione.

4.1 Come anticipare una violazione?

Il concetto di predizione non è nuovo in ambiti come la statistica, la teoria dei segnali, quella econometrica e la matematica finanziaria. In tutti questi ambiti è importante predire il comportamento di una variabile per poter reagire in maniera migliore alle sue fluttuazioni. Esempi di predizione nella vita reale vengono tutti i giorni dalla meteorologia o dalla finanza. I servizi di informazione che annunciano il tempo del giorno seguente non sono altro che il frutto di una accurata predizione, composta a partire da una serie di dati raccolti nelle ore odierne riguardanti la temperatura, la pressione atmosferica, le correnti, ecc. Per quel che riguarda il mercato finanziario, previsioni sull'andamento dei titoli azionari, o sui valori di apertura e chiusura delle piazze affari internazionali, sono argomenti all'ordine del giorno; anche questi sono il frutto di una serie di operazioni svolte a partire dalle informazioni disponibili agli addetti ai lavori. Maggiori e più dettagliate sono queste informazioni, più precisa sarà la previsione in

oggetto, ma, ovviamente, bisogna saper distinguere, in questo caso, tra informazioni attendibili e informazioni non veritiere.

4.1.1 Il concetto di serie temporale

In generale, per serie si intende la classificazione di diverse osservazioni di un fenomeno rispetto ad un carattere qualitativo. Se tale carattere è il tempo, la serie viene detta storica o temporale. Il fenomeno di interesse, detto variabile, può essere osservato in dati istanti di tempo, prendendo così il nome di variabile di stato, o alla fine di periodi di lunghezza definita, da cui il nome di variabile di flusso. Un esempio di variabile di stato è il numero dei dipendenti di un'azienda o la quotazione di chiusura di un titolo negoziato in borsa; una tipica variabile di flusso, invece, è rappresentata dalle vendite annuali di un'azienda, o dal livello di precipitazioni mensili in un dato luogo [Wik12b].

Indicando con Y il fenomeno, si indica con Y_t un'osservazione al tempo t, con $1 \leq t \leq L$, dove L è il numero complessivo degli intervalli o dei periodi temporali considerati. In generale, una serie storica è definita come $Y = [Y_1, Y_2, ...]$, o più formalmente $Y = [Y_t : 1 \leq t \leq L]$. La serie storica avrà, ovviamente, dimensione L.

Contrariamente a quanto avviene nella statistica classica, dove si suppone che n osservazioni indipendenti provengano da un'unica variabile aleatoria, nelle serie storiche si suppone che esistano n osservazioni provenienti da altrettante variabili aleatorie dipendenti. L'inferenza sulla serie storica si configura, quindi, come un procedimento che tenta di riportare la serie al suo processo generatore.

Queste serie vengono studiate sia per interpretare un fenomeno, così da individuare componenti di trend, ciclicità, stagionalità o accidentalità, ma anche per prevedere il loro andamento futuro.

Ai fini della pura e semplice analisi, esse vengono divise in due categorie: le serie di tipo deterministico e quelle di tipo stocastico. Una serie deterministica si identifica quando i valori della variabile in questione possono essere esattamente determinati sulla base dei valori precedenti; al contrario, in una serie stocastica i valori di tale variabile possono essere determinati sulla base dei valori precedenti solo in misura parziale. La maggior parte delle serie storiche è di tipo stocastico e si rivela, quindi, impossibile elaborare previsioni prive di errore.

L'approccio classico all'analisi delle serie storiche prevede un modello del tipo:

$$Y_t = f(t) + u_t$$

nel quale il valore del fenomeno al tempo t è il risultato della composizione di una sequenza deterministica $f(t)$, detta parte sistematica, e di una sequenza di variabili aleatorie u_t, detta parte stocastica.

Esiste anche un approccio moderno al problema, nel quale si assume che il processo descritto sia stato generato a partire da un processo stocastico descrivibile mediante un modello probabilistico di tipo parametrico. In questa trattazione, tuttavia, verrà utilizzato l'approccio tradizionale precedentemente esplicitato.

E' bene sottolineare che prima di qualsiasi analisi, è necessario esaminare i dati grezzi e apportare alcuni aggiustamenti per depurarli dalle discontinuità, dagli effetti della diversa durata degli intervalli, o periodi di tempo, considerati oppure per eliminare i valori anomali. Alcuni esempi di discontinuità sono i cambiamenti di base nelle serie storiche di numeri indice, oppure la presenza di metriche diverse per le variabili economiche. In tali casi, se non si riesce ad eliminare la discontinuità, può essere preferibile limitare l'analisi a dati omogenei.

La diversa durata dei periodi incide sui valori osservati nei medesimi periodi dei cicli seguenti, producendo variazioni non ascrivibili all'andamento del fenomeno; ad esempio, in serie mensili di dati di produzione, le variazioni nei dati grezzi dipendono solo in parte dal numero dei giorni lavorativi nei diversi mesi. Si possono eliminare tali perturbazioni in vari modi, come ad esempio aggregando i dati in periodi più lunghi o passando a dati medi giornalieri, oppure ancora applicando coefficienti correttivi.

Per quel che riguarda i valori anomali, se ne possono distinguere di due tipi: quelli che provocano una brusca variazione nella serie in un dato istante, dopo il quale però la serie stessa ritorna immediatamente, o gradualmente, all'andamento precedente. Nel primo caso si parla di "outlier additivo", mentre nel secondo di "cambiamento temporaneo". I cambiamenti che introducono una brusca variazione che permane nel tempo, provocando una variazione del livello o dello stesso andamento della serie, costituiscono un "cambiamento strutturale". In Figura 4.1 è possibile osservarne degli esempi [Wik11a].

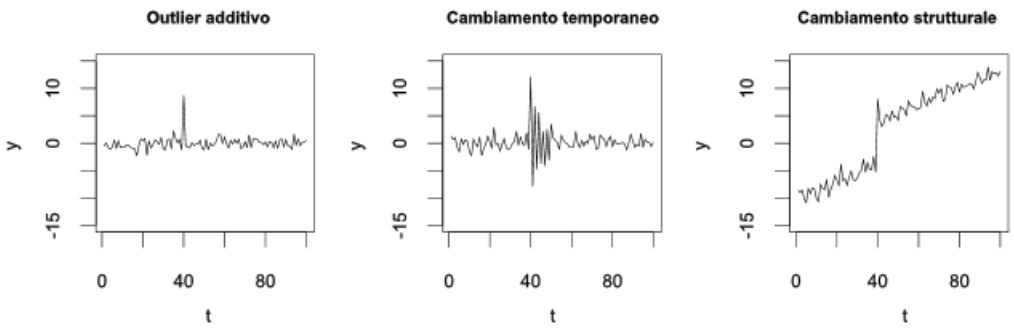

Figura 4.1: Esempi di valori anomali in una serie temporale

Un altro strumento molto utilizzato nell'analisi di serie temporali è il correlogramma. Un correlogramma, o autocorrelogramma, è un grafico che rappresenta la autocorrelazione di una serie storica in funzione del ritardo con cui la autocorrelazione è calcolata. Esso si usa principalmente per individuare l'eventuale prevalenza di una componente tendenziale, stagionale o stocastica [Wik12a].

Come già accennato prima, nell'approccio tradizionale si assume che esista una legge di evoluzione temporale del fenomeno, rappresentata da $f(t)$, e che i residui, vale a dire le differenze tra i valori teorici e quelli osservati, siano dovuti al caso e, pertanto, assimilabili a errori accidentali. I residui vengono normalmente indicati con il termine ε_t ed intesi come variabili aleatorie indipendenti, identicamente distribuite, con media nulla e varianza costante. Nell'approccio moderno, invece, si ipotizza che la parte sistematica manchi o sia già stata eliminata, mediante

stime o altri modelli, e si studia solo la componente stocastica u_t.

Sempre all'interno dell'approccio tradizionale, si ritiene che la parte sistematica sia la risultante di tre componenti non direttamente osservabili:

- *il trend*, o componente tendenziale, inteso come la tendenza di fondo del fenomeno considerato, spesso espressa mediante una funzione polinomiale di grado non troppo elevato;
- *il ciclo*, o componente congiunturale, cioè l'alternanza di fluttuazioni di segno diverso intorno al trend;
- *la stagionalità*, o componente stagionale, costituita da variazioni che si riscontrano con analoga intensità negli stessi periodi di anno in anno, ma con intensità diversa nel corso di uno stesso anno.

La componente accidentale è data dai residui ε_t, spesso indicati con la formula $\varepsilon_t = y_t - \hat{y}_t$, dove con y_t si intende il valore osservato della serie temporale allo tempo t, mentre con $\hat{y}_t$ ci si riferisce al valore stimato nel medesimo istante. In Figura 4.2 sono riportati degli esempi per ciascuna delle componenti presentate [Wik11a].

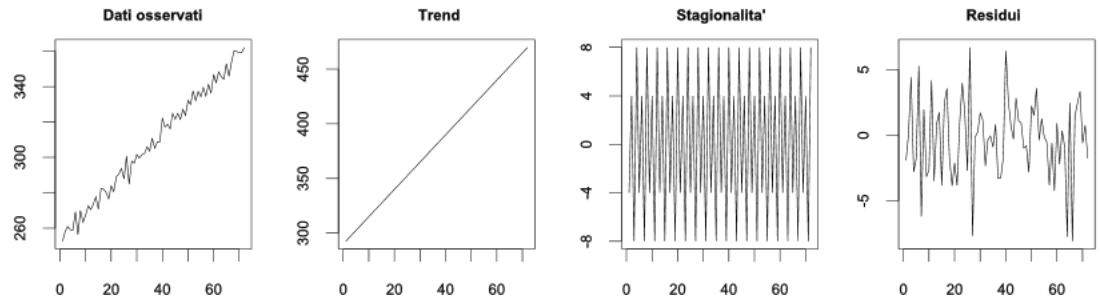

Figura 4.2: Esempio di serie storica e di sua scomposizione nelle componenti tendenziale, stagionale e accidentale

4.1.2 Il concetto di predizione nelle serie temporali

Predire l'andamento di una serie temporale significa utilizzare un modello per anticipare il comportamento futuro della variabile osservata, basandosi sui suoi valori passati.

I modelli per le serie temporali possono essere di vario genere e rappresentare diversi processi stocastici. Quando bisogna modellare variazioni a livello di processo, le tre classi a cui si fa riferimento sono i modelli autoregressivi (AR), i modelli integrati (I) e i modelli a media mobile (MA). Tutti questi sono modelli di dipendenza lineare diretta rispetto ai valori precedenti della serie temporale. Combinazioni di tali modelli possono portare a modelli autoregressivi a media mobile ($ARMA$) o modelli autoregressivi integrati a media mobile ($ARIMA$).

Tutti i modelli fino ad ora introdotti trattano con serie di tipo univariato, ovvero dove è presente una sola variabile da osservare. E' possibile estendere tali modelli per serie temporali multivariate, vale a dire dipendenti da più variabili, ma essi non si ritengono necessari ai fini della trattazione e perciò sarebbe inadeguato dilungarsi.

Talvolta si può richiedere che una serie temporale venga forzata all'interno di un modello: per questo motivo si parla di modelli esogeni. Essi sono caratterizzati dall'aggiunta del carattere "X" ai modelli prima introdotti. Anche in questo caso, però, tali modelli non sono strettamente correlati al progetto di tesi in questione e quindi si preferisce soprassedere.

Tornando ai modelli prima elencati, si definisce modello lineare autoregressivo di ordine p, indicato con $AR(p)$, la seguente relazione:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \phi_p y_{t-p} + \eta_t$$

in cui i parametri $\phi_1, \phi_2, ..., \phi_p$ costituiscono i coefficienti della regressione lineare della variabile casuale y_t rispetto ai suoi stessi valori passati; η_t è il processo bianco[25] raffigurante il termine di errore.

Il modello a media mobile, o MA, da moving average, è un modello statistico utilizzato per modellare le serie storiche, sulla base della media mobile[26] dei loro termini passati.

In generale il modello viene indicato con la notazione $MA(q)$, che sta ad indicare una media mobile di q termini passati:

$$y_t = \eta_t - \omega_1 \eta_{t-1} - ... - \omega_q \eta_{t-q}$$

in cui i parametri $\omega_1, \omega_2, ..., \omega_q$ sono costanti ed η_t è termine di errore, questa volta presente come regressore nelle q unità temporali considerate.

Dal momento che q è un numero intero, e quindi il processo è composto da un numero finito di termini, questo basta a garantire la stazionarietà dei processi $MA(q)$. La media di un $MA(q)$ è zero poiché questo processo, senza intercetta, si riconduce ad una combinazione lineare di variabili casuali di tipo "white noise" con media pari a zero.

Il modello autoregressivo a media mobile, detto anche $ARMA$, è un tipo di modello matematico lineare che fornisce istante per istante un output basandosi sui precedenti valori in entrata e in uscita. A volte, viene denominato modello di Box-Jenkins, dal nome dei suoi inventori George Box e Gwilym Jenkins.

Data una serie storica di valori Y_t, il modello $ARMA$ è uno strumento per analizzare e predire i valori futuri di tale serie e consiste di due parti, ossia una parte autoregressiva (AR) e di una parte di media mobile (MA). Il modello è solitamente indicato con $ARMA(p, q)$ dove p è l'ordine della parte autoregressiva e q è l'ordine della parte media mobile.

Chiamando $u(t)$ la funzione che descrive i valori in entrata in funzione del tempo e $y(t)$ la funzione che rappresenta l'uscita, e sapendo che il modello $ARMA$ ha un'uscita che è una combinazione lineare dei precedenti valori dell'entrata e dell'uscita, $y(t)$ si calcola come:

$$y(t) + \alpha_1 y(t-1) + ... + \alpha_p y(t-p) = \beta_0 u(t) + \beta_1 u(t-1) + ... + \beta_q u(t-q)$$

[25] Il rumore bianco è un particolare tipo di rumore caratterizzato dall'assenza di periodicità nel tempo e da ampiezza costante su tutto lo spettro di frequenze. Nell'ambito dell'analisi e predizione delle serie temporali il rumore bianco indica un processo che si dispone come una normale a media nulla e varianza uguale a 1. La notazione più comune con il quale viene indicato è $WN \sim N(0, 1)$.

[26] In statistica la media mobile è un tipo di filtro a risposta finita all'impulso, anche detto Finite Impulse Response, usato per analizzare un insieme di valori attraverso la creazione di una serie formata dai valori medi di diversi sottoinsiemi dell'insieme di partenza.

Si nota come tale formula risulta essere la somma di un termine autoregressivo AR, costituito dalla parte con i coefficienti α, e una parte di media mobile MA, data dai termini con i coefficienti β.

Ricorrendo alla trasformata discreta di Fourier[27], è possibile introdurre un operatore ritardo z (in genere si scrive s per i sistemi continui) che ha lo scopo di ritardare o anticipare un valore, ossia $f(t) = z \cdot f(t-1)$. Con esso è, dunque, possibile riscrivere il modello come:

$$y(t) + z^{-1}\alpha_1 y(t) + ... + z^{-p}\alpha_p y(t) = \beta_0 u(t) + z^{-1}\beta_1 u(t) + ... + z^{-q}\beta_q u(t)$$

e raccogliere $y(t)$ e $u(t)$ rispettivamente a primo e secondo membro ottenendo una frazione

$$y(t) = \frac{\beta_0 + z^{-1}\beta_1 + ... + z^{-q}\beta_q}{1 + z^{-1}\alpha_1 + ... + z^{-p}\alpha_p} \cdot u(t)$$

Questa rappresentazione del modello è detta funzione di trasferimento $W(z)$. Al numeratore è possibile trovare tutti i termini relativi ai valori in entrata, che solitamente coincidono con i coefficienti dei valori del rumore bianco nel tempo; al denominatore, invece, si trovano i precedenti valori dell'uscita, o meglio, i valori passati della serie temporale. In genere, la notazione più comune per indicare il polinomio al numeratore è $C(z)$, mentre per il denominatore si usa $A(z)$.

Risulta ancora facile ora vedere il modello $ARMA$ come la combinazione di un modello autoregressivo e di un modello a media mobile, ed è chiaro, quindi, che un modello $ARMA(p, 0)$ è in verità un modello autoregressivo $AR(p)$, mentre un modello $ARMA(0, q)$ non è altro che un modello a media mobile $MA(q)$.

È, inoltre, dimostrabile che un qualunque processo $ARMA$ stazionario può essere espresso in modo equivalente come un modello moving average di tipo $MA(\infty)$. Analiticamente, è sufficiente calcolare ricorsivamente i valori di $y(t-k)$ o $y(t)^{(k-1)}$ con $k > 0$ sostituendoli con i valori dell'entrata. Intuitivamente, basta pensare che l'uscita del modello dipende dai valori precedenti dell'entrata e dell'uscita stessa, ma questi ultimi dipendono ancora una volta dai precedenti valori in entrata e in uscita; procedendo a ritroso l'influenza delle uscite precedenti diventa asintoticamente sempre meno influente su quella attuale.

Il problema della predizione è quello di stimare il valore futuro del processo, per esempio al tempo $t + r$, a partire dalle osservazioni del processo stesso fino al tempo t. Il predittore viene indicato con il simbolo $\hat{y}(t + r \mid t)$ oppure, più semplicemente, con il simbolo $\hat{y}(t + r)$ se è chiaro dal contesto qual è l'istante a cui si riferisce l'ultima osservazione. L'intero r prende il nome di orizzonte predittivo [Bit05].

Per calcolare una predizione è possibile partire sia dalle osservazioni passate che alimentano il processo $ARMA$, sia dalle osservazioni passate del processo, vale a dire dai valori che la serie temporale assume negli istanti di tempo precedenti a quello di predizione. Disponendo solamente

[27] La Trasformata Discreta di Fourier (spesso abbreviata a DFT, Discrete Fourier Transform) è un particolare tipo di trasformata di Fourier, che richiede in ingresso una funzione discreta, ovvero non continua in ogni punto, e i cui valori non nulli abbiano una durata finita. L'operatore di riferimento nella DFT è spesso indicato dalla lettera "z".

di questa seconda categoria di dati, in seguito verrà approfondita solo la procedura per ottenere una predizione a partire dai dati.

Indicando, ancora una volta, con $\eta(t)$ il processo bianco a valor medio nullo che alimenta il modello $ARMA$ avente funzione di trasferimento $W(z)$, si supponga di conoscere $\eta(t), \eta(t-1), \eta(t-2), ...$, cioè "il passato di η". Per calcolare $y(t+r)$ è possibile riscrivere il modello $ARMA$ nell'espressione $MA(\infty)$:

$$y(t+r) = w_0\eta(t+r) + w_1\eta(t+r-1) + ... + w_{r-1}\eta(t+1) + w_r\eta(t) + w_{r+1}\eta(t-1) + ...$$

In questa somma infinita, si individuano i due termini:

$$\alpha(t) = w_0\eta(t+r) + w_1\eta(t+r-1) + ... + w_{r-1}\eta(t+1)$$

$$\beta(t) = w_r\eta(t) + w_{r+1}\eta(t-1) + ...$$

Si fa notare che $\alpha(t)$ dipende dai campioni di η tra $t+1$ e $t+r$, mentre $\beta(t)$ dipende solo da η fino a t. Dal momento che η è un rumore bianco, questo implica che $\alpha(t)$ e $\beta(t)$ sono due variabili casuali con coefficiente di correlazione[28] nullo.

Si può allora concludere che, mentre $\beta(t)$ può essere valutato una volta noto il passato di η, $\alpha(t)$ è del tutto impredicibile da tale passato. Il predittore ottimo, a partire dal processo bianco, è perciò:

$$\hat{y}(t+r \mid t) = \beta(t) = w_r\eta(t) + w_{r+1}\eta(t-1) + ...$$

Il corrispondente errore di predizione è dato da:

$$y(t+r \mid t) - \hat{y}(t+r \mid t) = \alpha(t)$$

Per determinare il predittore in pratica, si osserva che $\beta(t)$ può essere scritto anche in questo modo:

$$\beta(t) = [w_r + w_{r+1}z^{-1} + ...] \cdot \eta(t)$$

Dunque, se si definisce $\hat{W}_r(z) = w_r + w_{r+1}z^{-1} + ...$, il predittore ottimo è dato dal sistema in Figura 4.3.

La funzione di trasferimento $\hat{W}_r(z)$ è facilmente ricavabile dalla $W(z)$. Infatti, si consideri l'espansione di $W(z)$ in potenze negative di z:

$$W(z) = w_0 + w_1z^{-1} + ... + w_{r-1}z^{-r+1} + w_rz^{-r} + w_{r+1}z^{-r-1} + ...$$

[28] L'indice di correlazione di Pearson, anche detto coefficiente di correlazione (di Pearson o di Bravais-Pearson) tra due variabili aleatorie è un coefficiente che esprime la linearità tra la loro covarianza e il prodotto delle rispettive deviazioni standard.

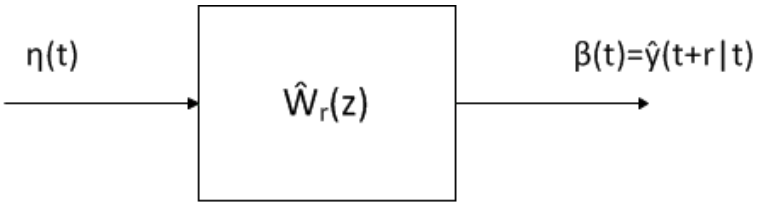

Figura 4.3: Predittore ottimo da $\eta(t)$

E' quindi evidente che

$$W(z) = w_0 + w_1 z^{-1} + ... + w_{r-1} z^{-r+1} + z^{-r} \hat{W}_r(z)$$

Pertanto, la funzione di trasferimento del predittore ottimo a r passi in avanti si ricava tramite la lunga divisione, effettuata per r passi, del numeratore per il denominatore di $W(z)$.

Per affrontare il problema della determinazione del predittore dai veri e propri dati, le osservazioni passate del processo y stesso, bisogna innanzitutto esplicitare il legame tra il passato di η e quello di y. A questo fine, è necessario affrontare preliminarmente il cosiddetto problema della fattorizzazione spettrale.

In molte applicazioni è necessario modellare un processo stocastico stazionario $Y(t)$, di cui si conoscono le caratteristiche, come risposta di un sistema dinamico lineare ad un processo stocastico $U(t)$ in ingresso. Poiché non è possibile determinare in maniera univoca sia il modello che le caratteristiche di $U(t)$, ci si limita a considerare come ingresso un processo standard $E(t)$, cioè un processo stazionario bianco a media nulla.

$E(t)$ è caratterizzato dal solo parametro λ^2, che rappresenta contemporaneamente la densità spettrale[29] e la varianza del processo.

La relazione fra E ed Y è quindi descritta dall'equazione di fattorizzazione spettrale:

$$\Phi_y(z) = W(z)W(z^{-1})\lambda^2$$

dove $W(z)$, funzione di trasferimento del modello lineare, è detto fattore spettrale di $\Phi_y(z)$.

Nell'ambito delle serie temporali, e precisamente nel caso della predizione, la fattorizzazione spettrale si usa per assicurarsi che la rappresentazione $ARMA$ dei dati sia unica.

Secondo il teorema della fattorizzazione spettrale, considerando un processo stazionario $ARMA$, esiste una ed una sola rappresentazione soddisfacente le condizioni di seguito enunciate:

- numeratore e denominatore della funzione di trasferimento sono
  - monici[30]
  - di ugual grado
  - coprimi[31]

[29] La densità spettrale, o spettro di potenza, di un processo $x(t)$ è uguale alla trasformata di Fourier della sua funzione di autocorrelazione $\mathcal{F}\{R_x(\tau)\}$.

[30] Un polinomio si dice monico se il coefficiente della potenza massima è 1.

[31] Due polinomi si dicono coprimi se non hanno fattori in comune, cioè se non sono possibili semplificazioni tra numeratore e denominatore.

- poli e zeri sono interni al cerchio di raggio 1 del piano complesso.

Tali condizioni sono imposte per prevenire eventuali problemi di molteplicità di rappresentazione per la serie temporale.

L'unica funzione di trasferimento definita da questo teorema viene contraddistinta dal simbolo $\hat{W}(z)$ e prende il nome di fattore spettrale canonico (o anche rappresentazione $ARMA$ canonica) [Bit05].

La ragione di questa terminologia è legata al fatto che la formula base con cui si lavora è quella per il calcolo di $\Phi_y(z)$, enunciata poco prima. Al secondo membro di questa espressione dello spettro compaiono due fattori a prodotto, $W(z)$ e $W(z^{-1})$. Trovare una rappresentazione $ARMA$ a partire da $\Phi_y(z)$ equivale dunque a trovare il "fattore" $W(z)$ nell'espressione sopra citata.

Il processo bianco che compare all'ingresso della rappresentazione ARMA canonica è un particolare segnale di tipo "white noise" che talvolta è bene distinguere da un generico processo bianco ricorrendo ad un simbolo opportuno. Per questo, tale processo viene spesso indicato con $\xi(t)$, oppure con $e(t)$, mentre $\eta(t)$ si utilizza per il generico processo bianco.

L'ipotesi base di partenza per il calcolo del predittore ottimo dai dati è che si disponga della rappresentazione spettrale canonica $(\hat{W}(z), \xi(t))$.

Come spiegato prima, il predittore ottimo da ξ si ricava tramite la lunga divisione secondo l'espressione:

$$\hat{W}(z) = w_0 + w_1 z^{-1} + ... + w_{r-1} z^{-r+1} + z^{-r} \hat{W}_r(z)$$

$\hat{W}_r(z)$ è appunto la funzione di trasferimento del predittore ottimo a r passi a partire da ξ. Si ricordi anche che il denominatore di $\hat{W}_r(z)$ coincide con quello di $\hat{W}(z)$. In altre parole, se

$$\hat{W}(z) = \frac{C(z)}{A(z)}$$

allora $\hat{W}_r(z)$ è del tipo:

$$\hat{W}_r(z) = \frac{C_r(z)}{A(z)}$$

D'altra parte il rumore bianco remoto $\xi(t)$ può essere ricavato dai dati grazie ad un filtro sbiancante[32]. Disponendo perciò tale filtro a monte del predittore ottimo da ξ, così come mostrato in Figura , la funzione di trasferimento complessiva è:

$$W_r(z) = \check{W}_r(z)\hat{W}_r(z) = (\hat{W}_r(z))^{-1}\hat{W}_r(z)$$

Ciò significa che il predittore ottimo da y è uguale a:

$$W_r(z) = \frac{A(z)}{C(z)}\frac{C_r(z)}{A(z)} = \frac{C_r(z)}{C(z)}$$

[32] Un filtro sbiancante è un particolare tipo di sistema in grado di ricostruire il processo bianco remoto che è all'ingresso della rappresentazione canonica, a partire dai dati in uscita da quest'ultima.

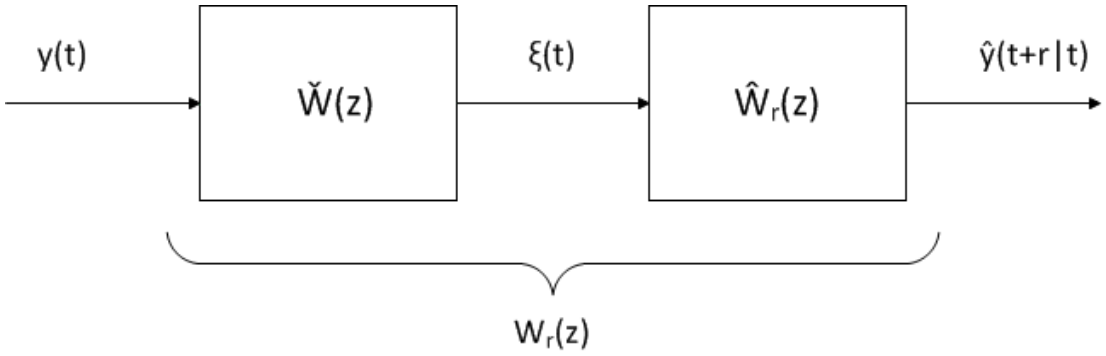

Figura 4.4: Predittore ottimo dei dati

cioè il predittore ottimo dai dati coincide con quello formulato a partire dal processo bianco pur di sostituire al denominatore il numeratore di $\hat{W}(z)$.

Le considerazioni fatte fino ad ora sono di carattere fortemente generale: nello specifico di questo progetto, il calcolo del predittore risulterà essere un'espressione semplificata di quella indicata, sia per il fatto che l'orizzonte predittivo r vale 1, sia per il fatto che il modello scelto per la rappresentazione della serie temporale è un modello $ARMA(p, 0)$, o meglio un modello $AR(p)$.

Eseguendo un solo passo del metodo della lunga divisione, dato che $r = 1$, si ha una formulazione semplificata di $C_r(z)$. Esso, infatti, è sempre uguale a:

$$C_r(z) = z \cdot (C(z) - A(z))$$

Il predittore ottimo ad un passo a partire dai dati è quindi:

$$\hat{y}(t+1 \mid t) = \frac{z \cdot (C(z) - A(z))}{C(z)} \cdot y(t)$$

In un modello autoregressivo, la parte stocastica è rappresentata dal solo termine di errore al tempo t; dunque, il valore del numeratore della funzione di trasferimento, $C(z)$, sarà uguale al coefficiente di tale termine, che solitamente è uguale a 1. Questa caratteristica semplifica ancora notevolmente la formula, rendendola una semplice differenza tra il numeratore e il denominatore della funzione di trasferimento del modello, moltiplicata per l'operatore di ritardo z.

Con $C(z) = 1$ si ha, inoltre, la certezza che la rappresentazione $ARMA$ usata sia una rappresentazione canonica: numeratore e denominatore sono, infatti, sicuramente monici, di ugual grado e coprimi. Gli zeri, banalmente uno solo e uguale a 1, e i poli della funzione di trasferimento ricadono, per costruzione, all'interno del cerchio di raggio 1 del piano complesso.

4.2 Come correggere una possibile violazione?

Qualora il modello di predizione sia sufficientemente adeguato e, quindi, la previsione si possa considerare affidabile oltre una certa soglia di confidenza definita dall'amministratore del sistema, è necessario trattare tale dato come un dato reale effettivamente riscontrato.

Poter anticipare di alcuni istanti di tempo la violazione di un vincolo GPI o KPI è molto importante ai fini del risparmio energetico: un'applicazione, o una infrastruttura, che rispetti propri vincoli ha sicuramente un impatto minore sul consumo di energia rispetto ad una che, invece, oltrepassa i limiti prestabiliti. Allo stesso modo, anche applicare un'azione di riparazione in condizioni di normale funzionamento è meno dispendioso rispetto a farlo quando è in atto una violazione.

Per completare il lavoro iniziato con la predizione, e dunque per prevenire il superamento delle soglie di sicurezza stabilite, verranno applicate le medesime procedure di adattamento o azioni di riparazione progettate per il caso di reale superamento di tali soglie. Per fare un esempio, qualora un'applicazione incida in maniera troppo marcata sulla percentuale di utilizzo della CPU, la sua migrazione su un'altra macchina pochi secondi prima che essa conduca la CPU al superamento della soglia superiore di utilizzo è certamente l'azione più corretta da mettere in atto; con ciò si eviterà di forzare lo spostamento dell'applicazione a vincolo violato.

La prevenzione, se così si può definire, rimane in ogni caso uno strumento di cui non abusare: il modello predittivo, di fatti, dovrà essere progettato per minimizzare il numero di falsi positivi, ovvero di tutti quei casi in cui il modello potrebbe segnalare la violazione di un vincolo GPI o KPI quando in realtà questo non si avvererà negli istanti di tempo seguenti. Maggiore è il numero di falsi positivi, maggiore sarà il numero di azioni di riparazione invocate a vuoto.

L'invocazione di queste azioni è comunque un costo per il sistema, in termini di consumo energetico e di impiego di risorse fisiche e virtuali. Se il calcolo di un GPI o KPI non rispetta i limiti prestabili, il sistema aziona una procedura di adattamento perché è stato progettato per far sì che il costo di tale operazione, da pensarsi sempre nei termini prima esplicitati, sia di gran lunga inferiore rispetto a quello necessario per continuare ad operare con un GPI o KPI al di là della soglia prevista. Migrare un'applicazione da una macchina virtuale all'altra, spegnere un server, o qualunque altra azione invocata da falsi positivi è, invece, uno spreco di risorse critiche per il sistema.

Per questo motivo, spesso nell'analisi di serie temporali l'identificazione del miglior modello di predizione passa attraverso il calcolo di due indicatori, il Final Prediction Error (FPE) e il Akaike's information criterion (AIC). Essi sono due misurazioni statistiche correlate, ma alternative, della bontà di un modello $ARMA(p, q)$.

Generalmente la bontà di un modello si calcola a partire da una qualche funzione della varianza dei residui: più i residui sono piccoli, maggiore è l'adeguatezza del modello. Sia FPE che AIC sono, infatti, effettivamente funzioni dipendenti dalla varianza dei residui.

Un'altra variabile per il calcolo di questi indicatori, però, è espressa dal numero di parametri stimati $n = p + q$. Tale dipendenza è necessaria per permettere al modello di adeguarsi il più possibile alla serie di dati iniziale. Come potrebbe essere chiaro sin da ora, qualora il modello che si ritiene migliore non sia un modello $ARMA$, bensì, quindi, un modello autoregressivo AR o un modello a medie mobili MA, il valore di q nel primo caso e p nel secondo saranno uguali a zero e dunque n corrisponderà solo al numero di parametri del modello rimanente.

Il valore del Final Prediction Error può essere espresso in più modi. Quello risultato più comodo nell'ambito della trattazione è il seguente:

$$FPE_n = \frac{(L + (n+1))}{(L - (n+1))} \cdot \sigma_n^2$$

dove L è il numero di osservazioni della serie temporale e σ_n^2 rappresenta la varianza dei residui del modello con n parametri stimati. Il calcolo di tale valore viene eseguito ricorsivamente con la formula:

$$\sigma_n^2 = \sigma_{n-1}^2(1 - (\hat{\phi}_n \cdot n)^2)$$

con $\hat{\phi_0}^2 = 0$ e $\sigma_0^2 = \frac{1}{L}\sum_{t=1}^{L}(Y_t - \overline{Y})^2$, varianza della serie temporale.

Nei casi in cui si rende necessario un confronto tra più modelli il valore di FPE_n potrebbe non essere quello più indicato per via della sua dipendenza dal numero di parametri stimati. Per questo motivo, a volte si definisce il valore relativo $RFPE_n$. Esso si calcola come il rapporto tra il Final Prediction Error del modello con n parametri stimati e quello del modello esatto, ovvero il modello in cui $n = 0$.

$$RFPE_n = \frac{FPE_n}{FPE_0}$$

Il valore di FPE_0 è uguale al valore dell'autocovarianza della serie temporale con ritardo 0. In altre parole, coincide con la sommatoria pesata di ogni valore della serie temporale moltiplicato per sè stesso, ed è formalmente espressa da:

$$FPE_0 = C_{xx}(0) = \frac{1}{L}\sum_{t=1}^{L-0}\widetilde{Y}(t+0)\widetilde{Y}(t)$$

in cui si ricorda che $\widetilde{Y}(t)$ rappresenta la serie temporale depolarizzata, ovvero resa a media zero [Aka69].

Il miglior modello di predizione sarà dunque quello che avrà il minor valore di FPE_n e n sarà il numero dei suoi parametri stimati. In alternativa, si può considerare il modello con il valore di $RFPE_n$ più vicino a 1.

Qualora risultasse più comodo, tali considerazioni sono valide anche per l'indicatore AIC. Come già detto, esso è direttamente correlato al FPE e si calcola come:

$$AIC_n = log[\sigma_n^2(1 + \frac{2(n+1)}{L})]$$

Nell'ambito di questo progetto di tesi, è risultato più semplice trattare con l'indicatore FPE e saranno le considerazioni fatte su di esso che determineranno la scelta del modello di predizione da utilizzare per calcolare i valori degli indicatori GPI e KPI del sistema, al fine di prevenire il maggior numero di violazioni possibili.

Capitolo 5

Progettazione e realizzazione del tool

In questo Capitolo sarà fatta maggior luce sul tool sviluppato, che ambisce ad unire la caratteristica del controllo del flusso di operazioni di una applicazione alla possibilità di predirne il comportamento e il rispetto di alcuni particolari vincoli, definiti come soglie massime e minime per indicatori GPI e KPI. L'esecuzione di questo software all'interno di un qualunque sistema, logico o fisico, è in grado di aggiungere ad esso caratteristiche di adattività, coniugabili nei più svariati vantaggi possibili, tra cui ad esempio l'aumento dell'efficienza energetica dell'impianto.

Dopo una rapida presentazione dell'architettura generale, verrà fornita una spiegazione dettagliata di ogni fase del programma, con particolare attenzione alle sue interazioni con altri strumenti, quali database, Octave, un programma di calcolo distribuito, e il motore di Complex Event Processing fornito da EsperTech.

5.1 Architettura generale

La Figura 5.1 mostra l'architettura generale del tool. In essa sono presenti quattro diverse fasi di computazione, ognuna delle quali svolge un ruolo fondamentale per il corretto funzionamento dell'applicazione. La fase di Fetching, contenente il modulo Data Retriever si occupa del recupero dei dati di monitoraggio, ottenuti tramite simulazioni e test avvenuti su macchine del progetto GAMES e, successivamente, salvati all'interno di un database chiamato *games_fcim*. Questo database è la fonte da cui si attingeranno i dati per fare predizione.

Il Prediction module, che costituisce la fase di AR modelling, è il metodo incaricato di calcolare i valori futuri degli indicatori GPI e KPI attraverso la costruzione di un modello autoregressivo AR, definito a partire dalle loro rilevazioni passate.

Ogni dato estratto da *games_fcim* e ogni predizione calcolata dal Prediction module corrisponde ad un evento, concreto o possibile, all'interno del sistema. Ciascuno di questi eventi deve poi essere ricondotto ad una struttura comprensibile all'engine CEP per poter essere analizzato.

E' esattamente per questo motivo che è stata implementata una serie di classi Java nelle quali è possibile codificare tutte le informazioni necessarie per il calcolo di ognuno dei indicatori GPI e KPI di interesse, che per semplicità sono stati scelti tra quelli già calcolabili all'interno di GAMES. Nella fase di Event Processing, quindi, ogni evento, relativo alla misurazione di un GPI, o di un KPI, e codificato in un'istanza della classe Java relativa al medesimo indicatore, viene esaminato per ricercare eventuali condizioni di funzionamento anomale.

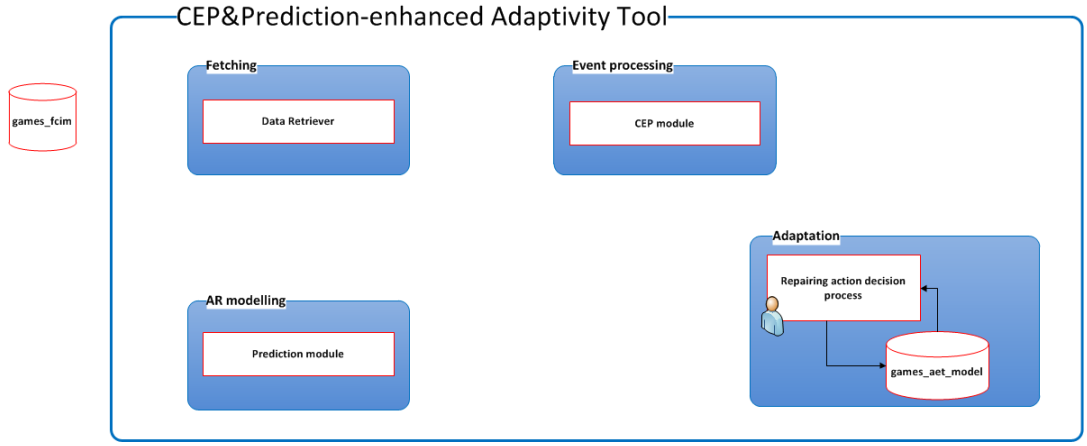

Figura 5.1: Schema generale del tool per la predizione e l'event processing all'interno del progetto GAMES

L'individuazione di una violazione porta il tool ad attivare una procedura di adattamento che ha il compito di fornire all'amministratore di sistema dei suggerimenti utili per l'attivazione della migliore azione di riparazione possibile, tra quelle implementate, per riportare il tutto alla normalità. Questi suggerimenti nascono da informazioni che il codice è in grado di ritrovare nel modello Goal-based, di cui si è parlato nella Sezione 2.2.4. La sua digitalizzazione prende il nome di *games_aet_model*.

Tale framework, a cui spesso ci si riferisce anche con la terminologia "Asset-Event-Treatment Model", per via degli oggetti che lo compongono, è lo strumento a cui si appoggia ogni procedura di adattamento per arrivare a fornire il nome dell'azione di riparazione più indicata. In Figura 5.2 è possibile vedere una sua schematizzazione [CFP+11].

Gli elementi contenuti nell'Asset Layer sono i vincoli sugli indicatori GPI e KPI che il sistema deve rispettare: in esso è, infatti, possibile ritrovare il vincolo sull'indicatore KPI "ResponseTime" o quelli sul GPI "CPUUsage" usati fino ad ora come esempi. Nel livello intermedio sono riportati gli eventi esterni più probabili che potrebbero andare ad influenzare il sistema a runtime; questi eventi sono totalmente al di fuori del controllo del tool e non si può far altro che riportare i loro effetti nel tempo all'interno del database *games_fcim*. Infine, nel Treatment Layer sono descritte tutte le azioni di riparazioni possibili, con le loro relative gerarchie.

Gli oggetti che fanno parte dei vari livelli possono avere vincoli di ereditarietà tra di loro. Come esempio, si guardi l'oggetto dal nome "CPU payload". I due oggetti ad esso collegati, "Reduce" e "Increase" sono specializzazioni dell'oggetto di partenza, in congiunzione AND tra di loro. Questo significa che gli eventi non possono verificarsi simultaneamente, ma solo uno

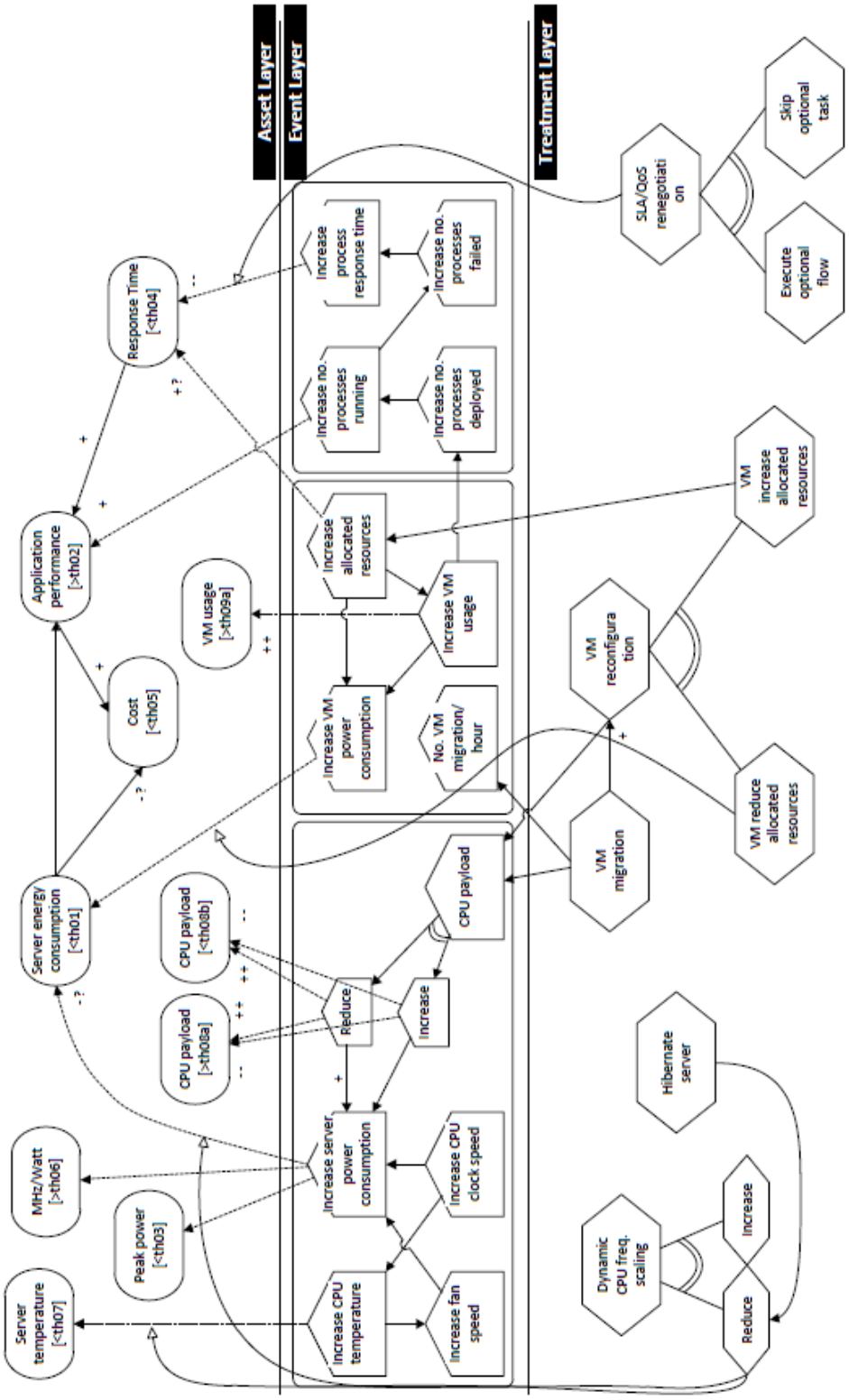

Figura 5.2: Asset-Event-Treatment Model

alla volta, e che il verificarsi di un evento legato al valore del payload della CPU deve per forza ricadere in uno di questi due casi.

Le frecce che collegano gli elementi del grafico hanno significati diversi a seconda di quale sia il loro punto di partenza e quale il loro punto di arrivo: tutte le frecce che partono e finiscono in un oggetto dello stesso tipo vengono detti "contributi", mentre quelle che partono da un oggetto, Event o Treatment che sia, e finiscono in un oggetto diverso da quello di partenza, di tipo Asset o Event, sono definiti "impatti". Quelle frecce che, invece, partono da un Treatment e terminano su una freccia che raffigura un impatto si dicono "attenuazioni".

I contributi e gli impatti portano con loro dei simboli in corrispondenza del loro punto di arrivo. Essi indicano l'effetto positivo o negativo che l'azione corrispondente all'oggetto di partenza ha su quella dell'oggetto finale attraverso un insieme di simboli composto da $\{--, -, +, ++, ?\}$. Ad esempio, l'oggetto che schematizza un evento di aumento della percentuale di utilizzo di una Virtual Machine ha un effetto molto positivo sul rispetto del vincolo GPI "VMUsage [>th09a]", relativo alla soglia di utilizzo minima della VM; al contrario, l'aumento del tempo di risposta di un processo ha un effetto negativo sul rispetto del vincolo KPI "Response Time [<th04]". Il simbolo ? è usato, invece, in tutti quei casi in cui non è possibile determinare a priori l'effetto che l'azione di partenza ha su quella di arrivo.

Ogni contributo o impatto in cui non è specificato alcun effetto, ne sottintende uno negativo.

Le attenuazioni hanno un significato leggermente diverso: esse implicano sempre la riduzione dell'effetto di un particolare impatto. Per spiegarlo tramite un esempio, si guardi all'attenuazione uscente dall'oggetto che modellizza l'azione di riparazione "VM reduce allocated resources": il significato di quella freccia è che l'attivazione di una procedura di adattamento, che sfocia nella scelta di ridurre le risorse allocate di una particolare Virtual Machine, riduce anche l'effetto negativo che un possibile aumento della potenza consumata dalla stessa VM ha sul vincolo GPI Server Energy Consumption.

La comunicazione e lo scambio di dati tra il codice e i database è stata gestita con Hibernate[33], una piattaforma middleware open source che fornisce un servizio di Object-Relational Mapping (ORM). Lo scopo principale di Hibernate è quello di fornire un mapping delle classi Java in tabelle di un database relazionale; sulla base di questo mapping, Hibernate gestisce il salvataggio degli oggetti di tali classi su database. Si occupa, inoltre, del reperimento degli oggetti da database, producendo ed eseguendo automaticamente le query SQL necessarie al recupero delle informazioni e la successiva reistanziazione dell'oggetto precedentemente mappato sul database. Il suo obiettivo è, quindi, quello di esonerare lo sviluppatore dall'intero lavoro relativo alla persistenza dei dati.

Il calcolo predittivo è stato reso possibile dall'integrazione di Octave[34] nel programma. GNU Octave è un linguaggio di interpretazione di alto livello, pensato principalmente per il calcolo numerico. Esso fornisce strumenti per la risoluzione numerica di problemi, lineari e non lineari, e per eseguire altri esperimenti, sempre di carattere numerico. Garantisce anche ampie possibilità grafiche per la visualizzazione e manipolazione dei dati. Octave viene normalmente utilizzato attraverso la sua interfaccia interattiva a linea di comando, ma può anche essere usato per

[33] <http://www.hibernate.org/>
[34] <http://www.gnu.org/software/octave/>

scrivere programmi non interattivi. Il linguaggio Octave è molto simile a MATLAB, in modo che la maggior parte dei programmi siano facilmente trasportabili.

Il motore di Complex Event Processing usato all'interno del tool è Esper, le cui caratteristiche sono già state discusse nella Sezione 3.3.2.

5.2 Fetching

Come già accennato, la prima operazione svolta dal tool è quella di recupero dei dati di monitoraggio. La Figura 5.3 mostra schematicamente l'interrogazione da parte del modulo Data Retriever del database *games_fcim* e l'eventuale e il conseguente recupero dei dati.

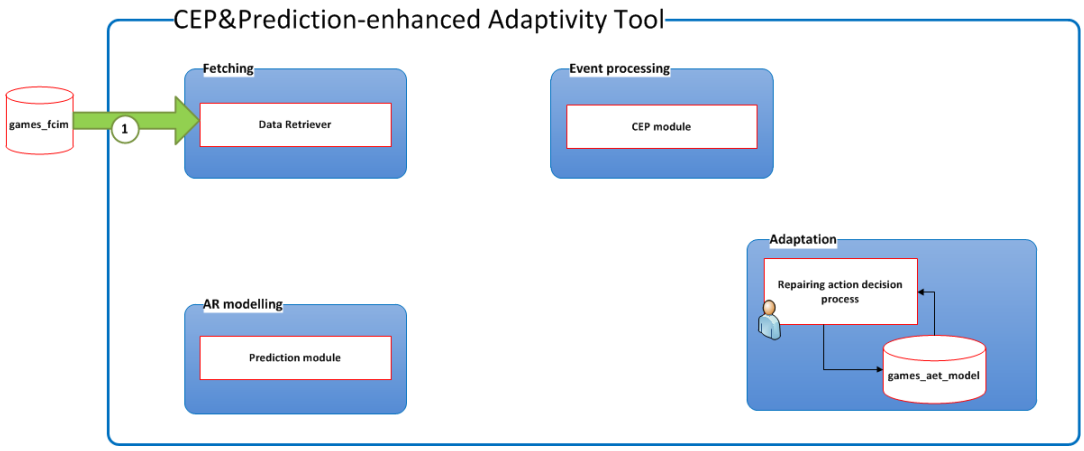

Figura 5.3: Data Retriever

Essa consiste nell'interrogazione costante di una base di dati, precedentemente popolata tramite simulazioni effettuate su macchine fisiche dotate di sensoristica per la campionatura dei valori di sistema, richiamando all'interno dell'applicazione tutti i dati necessari al calcolo dei vari indicatori GPI e KPI.

Il database contenente tutti questi dati di monitoraggio, chiamato *games_fcim*, ha una struttura che è stata definita, a monte di questo lavoro di tesi, nel progetto GAMES. In Figura 5.4 è possibile vedere una sua schematizzazione.

Dalla denominazione di ciascuna tabella è facile capire che i dati interessano la componentistica interna alla macchina fisica, come la CPU, i dispositivi di memoria di massa e la memoria RAM, ma anche le applicazioni, di ogni genere, deployate sulla stessa; per questo motivo, si monitorano anche le Virtual Machine attive e il numero di processi in esecuzione. Oltre ai dati relativi all'hardware e al software sono disponibili anche quelli riguardanti l'ambiente circostante la macchina, come umidità e temperatura dell'aria, e i dati relativi alla rete a cui essa è collegata.

Nonostante la ricchezza di informazioni messe a disposizione da *games_fcim*, talvolta è indispensabile un ulteriore livello di computazione per ottenere ciò a cui si è realmente interessati.

Esistono indicatori GPI e KPI i cui valori sono direttamente estraibili da un unica tabella della base di dati; altri, invece, sono il risultato di un'operazione elementare tra due o più

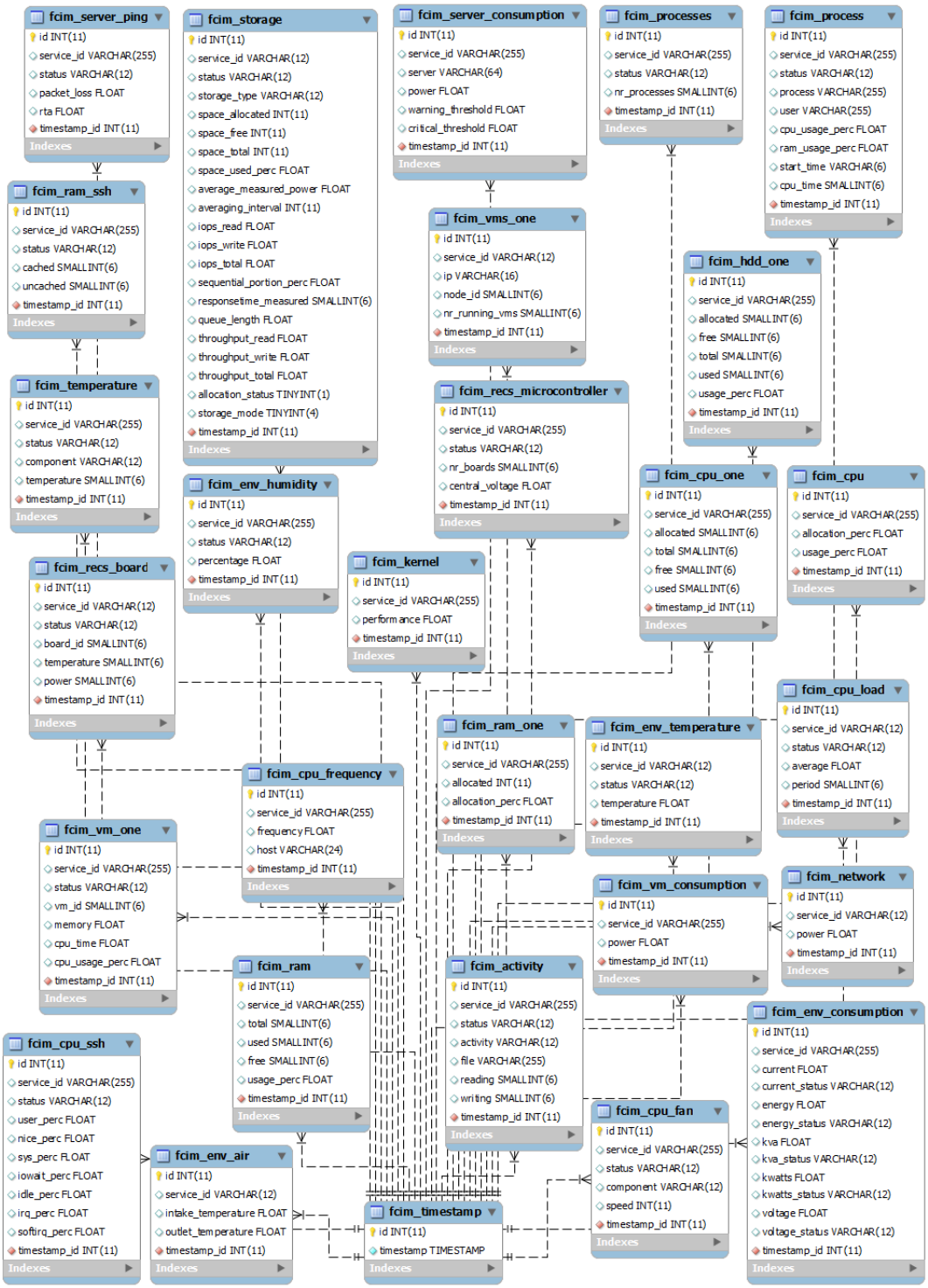

Figura 5.4: Schema del database *games_fcim*

dati. Ad esempio, per calcolare la percentuale di utilizzo della CPU della macchina su cui è in funzione l'applicazione monitorata, un indicatore GPI a cui ci si riferirà d'ora in poi con il termine "CPU Usage", è necessario disporre sia del valore indicante la percentuale di capacità di calcolo della CPU usata dall'applicazione, sia la percentuale allocata per quest'ultima. Il primo dato è ricavabile dal campo *average* della tabella *fcim_cpu_load*, mentre il secondo è uguale a quello riportato nel campo *allocated* della tabella *fcim_cpu_one*. Il rapporto tra questi due valori corrisponde all'indicatore GPI in questione. Al contrario, per calcolare il tempo di risposta, o meglio "Response Time" relativo all'accesso di un dato salvato su disco da parte di un'applicazione è sufficiente estrarre il valore del campo *responsetime_measured* dalla tabella *fcim_storage*.

La tabella 5.1 mostra alcuni esempi di indicatori GPI e KPI, così come sono definiti e calcolati all'interno del progetto GAMES [CFP+11]. Dato il loro carattere generale e l'importanza dei dati a cui sono associati, si è deciso di usare molti di questi come parametri del corretto funzionamento del sistema sul quale andrà ad agire lo strumento software implementato.

Categoria	**Indicatore**	**Threshold**	**Formula**
Server node	Server temperature	< th07	-
	Peak power	<th03	$\max(power)$
	Mhz/Watt	>th06	$frequency/power$
	CPU usage	>th08a	-
		<th08b	
	Energy consumption	<th01	$\int_{time}(power)\,dt$
VM	VM utilization	>th09a	-
		<th09b	
Application container	Application performance	>th02	$TPS/power$
	Response time	<th04	$\sum(task\ response\ time)$
	Cost	<th05	$\sum(task\ cost)$

Tabella 5.1: Esempi di indicatori GPI e KPI

Come calcolare un indicatore GPI o KPI, da quali tabelle ottenere ciò che è necessario per ricavare il suo valore corrente e quante e quali sono le soglie di funzionamento di ognuno di loro sono questioni definite all'interno del progetto GAMES, precisamente nella Energy Practice Knowledge Base (EPKB). E' possibile vedere questa base di dati come un vero e proprio punto di partenza: in essa sono memorizzati i dati relativi alla componentistica hardware dell'intero data centre, dai server fino agli impianti di raffreddamento, la cablatura di rete, i sistemi ausiliari e la sensoristica. Oltre a ciò è specificato anche l'insieme di indicatori GPI e KPI di interesse per lo specifico data centre, con riferimento agli M-file[35] che contengono il codice MATLAB[36] necessario a calcolarli, e ai sensori dai quali è possibile attingere i valori di ingresso necessari a tali file. In un'altra tabella ancora sono indicate le soglie da rispettare unitamente al componente su cui tali valori vanno impostati.

Per motivi di semplicità si è deciso di non collegare il tool, e in particolare i Data Retriever, anche a questo database in quanto le modalità di calcolo degli indicatori sono state definite a

[35] I file che contengono codice MATLAB sono chiamati M-file e hanno estensione .m.

[36] MATLAB (abbreviazione di Matrix Laboratory) è un ambiente per il calcolo numerico e l'analisi statistica che comprende anche l'omonimo linguaggio di programmazione creato dalla MathWorks.

priori e inverosimilmente verranno mutate in futuro. Inoltre, l'interrogazione di una base di dati ampia come l'EPKB usata solo per richiedere le formule per il calcolo di alcuni indicatori è parso subito un inutile spreco di risorse: per ricavare un indicatore GPI o KPI, il più delle volte, basta semplicemente eseguire una divisione tra due valori direttamente ricavabili da *games_fcim*. Per quel che riguarda la definizione della soglie, il basso numero di indicatori permette di settarle manualmente senza difficoltà una ad una.

Come è già stato più volte accennato, tutti i dati importati all'interno del programma non possono essere passati, così come sono, al motore di Complex Event Processing e al modulo per la predizione, in quanto il primo non sarebbe in grado di interpretarli e riscontrare eventuali anomalie, mentre il secondo non riuscirebbe a calcolare un valore futuro attendibile senza altre informazioni a sostegno. Ciò che si rende necessario è, quindi, definire delle strutture dati, in questo caso delle classi Java, in cui salvare, di volta in volta, le informazioni provenienti dal database *games_fcim*. Per ciascuno degli indicatori prima riportati in Tabella 5.1 è stata creata una apposita classe Java, le cui proprietà sono il timestamp a cui fa riferimento il dato, il valore riscontrato, la soglia inferiore o superiore stabilita e, eventualmente lo stato e il nome del componente da cui è stato ricavato.

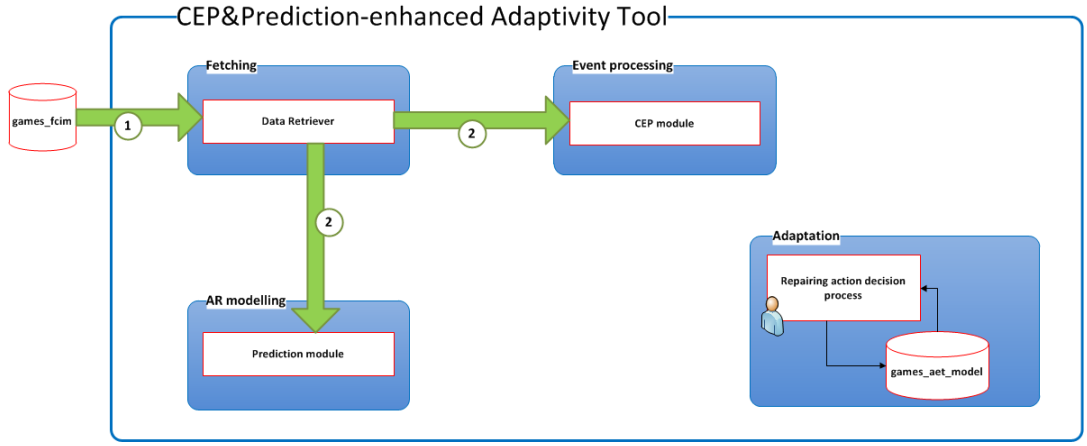

Figura 5.5: Invio dei dati al blocco di predizione e al blocco di Event processing

Ognuna di queste classi Java rappresenta una classe di eventi interpretabile dal motore di Complex Event Processing; il fatto che ne esista una per ogni tipo di indicatore è da intendersi come un modo per semplificare il compito del modulo di predizione, che così non dovrà continuamente differenziare i dati in ingresso per capire a quale GPI o KPI fanno riferimento. Per fare qualche esempio, il valore dell'indicatore KPI "Response Time" viene interpretato all'interno del progetto come un'istanza della classe ResponseTimeMsg.java; allo stesso modo, un'istanza della classe CPUUsageMsg.java conterrà le informazioni relative al GPI "CPU Usage".

Quando tutti i dati relativi ad una specifica misurazione di un indicatore sono stati raccolti e salvati all'interno di una istanza della classe di eventi corrispondente, vengono spediti al blocco di predizione e al blocco di Event processing. Questo invio multiplo, rappresentato in Figura 5.5, risponde all'esigenza di poter riscontrare sia una violazione in atto, sia una potenziale. Infatti,

nel caso in cui si inviassero i dati solamente al blocco di predizione, potrebbe passare troppo tempo prima che il sistema riscontri il superamento di una soglia massima o minima, o, peggio ancora, tale dato potrebbe andar perso per via dell'analisi che viene fatta sui dati in ingresso al blocco predittivo, di cui si forniranno i dettagli in seguito. L'invio dei dati al solo motore di Complex Event Processing, invece, andrebbe banalmente ad annullare la capacità di predizione del tool.

5.3 AR modelling

Come detto nel Capitolo 4, per ottenere una predizione attendibile è necessario disporre di una buona quantità di dati, possibilmente raccolti in modo in modo da formare una serie ordinata nel tempo, a cui ci si riferisce formalmente con il nome di serie storica o serie temporale.

Proprio tale caratteristica è un elemento critico di questo progetto: il tool è stato pensato per mantenere un profilo energetico e di utilizzo di risorse molto ridotto. Il dimensionamento delle strutture di memoria per tenere traccia del passato di ciascun indicatore GPI e KPI è perciò stato soggetto a svariate prove. Il numero di dati all'interno di queste strutture, le quali non sono altro che una forma di rappresentazione logica di una serie temporale, deve essere contenuto, ma abbastanza significativo per produrre una predizione affidabile, vale a dire con un basso errore di predizione. Considerando che gli indicatori GPI e KPI di cui è possibile ricavare i dati di monitoraggio dal database *games_fcim* al momento dell'implementazione di questo tool sono cinque ("Application Performance", "CPU Usage", "Response Time", "Storage Usage" e "IOPS/Watt") e che alcuni di questi vanno osservati su diversi componenti, si è giunti a stimare, empiricamente, che il numero ottimale di dati per serie temporale è di cinquanta misurazioni per componente. In questo modo, il valore di Final Prediction Error (FPE) non è quasi mai superiore al 30% e l'utilizzo di risorse di memoria da parte del tool è minimo.

Un altro problema, sempre indipendente dalla scelta del metodo di predizione, ma strettamente legato ai dati, è l'aderenza ad un modello da parte della serie temporale. Una serie temporale con una componente stocastica fortemente dominante è una serie per cui è molto difficile calcolare una previsione attendibile; invece, una serie in cui la parte sistematica è dominante sarà più facilmente riconducibile ad un modello $ARMA$ e, quindi, predicibile.

Sfortunatamente, le serie temporali ricavate dai dati di *games_fcim* sono serie con una forte componente stocastica ed è per questo che l'errore di predizione minimo si aggira attorno al 30%. Tutto ciò, verosimilmente, si spiega con la difficoltà da parte della sensoristica installata all'interno e all'esterno delle macchine di catturare il dato istantaneo relativo alla sola applicazione che si vuole monitorare e, molto probabilmente, anche la trasmissione di questi dati in un ambiente così fortemente informatizzato potrebbe essere causa di una serie così poco predicibile. Non si può, di fatti, escludere che la sensoristica sia vittima di disturbi causati da altri segnali, i quali non fanno altro che aumentare la componente di rumore del modello della serie.

Ciò nonostante, il blocco di predizione è stato studiato per ridurre al minimo il numero di falsi positivi, intesi come eventi che segnalano la futura violazione di una soglia anche quando questo non si verificherà realmente. E' importante che tale fenomeno sia il più raro possibile

perché ogni volta che viene riscontrata una violazione di un indicatore GPI o KPI si innesca una procedura di adattamento del sistema. Attivare una di queste procedure per giungere a eseguire un'azione di riparazione è un costo per il sistema, in termini di consumo di energia e di utilizzo di risorse. Dalle prove effettuate, su un campione di mille previsioni, confrontando i valori ottenuti con quelli reali si è riscontrato che solo circa il 5% corrisponde a falsi positivi.

Una volta che una predizione è stata calcolata, essa viene incapsulata in una struttura dati comprensibile al motore di Event processing e spedita ad esso. Le strutture dati in questione sono estensioni delle classi Java utilizzate nel blocco di Fetching. Il calcolo di una predizione relativo all'indicatore KPI "Response Time", per esempio, viene convertito in un'istanza della classe ResponseTimePredictionMsg.java, estensione della classe ResponseTimeMsg.java.

I motivi per cui non è possibile usare le stesse strutture già utilizzate dal Data Retriever sono due: prima di tutto, si vuole far presente al motore CEP che questi dati provengono dal blocco di predizione e non da quello di Fetching e, in secondo luogo, perché la previsione porta con sé un'altra proprietà fondamentale da aggiungere a quelle già dichiarate nelle strutture del passo precedente, vale a dire l'attendibilità della previsione, intesa come la probabilità che la previsione calcolata si riveli corretta.

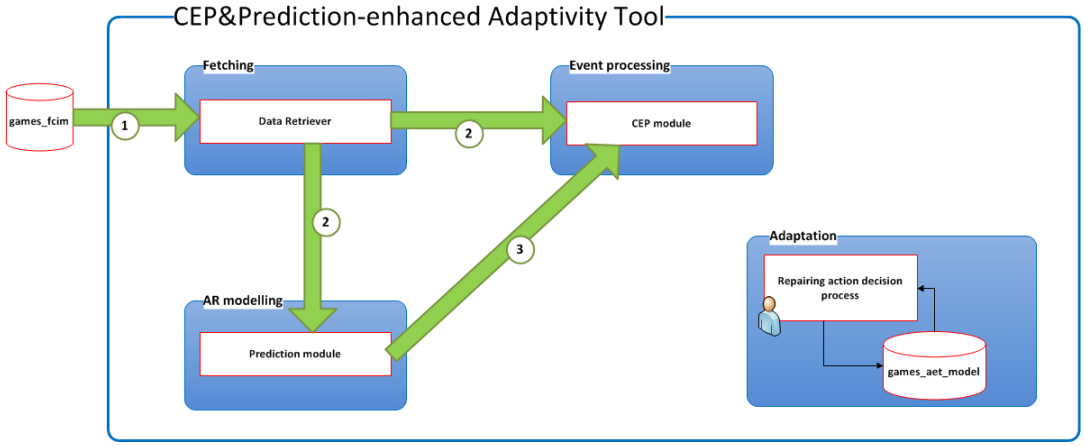

Figura 5.6: Operazione di predizione

Il calcolo della predizione avviene grazie all'integrazione del tool con Octave. Grazie alle API Java fornite dal progetto chiamato javaoctave[37], è possibile chiamare, in maniera rapida e computazionalmente poco onerosa, più istanze del programma di calcolo dal codice del tool e fargli eseguire le operazioni che si desiderano semplicemente caricando al suo interno delle stringhe tramite il comando "*eval*".

Il compito del blocco di predizione è, perciò, quello di comporre due stringhe, una rappresentante la serie di dati storica relativa all'indicatore di cui si vuole calcolare il valore futuro e l'altra relativa ai timestamp nei quali sono stati raccolti i dati. Fatto ciò verrà avviata un'istanza di Octave dal programma, a cui saranno passate le stringhe e il comando per calcolare

[37] http://kenai.com/projects/javaoctave/pages/Home

la predizione. Il risultato, contenente il valore predetto e l'attendibilità della previsione, sarà disponibile poco istanti dopo sotto forma di oggetti riconducibili alla classe Double.java.

A questo punto i dati vengono opportunamente trasformati e inviati al motore di Event processing per essere analizzati. In Figura 5.6, questa operazione è identificata dalla freccia entrante nella fase di Event Processing e uscente da quella di AR modelling.

5.3.1 La funzione *predict*

La vera e propria operazione di predizione è svolta dalla funzione *predict*, personalmente sviluppata per meglio soddisfare i requisiti richiesti. Il suo codice MATLAB, contenuto nel M-file predict.m è riportato in Appendice. Di seguito, tuttavia, è possibile apprezzare il suo pseudo-codice.

Al suo interno sono state utilizzate funzioni contenute in altri listati, alcune disponibili in tutte le installazioni di Octave, altre appartenenti a package opzionali. Esse sono:

- *arburg* - funzione implementata all'interno dell'omonimo M-file, arburg.m, e contenuta nel package che si occupa di Signal Processing. Il suo compito è quello di calcolare i coefficienti del modello AR che si userà per ottenere la predizione. In input a questa funzione va fornito un vettore riga contenenti i dati relativi alla serie temporale di cui si vuole ottenere il modello, il numero di poli di quest'ultimo e il criterio con cui ottenerli.

- *detrend* - accessibile attraverso l'installazione del toolbox TSA, un insieme di funzioni per la Time Series Analysis, questo codice è in grado di rimuovere la componente di trend lineare all'interno di serie non equispaziate e renderle a media zero, passaggio fondamentale per ridurre l'errore macchina durante il calcolo della predizione. Per utilizzare questa funzione è, tuttavia, necessaria una certa dose di accortezza: il nome "detrend" è, infatti, legato ad un'altra funzione propria di Octave. Quello che si richiede, quindi, per essere sicuri di usare la funzione corretta, è di rinominare la funzione del package aggiuntivo.

- *interp1* - questa funzione, presente nel pacchetto generale allegato all'installazione di Octave, permette di rimpiazzare gli outlier della serie temporale con dei valori più vicini alle aspettative tramite un processo di interpolazione, che può essere di vario genere, a seconda della precisione ricercata.

La funzione *predict* è progettata per richiedere in ingresso al più tre parametri: l'unico indispensabile è il vettore riga che riporta i valori della serie temporale nelle precedenti misurazioni, ordinate cronologicamente dalla più vecchia alla più recente.

Il secondo parametro rappresenta, in un altro vettore riga, il valore del timestamp corrispondente alla misurazione collocata nella stessa posizione all'interno del vettore dei valori della serie temporale; in altre parole, all'istante di tempo discreto indicato dal timestamp t_3, per esempio, corrisponde il terzo valore nella serie temporale delle misurazioni dell'indicatore di cui si vuole calcolare una previsione, vale a dire $y(t_3)$. La Figura 5.7 aiuta a comprendere definitivamente questo concetto.

Algoritmo 5.1 Pseudo-codice della funzione *predict*

```
[PredictedValue, ProbabilityOfCorrectnessOfThePrediction] =
        predict(TimeSeries, TimestampSeries, Criterion);

% Mean and variance of the time series
m = mean(TimeSeries);
var = var(TimeSeries);

%Number of observations
L = length(TimeSeries);

% Elimination of the spikes
for i=1:L
        if (TimeSeries(i)>treshold)
                %% Cancel the spike and, if necessary, replace it by
                    interpolation
        endif
endfor

% Depolarization of the time series
[Mean0TimeSeries,Trend]=detrend(TimestampSeries,TimeSeriesWithNoSpikes);

%AutoCovariance calculation

%Calculation of all possible predictions with all possible criterions
%Calculation of the final prediction error and the relative one
for j=1:(L-3)
        %AKIc criterion
        [C(z),A(z)]=arburg(Mean0TimeSeries,j,'AKICc');
        prediction= (z*(C(z)-A(z))/C(z))*TimeSeries

        %KIC criterion
        % Same as above, but this time coefficients of AR model were
            calculated by KIC criterion

        %AICc criterion
        % Same as above, but this time coefficients of AR model were
            calculated by AICc criterion

        %AIC criterion
        % Same as above, but this time coefficients of AR model were
            calculated by AIC criterion

        %FPE criterion
        % Same as above, but this time coefficients of AR model were
            calculated by FPE criterion

        %FPE and RFPE
        FPE0=((L+1)/(L-1))*var;
        %Calculation of FPE is based on both the number of coefficients
            and the chosen criterion
        RFPE=FPE/FPE0
endfor

PredictedValue=prediction(OrderWithWhichTheFPEisMinimum,Criterion);
ProbabilityOfCorrectnessOfThePrediction=abs((1-MinimumFPE)*100);
```

Il dato relativo al timestamp, tuttavia, può essere mancante quando la funzione *predict* viene invocata: ciò si spiega con il fatto che si è voluto lasciare ai futuri utilizzatori la maggiore flessibilità possibile, risolvendo a monte l'eventuale problema della mancanza di questo dato. Quando la serie dei timestamp non viene inserita, ne viene calcolata una automaticamente, con termini equispaziati tra di loro, che ha lunghezza pari a quella che contiene i dati dell'indicatore GPI o KPI. Gli indici di questa serie andranno da 1 fino alla lunghezza della stessa.

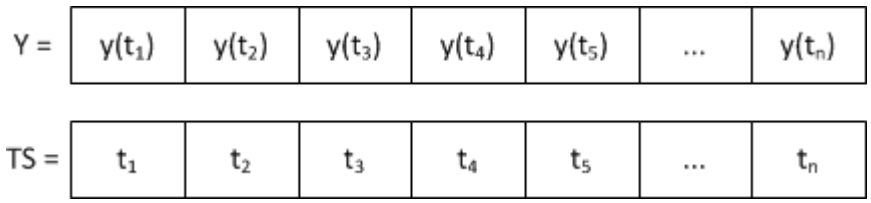

Figura 5.7: Relazione tra la serie temporale e la serie dei timestamp in input alla funzione *predict*

Il terzo e ultimo dato in ingresso rappresenta il metodo con cui si vuole estrarre dai dati in ingresso il modello AR, grazie al quale poi verrà calcolata la predizione. Le scelte possibili sono cinque:

1. "AKICc", per utilizzare il criterio di approssimazione dell'informazione di Kullback[38] corretto;
2. "KIC", per il criterio di approssimazione dell'informazione di Kullback semplice;
3. "AICc", per la creazione di un modello AR che abbia un valore minimo relativamente all'indicatore Akaike's Information Criterion (AIC) corretto;
4. "AIC", per la creazione di un modello che guarda alla minimizzazione dell'indicatore AIC;
5. "FPE", infine, per ottenere un modello che abbia il minimo Final Prediction Error possibile.

AIC, AICc, KIC e AKICc sono basati sulla teoria dell'informazione. Essi cercano di bilanciare la complessità del modello, intesa come numero di parametri stimati, e la sua bontà di approssimazione dei dati reali. KIC e AIC sono stime polarizzate, rispettivamente in maniera simmetrica e non, della divergenza di Kullback-Leibler. AICc e AKICc cercano di correggere tale polarizzazione.

FPE è, invece, un criterio a parte, in quanto esso guarda solamente alla minimizzazione del valore di Final Prediction Error.

Tutti questi sono definiti all'interno di una funzione ausiliaria usata da *predict*, il cui M-file è denominato arburg.m. Nel caso in cui non venisse forzato nessun particolare metodo, si utilizzerà quello che anche la funzione ausiliaria raccomanda, ovvero "AKICc". La scelta di questo metodo è da intendersi come un compromesso tra un modello affidabile, vicino ai dati

[38] Il criterio di approssimazione dell'informazione di Kullback sfrutta il concetto di divergenza di Kullback-Leiber (anche divergenza delle informazioni, guadagno delle informazioni, o entropia relativa), ossia la misura non commutativa della differenza fra due distribuzioni di probabilità P e Q, come metro di paragone per la scelta di un modello di serie temporale. La correzione di tale criterio sta nel non utilizzo dei valori dei poli della funzione di trasferimento della serie temporale, nell'ambito della scelta del modello.

reali, e uno in grado di calcolare una predizione veritiera e, quindi, con un basso valore di Final Prediction Error.

Prima, però, che una qualunque previsione possa essere calcolata, la serie temporale ha bisogno di essere analizzata e trattata per rimuovere trend, periodicità, stagionalità e valori anomali, i cosiddetti "outlier". Tutte queste operazioni preliminari, che compongono la fase di analisi della serie temporale, sono state ovviamente implementate anche all'interno della funzione *predict*.

Per prima cosa vengono calcolate la media e la varianza della serie temporale, due componenti essenziali per le successive fasi di eliminazione dei valori anomali e depolarizzazione della serie.

Come già discusso nella Sezione 4.1.1, è necessario cancellare ogni valore anomalo all'interno della serie per garantire un modello affidabile. Con il termine "valore anomalo" si intende, di solito, un dato in valore assoluto molto distante dagli altri all'interno della serie. Le modalità e i criteri di ricerca di outlier nelle serie temporali sono molti; nell'ambito di questo progetto di tesi, si è deciso di seguire l'approccio proposto da Ma, van Genderen e Beukelman, che etichettano come outlier tutti i valori superiori alla soglia $h = (max + abs\_avg)/2 + K \cdot abs\_dev$, dove max è il massimo valore riscontrabile all'interno della serie, abs_avg è la media dei valori assoluti, abs_dev è la deviazione media assoluta e K è una costante che l'utente può specificare a suo gradimento [Pal09].

Ogni valore che supera la soglia h viene sostituito con la costante matematica "NaN"[39]. Essa viene utilizzata per marcare la posizione degli outlier e rendere più facile il compito della sottofase di interpolazione. La costante "NaN", infatti, non può essere utilizzata per fare calcoli perché qualunque valore moltiplicato o sommato ad essa risulta ancora "NaN"; per questo motivo, è necessario rimuoverla e calcolare un valore per la serie temporale che non risulti essere un outlier e che allo stesso tempo non alteri le qualità della serie, come media e varianza. Per fare ciò si usa l'operazione di interpolazione interp1, che è in grado di ricavare un valore per la serie temporale e rimpiazzare così il punto di discontinuità lasciato dalla cancellazione dell'outlier.

Nella fase di depolarizzazione, la serie temporale viene detrendizzata e resa a media nulla. Questa operazione è necessaria per ricavare un modello predittivo efficiente e per il calcolo dell'autocovarianza, essenziale per la successiva fase di computazione del Final Prediction Error (FPE). La depolarizzazione è un'operazione imprescindibile per poter essere sicuri che la funzione di trasferimento che si andrà a costruire sia un fattore spettrale canonico, così come è stato definito nella Sezione 4.1.2.

Ora che la serie è stata ripulita da valori anomali e depolarizzata, è finalmente possibile calcolare la predizione. Il primo passo consiste nel ricavare i coefficienti del modello AR che rappresenta la serie temporale. Il modo per ottenerli, come noto, è esplicitato dal terzo parametro in ingresso alla funzione *predict*.

Con tale parametro, definito dall'utente o meno, si computano tutti i modelli possibili, con un minimo di coefficienti che va da due fino alla lunghezza della serie diminuita di tre, e il loro relativo valore di Final Prediction Error. Successivamente si ricercherà l'FPE minimo tra tutti quelli calcolati, così da capire il numero ottimale di coefficienti a cui esso è associato; così

[39] In informatica, NaN è un simbolo per indicare il risultato di un'operazione eseguita su operandi non validi, specialmente in calcoli in virgola mobile. Il suo nome è un acronimo di "Not a Number".

facendo si otterrà la funzione di trasferimento della serie temporale, $W(z)$, esprimibile anche come il rapporto tra la componente stocastica della serie $C(z)$ e quella sistematica $A(z)$.

Come spiegato nella Sezione 4.1.1, la formula semplificata per calcolare la predizione ad un passo per una serie temporale a partire dai dati è:

$$\hat{y}(t+1 \mid t) = \frac{z \cdot (C(z) - A(z))}{C(z)} \cdot y(t)$$

La probabilità che esprime la correttezza della predizione appena calcolata, invece, sarà espressa banalmente come opposto del valore di errore sulla predizione, ovvero:

$$P(\hat{y}(t+1 \mid t)\, \grave{e}\, corretto) = 1 - FPE$$

Con questi dati è finalmente possibile inviare un messaggio di predizione al motore di Complex Event Processing, allo scopo di poter anticipare la violazione di una qualche soglia di consumo per i GPI e KPI in questione.

5.4 Event processing

Nell'engine CEP confluiscono tutti i messaggi scaturiti da eventi che si sono già verificati nel sistema o che sono in procinto di accadere. Lo scopo di questo modulo è duplice: esso, di fatti, ambisce a monitorare la situazione generale del sistema stesso, identificando possibili violazioni in corso d'opera e operando per correggerle, ma, allo stesso tempo, è attivo anche sul fronte predittivo, riservando attenzioni particolari anche ai messaggi provenienti dalla fase di AR modelling, per essere in grado di anticipare e agire preventivamente su un GPI o KPI.

Il software di base usato per fare tutto ciò è Esper, così come anticipato nella Sezione 3.3.2.

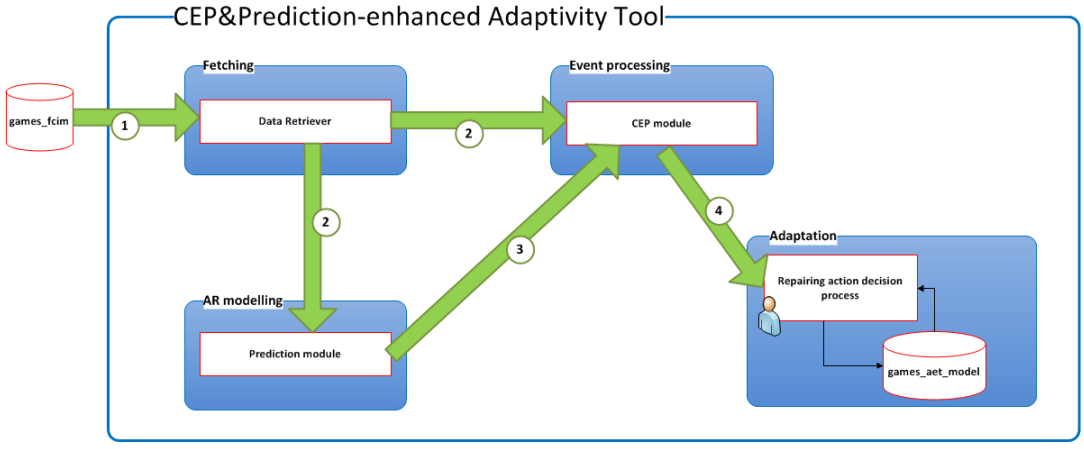

Figura 5.8: Attivazione della procedura di adattamento in seguito all'individuazione di una violazione reale o possibile

Il mezzo con cui l'engine è in grado di rilevare anomalie all'interno del flusso di messaggi in ingresso è l'Event Processing Language (EPL), un linguaggio dichiarativo facente parte della

famiglia dei continuous query language, appositamente sviluppato per gestire eventi ordinati nel tempo e con un alta frequenza di invio. In Figura 5.9 sono riportati alcuni esempi di query in EPL.

```
//Response Time
select * from RT(responseTime > th04)

// CPU Usage inferior threshold prediction
select * from CPUPrediction(probability > 60.0, CPUUsage < th08a)
// CPU Usage superior threshold prediction
select * from CPUPrediction(probability > 60.0, CPUUsage > th08b)
```

Figura 5.9: Esempi di query in EPL

Per ognuno degli indicatori GPI e KPI a cui si è interessati sono state sviluppate due apposite query in EPL, una per catturare violazioni in atto e una per segnalarne possibili future. A queste è stato poi collegato un Listener che verrà innescato ogniqualvolta il CEP incontrerà un messaggio con le caratteristiche riportate nelle query: ad esempio, il Listener per i messaggi relativi all'indicatore KPI "ResponseTime" verrà attivato tutte le volte che un messaggio in ingresso presenta un valore superiore alla soglia massima stabilita di sette secondi, e, se si tratta di un messaggio di predizione, una probabilità di avverarsi superiore al 60%.

Esistono, inoltre, due casi particolari di indicatori ai quali sono stati collegati due Listener. Questi sono gli indicatori GPI "CPUUsage" e "VMUsage". La presenza di un secondo Listener si spiega con il fatto che essi possiedono sia una soglia massima di funzionamento, che una soglia minima; dunque, per alleggerire l'invocazione della procedura di adattamento senza dover far controllare a quest'ultima se si tratta di una violazione della soglia superiore o di quella inferiore, si è deciso di implementare a monte la distinzione creando due diversi Listener, vale a dire uno per ogni possibile violazione.

Ogni Listener, quando viene invocato, richiama al suo interno il metodo update(), il quale non fa altro che accedere all'istanza singleton della classe Java DecisionMaker, con la quale è possibile gestire le violazioni del sistema. Grazie ad essa, sarà possibile ottenere il nome della migliore azione di riparazione possibile per il problema riscontrato, semplicemente fornendogli il nome dell'indicatore GPI o KPI violato e, qualora fosse necessario, il nome del componente su cui è stata riscontrata la violazione.

5.5 Adaptation

Il compito della fase di Adaptation, rappresentata in Figura 5.10, è quello di proporre all'amministratore di sistema la migliore azione di riparazione, tra quelle già implementate e note al tool, per risolvere il problema in atto o imminente.

Va, però, fatto notare che il sistema non attiva una procedura di adattamento immediatamente dopo la ricezione della prima violazione relativa ad uno specifico GPI o KPI. Per scelta implementativa, infatti, si è deciso di procedere nel modo seguente: alla registrazione della pri-

ma violazione, un semplice messaggio segnala all'amministratore che essa è stata riscontrata, mentre per la seconda, invece, si guarda al suo timestamp per decidere cosa fare. Dalla terza violazione in poi, infine, ognuna di queste attiva una procedura di adattamento.

La scelta di non intervenire quando viene riscontrata per la prima volta la violazione di un indicatore si basa sulla convinzione che un singolo episodio isolato di superamento della soglia massima, o minima, di funzionamento consentita possa essere catalogato come un outlier, un valore anomalo che difficilmente, statisticamente parlando, ricapiterà.

Tale considerazione è ritenuta valida anche nel caso di una seconda violazione, purché essa avvenga in un istante molto lontano da quello della prima violazione. Qualora, invece, tale distanza temporale risulti esigua, è molto probabile che tali valori rappresentino una stagionalità all'interno della serie temporale dell'indicatore, una condizione che va assolutamente corretta.

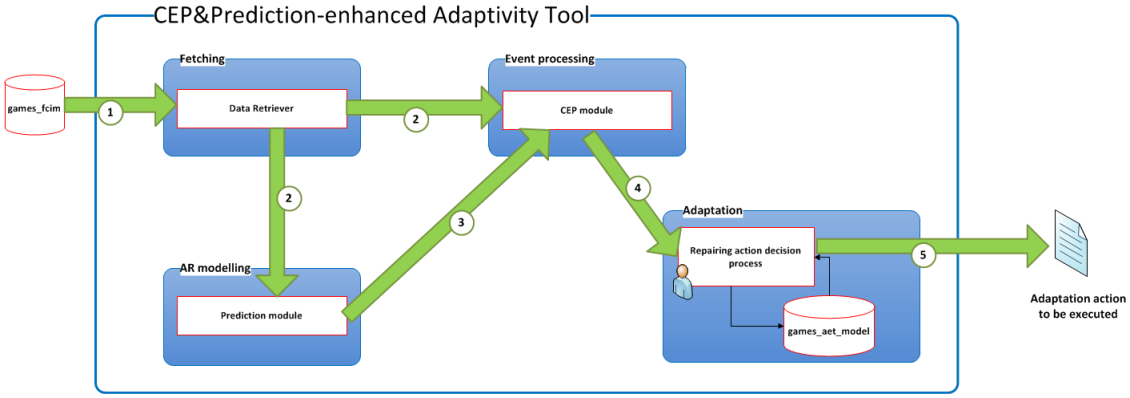

Figura 5.10: Scelta di una azione di riparazione

A differenza di tutti i moduli fino ad ora presentati, la fase adattiva può definirsi semi-automatizzata; infatti, la scelta finale sull'azione di riparazione spetta sempre all'utente. Un approccio di questo tipo è stato pensato per coinvolgere maggiormente l'utilizzatore dell'applicazione, in quanto da esso ci si aspetta collaborazione: la richiesta di un intervento umano in seguito ad ogni violazione riscontrata potrebbe, ad esempio, ridurre al minimo il numero di invocazioni di una azione di riparazione in seguito a falsi positivi.

Per assistere ulteriormente la persona responsabile della scelta, il tool fornisce anche l'elenco di tutte le variabili monitorate intaccate dalla violazione in atto e una lista di tutte le azioni di riparazione in grado di risolvere il problema riscontrato, in cui al primo posto si trova ovviamente quella suggerita. Ad ognuna di esse è assegnato un punteggio, frutto della differenza tra il numero di impatti positivi e negativi che l'attuazione dell'azione di riparazione avrebbe sul sistema. L'azione di riparazione suggerita ha quasi sempre un punteggio superiore o uguale a 0; il che significa che la sua attivazione ha un numero maggiore di effetti positivi sul sistema di quanti siano quelli negativi, oppure, nel peggiore dei casi, non produce eventi critici per il sistema stesso. Questi impatti sono informazioni modellate all'interno del framework Goal-based, di cui si è già discusso nella Sezione 5.1.

Per rendere disponibile l'Asset-Event-Treatment Model al processo che si occupa della scelta dell'azione di riparazione più performante, si è deciso di riportare ogni sua caratteristica al-

l'interno di un database, chiamato *games_aet_model*. In Figura 5.11 è possibile vedere il suo diagramma ER[40].

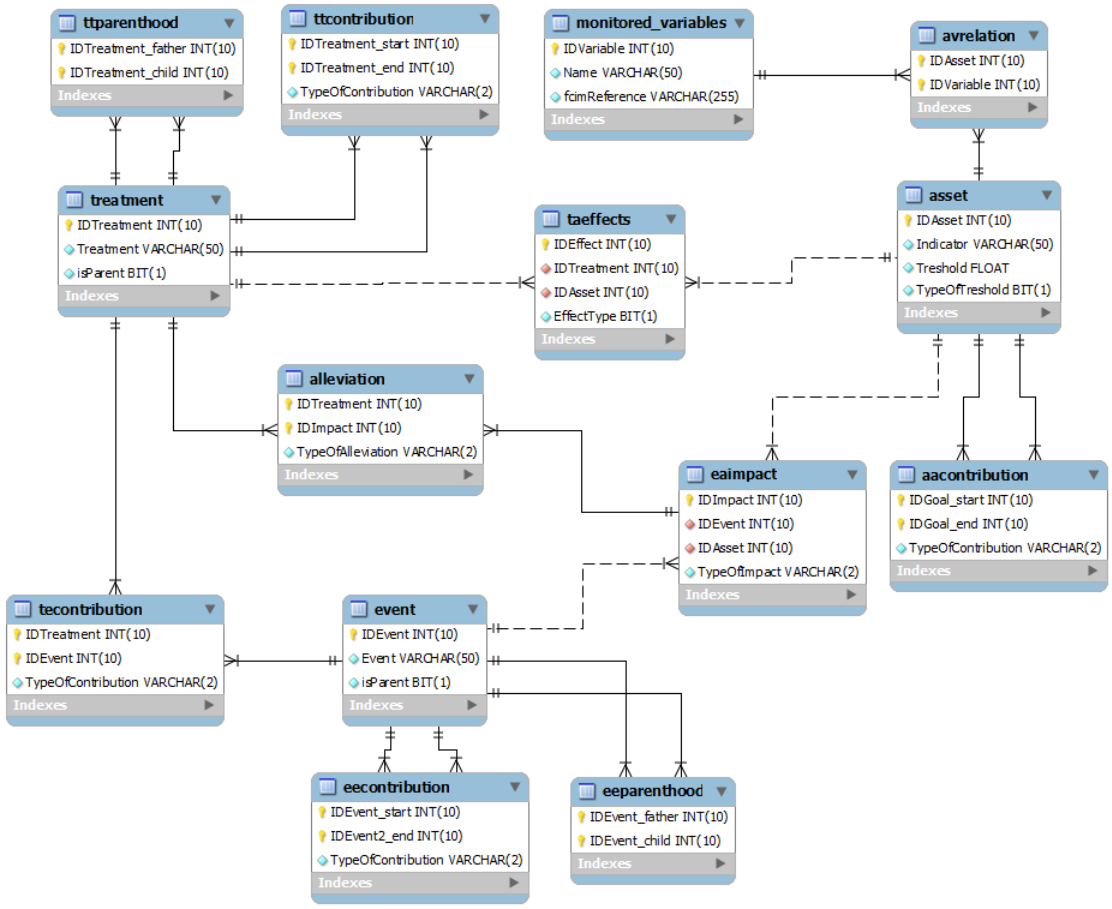

Figura 5.11: Schema del database *games_aet_model*

Ad ogni tipo di oggetto facente parte del modello corrisponde una tabella con lo stesso nome. I contributi, gli impatti e le attenuazioni, sono state divisi, per comodità di implementazione del database, in base al loro punto di partenza e al loro punto di arrivo: in altre parole, il concetto di contributo, per esempio, è modellato dalle tabelle *tecontribution*, che indica il contributo di un oggetto Treatment ad un oggetto Event, *eecontribution*, *aacontribution* e *ttcontribution*, i quali contengono i contributi che si applicano ad oggetti dello stesso tipo, rispettivamente Event nel primo caso, Asset nel secondo e Treatment nel terzo.

La tabella *monitored_variables* esplicita le variabili del sistema monitorate attraverso la sensoristica presente nell'impianto. Esse sono più precisamente le variabili su cui gli eventi esterni agiscono e da cui, allo stesso tempo, si parte per calcolare gli indicatori GPI o KPI da controllare. Il legame tra un Asset e una variabile che lo influenza, direttamente o meno, è esplicitato nella tabella *avrelation*.

[40] In ingegneria del software, un diagramma Entità-Relazione (spesso abbreviato in diagramma ER) è una modalità di rappresentazione astratta e concettuale dei dati.

Il database *games_aet_model* risulta essere, quindi, una copia fedele e arricchita del modello Asset-Event-Treatment.

Il passo seguente per il tool consiste nel verificare che le azioni di riparazione proposte abbiano davvero un impatto positivo sull'indicatore GPI o KPI violato e sul sistema in generale.

Nel prossimo Capitolo verrà fatta maggiore chiarezza su come tutto ciò è stato reso possibile e quali sono stati i risultati ottenuti.

Capitolo 6

Validazione della soluzione adattiva

In questo Capitolo si valuterà la correttezza dei suggerimenti forniti dal tool e del modello sul quale esso di basa, l'Asset-Event-Treatment Model.

Per fare ciò verrà sviluppata una rete locale formata da macchine fisiche e Virtual Machine, sulle quali saranno fatte funzionare delle applicazioni con il solo scopo di raccogliere dati per il calcolo di alcuni indicatori GPI e KPI. Tramite l'adozione di un paradigma di rete client-server, i dati di monitoraggio prima ottenuti verranno inviati al tool e qui analizzati per riscontrare eventuali anomalie. All'atto dell'individuazione di una violazione, concreta o possibile, esso suggerirà all'utente la migliore di azione di riparazione possibile per risolvere il problema riscontrato.

Lo scopo di questa attività di valutazione è quello di cercare di simulare il comportamento dell'azione di riparazione scelta e verificare che, una volta modificato il sistema secondo le sue direttive, esso ritorni in una condizione accettabile in base ai vincoli di funzionamento stabiliti. Solo così facendo, sarà possibile verificare l'efficacia dei suggerimenti del software progettato.

6.1 Introduzione

Durante l'arco della trattazione è stato più volte sottolineato come il tool sviluppato sia in grado di riconoscere una violazione in tempo reale e anticiparne di possibili. Ognuna di queste violazioni attiva un processo di adattamento che culmina con il suggerimento all'amministratore di sistema di una azione di riparazione per far sì che l'impianto possa tornare ad operare in una condizione di "normalità", o, per meglio dire, una condizione riconosciuta come valida da coloro che hanno imposto i vincoli di efficienza produttiva e di consumo, sulle macchine e sulle applicazioni di interesse. Ogni azione di riparazione porta con sè dei cambiamenti logici o strutturali da applicare sull'hardware o il software del sistema.

La correttezza del suggerimento è subordinata a quella del modello Asset-Event-Treatment, che racchiude al suo interno tutte le informazioni relative alle dipendenze tra i vari vincoli posti, agli effetti, positivi o meno, che alcuni eventi esterni potrebbero avere sui vincoli prima citati e all'efficacia delle azioni di riparazione implementate. Quando un evento esterno si rende responsabile dell'aumento o della diminuzione del valore di un indicatore GPI o KPI, ed esso

finisce per violare la propria soglia di funzionamento imposta, è solo grazie al modello Asset-Event-Treatment che è possibile risalire alle variabili di sistema intaccate da questa violazione e, di conseguenza, scegliere l'azione di riparazione più efficace, tra quelle implementate, per ripristinare il tutto.

Fino ad ora, a causa dell'impossibilità di accedere a dati di monitoraggio in tempo reale e della mancanza di un hardware con capacità adattive, l'efficacia ed efficienza dei suggerimenti forniti dal tool e, di conseguenza, del modello da cui essi scaturiscono non hanno potuto avere riscontri su casi concreti.

Per risolvere questo problema è stata pensata una rete locale composta da macchine fisiche e Virtual Machine, sulle quali è presente un'istanza di server GlassFish con deployate al suo interno delle semplici Web Application. Tramite l'invocazione di quest'ultime, si inizierà una serie di test atti a raccogliere il maggior numero possibile di dati necessari per il calcolo degli indicatori GPI e KPI di interesse. Questi dati, una volta trasformati in messaggi capaci di viaggiare nella rete, verranno inviati all'unica macchina su cui è presente e attivo il CEP&Prediction-enhanced Adaptivity Tool per poter essere analizzati.

All'interno del tool, per facilitare l'individuazione di una violazione, verranno definite delle severe soglie di funzionamento, così da innescare agevolmente una procedura di adattamento che culminerà con la segnalazione dell'azione di riparazione che si consiglia di eseguire. A questo punto si cercherà di mettere in pratica le modifiche, logiche o strutturali, invocate dall'azione di riparazione scelta e, successivamente, di rieseguire le stesse operazioni con le quali si sono ottenuti i dati di monitoraggio forniti al tool in precedenza, per arrivare così a comprendere il reale impatto sul sistema della procedura di adattamento appena terminata.

Per ragioni di semplicità di implementazione, si è deciso di raccogliere solamente i dati relativi ad un ristretto numero di indicatori GPI e KPI; si è, in ogni caso, optato per indicatori riportanti informazioni molto diverse tra di loro, specifiche sia dell'applicazione, ma anche dell'intera macchina, virtuale o fisica. Questo ridotto numero di indicatori ha permesso di non dover implementare tutte le azioni di riparazione rappresentate nel modello Asset-Event-Treatment in Figura 5.2: si è optato, infatti, per fornire all'utente solamente le azioni che si è ritenuto significative per la rete locale sviluppata.

6.2 GAMES-BPEL-Simple

Per ottenere un numero di dati di monitoraggio sufficiente a poter svolgere delle attività di test abbastanza approfondite, si è reso essenziale l'utilizzo di un'applicazione di e-commerce definita all'interno del progetto GAMES, poi trasformata in una web application in BPEL.

Le conoscenze associate all'ambito dei servizi e della loro composizione possono essere sfruttate per perseguire l'efficienza energetica, pur garantendo requisiti di QoS tipici delle applicazioni di business. Tali requisiti, infatti, possono includere profili energetici per il software. In aggiunta a tutto ciò, informazioni di contesto possono aiutare nella gestione dinamica delle risorse, che ha, quindi, lo scopo di allocare efficacemente le risorse a disposizione e, possibilmente, controllare le caratteristiche di ciascun software, in modo da sfruttare al meglio le sue possibili capacità

adattive a livello applicativo. Per esempio, ad una applicazione potrebbe essere richiesto di usare solo servizi ad un basso consumo di energia quando è necessario limitare il consumo in potenza di un data centre.

Il cosiddetto "context-aware commerce" è una delle direzioni emergenti di maggior rilievo dell'e-commerce [CPP+10]. I servizi vengono arricchiti da informazioni di contesto riguardanti l'utente finale, come le sue preferenze o il background culturale, per poter anticipare i suoi bisogni più immediati e offrirgli in modo proattivo servizi personalizzati.

A tale proposito, si consideri il servizio di e-commerce per la vendita di libri rappresentato in Figura 6.1. Per acquistare un libro, il cliente naviga attraverso i contenuti del negozio online e seleziona gli elementi che desidera a comprare. Conclusa la scelta, l'utente provvede ad inserire i suoi dati all'interno del sistema. A questo punto, è sia possibile che esso sia già presente all'interno del sistema, nel quale caso le informazioni più dettagliate vengono richieste al database clienti, oppure che si tratti di un nuovo cliente, il che significa che la persona in questione dovrà fornire tutti i dati necessari a creare una nuova voce all'interno del database. Dopo aver ottenuto le informazioni del cliente, è possibile rivedere l'ordine d'acquisto prima di procedere al pagamento. In seguito vi sono due opzioni: la prima coincide con la conferma dell'ordine e il pagamento dello stesso, con conseguente spedizione dei beni, mentre l'altra corrisponde alla cancellazione dell'ordine. Questa seconda possibilità è molto rara e per questo è stata modellata con il solo 1% di possibilità di avversarsi.

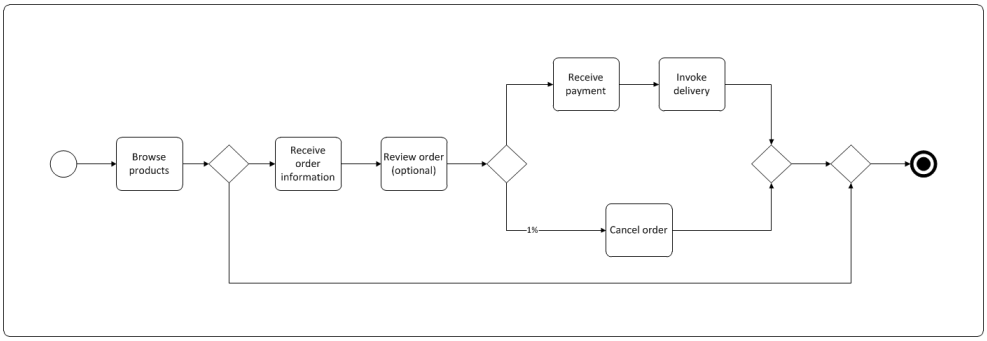

Figura 6.1: Servizio di e-commerce per la vendita di libri

Questo semplice processo di business è un tipico esempio di servizio online e la sua struttura generale può essere rappresentativa di molti diversi scenari, semplicemente cambiando il nome delle varie fasi. Inoltre, esso permette di testare diversi carichi di lavoro che sono ugualmente applicabili ai processi di business, così da ricreare diverse condizioni ed essere in grado di analizzare le caratteristiche di adattività dell'approccio GAMES al problema dell'efficienza energetica.

Da qui si è partiti per sviluppare l'applicazione GAMES-BPEL-Simple. Ogni fase del processo di business prima esposto è stata trasformata in un servizio BPEL che si occupa delle stesse identiche operazioni. Questi servizi vengono invocati da un orchestratore che simula un ordine di acquisto, cercando di concludere il processo in tutte la maniere possibili: nella maggior parte dei casi, quindi, il processo va a buon fine e il cliente acquista i prodotti selezionati all'interno del negozio online, ma, talvolta, è anche possibile che un ordine venga cancellato.

La ripetuta esecuzione nel tempo di questo processo permetterà di raccogliere dati relativi al tempo di risposta di ciascuno dei servizi invocati.

6.3 Configurazione del benchmark

6.3.1 Configurazione della rete locale

Per provare l'efficacia delle capacità adattive proposte dal tool, si è ritenuto indispensabile affiancare un secondo elaboratore a quello usato per la progettazione e implementazione del codice, così da creare una rete locale. Per aumentare la complessità di quest'ultima si è scelto di caricare su ciascuna macchina fisica una Virtual Machine Linux Ubuntu. In Figura 6.2 è possibile osservare una rappresentazione schematica della rete.

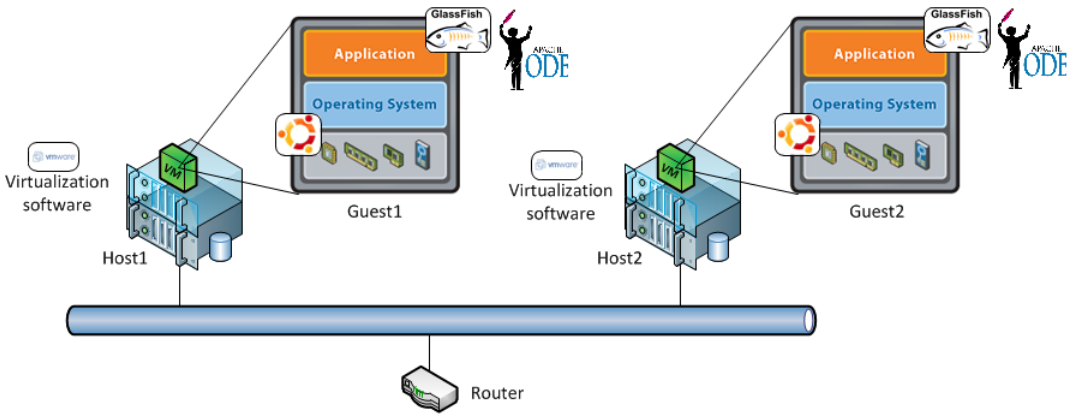

Figura 6.2: Rete locale per la valutazione delle capacità adattive fornite dal tool

La notazione utilizzata per distinguere i vari componenti è tra le più note in letteratura; con il termine "guest", infatti, si indica il sistema operativo della macchina virtuale, isolato da quello della macchina fisica e all'interno del quale viene eseguito. La traduzione italiana di questa terminologia aiuta ancor di più a capire il ruolo di ciascun componente della rete: un sistema "guest" è, banalmente, un sistema operativo ospitato da un altro, il quale, appunto, ricopre il ruolo di oste, da cui la parola inglese "host".

Ogni membro della rete è in grado di comunicare con tutti gli altri grazie alle connessioni virtuali instaurate tra ogni sistema "guest" e il relativo "host", mentre la comunicazione tra macchine fisiche è garantite da una rete wireless domestica protetta. Per questo motivo, ogni macchina fisica avrà due indirizzi di rete e ogni macchina virtuale sarà accessibile ad entrambe le macchine fisiche solo attraverso la rete comune alle due.

Su ogni macchina virtuale è presente un'istallazione di GlassFish[41], un application server open source progettato da Sun Microsystems per la piattaforma Java e ora sponsorizzato dalla Oracle Corporation. Un application server è un software che fornisce l'infrastruttura e le funzionalità di supporto, sviluppo ed esecuzione di applicazioni e componenti server in un con-

[41] <http://glassfish.java.net/>

testo distribuito. Si tratta di un complesso di servizi orientati alla realizzazione di applicazioni multilivello ed enterprise, con alto grado di complessità, spesso orientate per il Web [Wik11b].

Tra le applicazioni deployate al suo interno è possibile trovare Apache ODE[42], uno strumento per l'orchestrazione di processi di business in linguaggio WS-BPEL[43]. In esso è caricato il processo GAMES-BPEL-Simple, di cui si è già parlato nella Sezione 6.2. Ad ognuno dei servizi che compongono il processo è associato un file WSDL[44] che li descrive in termini di operazioni messe a disposizione e vincoli di utilizzo, come il protocollo di comunicazione da utilizzare per accedere al servizio, il formato dei messaggi accettati in input e restituiti in output ed i dati correlati. Oltre a tutto questo, nel file WSDL è contenuto anche il cosiddetto "endpoint" del servizio che solitamente corrisponde all'indirizzo, in formato URI, presso il quale è disponibile il Web Service.

L'esecuzione ripetuta di questo processo è in grado di fornire dati relativi all'indicatore KPI "Response Time", per quel che riguarda il lasso di tempo necessario alla sua invocazione sulla macchina virtuale a partire da quella fisica, e all'indicatore GPI "Server Energy Consumption", grazie ad uno script di monitoring del livello di carica della batteria.

Per poter attivare un processo sulla macchina virtuale, a partire dalla macchina fisica, è necessario impostare correttamente dei parametri di tunneling tramite la procedura di configurazione di rete offerta dal software di virtualizzazione. Il codice dello script di monitoring è molto semplice e non fa altro che estrarre il valore di capacità della batteria dalle informazioni di sistema e stamparlo su file ad intervalli di tempo regolari; successivamente, conoscendo il tempo di campionamento e calcolando la variazione tra due valori contigui, è possibile estrarre il vero e proprio consumo energetico.

Tutti i dati raccolti vengono forniti in ingresso ad una seconda applicazione, sviluppata ad-hoc, per essere trasformati e inviati alla macchina su cui è attivo il CEP&Prediction-enhanced Adaptivity Tool. Questa seconda operazione necessita di rivedere il codice del tool per far sì che offra una interfaccia di comunicazione con la rete: esso, infatti, d'ora in avanti dovrà ricoprire il ruolo di server a cui le altre macchine si connetteranno per inviare i dati di monitoraggio, andando così a ricreare un paradigma di rete client-server.

6.3.2 Implementazione di una interfaccia client-server per il tool

La modalità di comunicazione scelta per lo scambio di messaggi tra le macchine è quella implementata attraverso l'uso dei Java Socket[45]. Il server, o per meglio dire la macchina sulla quale è attivo il tool, apre un canale di comunicazione bidirezionale indicando la porta da cui si aspetta

[42] <http://ode.apache.org/>

[43] WS-BPEL (Web Services Business Process Execution Language) è un linguaggio basato sul XML, costruito per descrivere formalmente i processi commerciali ed industriali in modo da permettere una suddivisione dei compiti tra attori diversi.

[44] Il Web Services Description Language (WSDL) è un linguaggio formale in formato XML utilizzato per la creazione di file per la descrizione di Web Service.

[45] Con il termine "socket" si indica un'astrazione software progettata per poter utilizzare delle API standard e condivise come meccanismo di Inter-Process Communication (IPC). L'accesso ai socket nel linguaggio di programmazione Java avviene tramite l'uso delle classi e delle interfacce presenti nel package *java.net*. Un'applicazione può, quindi, accedere a flussi di dati su reti basate sul protocollo TCP/IP e UDP/IP.

Algoritmo 6.1 monitoring.sh

```
#!/bin/bash
#print start time
echo "Start time: " $(date) >> monitoring.txt;
#calculate initial capacity
echo "Initial value: " $(cat /proc/acpi/battery/BAT1/state |
        grep 'remaining capacity') >> monitoring.txt;
#print sampling time
echo "Sampling time: 5 min" >> monitoring.txt;
while true; do
        #set sampling interval in seconds (usually set to 5 minutes)
        sleep 300;
        #calculate remaining capacity
        echo $(date) "\t" >> monitoring.txt;
        cat /proc/acpi/battery/BAT1/state | grep 'remaining capacity'
                >> monitoring.txt;
done
```

di ricevere dei messaggi; i client, invece, non fanno altro che connettersi all'indirizzo remoto del server tramite la porta specificata.

Una volta che la connessione è stata accettata, può avere inizio l'invio della serie di messaggi lato client e, parallelamente, l'attività di predizione e monitoraggio da parte del server. Per notificare la necessità di intervenire sul sistema con un'azione di riparazione il più presto possibile, il server cerca di installare una connessione verso il client su una porta diversa, adibita esclusivamente alla ricezione di messaggi relativi a Treatment da eseguire sul client. Per semplicità implementativa, si suppone che le informazioni riguardo alla porta verso cui inviare i messaggi, indicanti le azioni di riparazione da eseguire, siano state discusse e condivise in un tempo precedente a quello di computazione.

L'invio dei dati contenenti i valori nel tempo dei vari indicatori GPI e KPI avviene attraverso messaggi JSON[46]. In ognuno di essi è specificato l'ID dell'indicatore a cui la rilevazione si riferisce, il timestamp del momento di tempo continuo in cui il valore dell'indicatore è stato campionato, l'effettivo valore misurato e il componente dal quale è stato misurato, qualora sia un dato dipendente da un oggetto hardware. Un esempio di un possibile messaggio JSON è riportato nel codice in Figura 6.3.

```
{
        "indicator": 6,
        "timestamp": 152,
        "value": 289.6,
        "component": "Server1"
}
```

Figura 6.3: Esempio di messaggio JSON in entrata

[46] Acronimo di JavaScript Object Notation, il JSON è un formato adatto per lo scambio dei dati in applicazioni client-server.

All'atto della ricezione di ciascuno di questi messaggi, il tool inizia una procedura di interrogazione delle basi di dati ad esso collegate per poter tradurre i valori simbolici in ingresso in strutture note. Le informazioni che si va ricercando sono tutte codificate all'interno del database *epkb*. Essa rappresenta una visione di insieme dei dati raccolti nella Energy Practice Knowledge Base (EPKB), un elemento facente parte del progetto GAMES e di cui si è già discusso nella Sezione 2.2.3. L'ID dell'indicatore, passato come valore nel messaggio JSON per distinguerne uno dagli altri, non è altro che l'identificativo numerico progressivo, chiave primaria della tabella *gpi_kpi*, usato per classificare tutti i possibili GPI e KPI attivabili nel progetto GAMES. Analogamente, l'ID del componente sul quale il valore dell'indicatore è stato calcolato equivale alla chiave primaria della tabella *component*. Sebbene in questo progetto di tesi sia risultato superfluo, in un primo momento, collegare il tool ad un database così specifico del progetto GAMES, ora tale comunicazione permette di inviare e ricevere messaggi che non necessitano di rispecchiare la struttura assunta all'interno del codice.

Il reale valore del timestamp è, invece, ricavabile dal database *games_fcim*. Esso, come nei casi precedenti, corrisponde alla chiave primaria della tabella *fcim_timestamp*, un numero intero positivo che funge da rappresentazione semplificata dell'istante di tempo in cui avviene una misurazione.

La misurazione vera e propria dell'indicatore, infine, è semplicemente un numero in virgola mobile, già pronto per la computazione lato server così come viene estratto dal messaggio JSON.

Eseguite queste semplici operazioni di conversione, i dati possono essere inviati correttamente al modulo di predizione e all'engine CEP, nei quali vengono eseguite le stesse identiche operazioni spiegate nelle Sezioni 5.3 e 5.4.

Ogniqualvolta il CEP&Prediction-enhanced Adaptivity Tool rileva una violazione, possibile o effettivamente in atto, invia al client dal quale ha ricevuto i dati anomali un messaggio JSON indicante l'azione di riparazione che l'utente lato server ha scelto, dopo i vari suggerimenti proposti dal tool.

```
{
        "treatment": "Skip optional tasks",
        "component": "Not specified"
}
```

Figura 6.4: Esempio di messaggio JSON in uscita

Come è possibile notare in Figura 6.4, il componente su cui attivare l'azione di riparazione potrebbe non essere specificato: questo succede perché, banalmente, alcune azioni di riparazione non influenzano componenti hardware.

6.3.3 Attuazione di una azione di riparazione

Non potendo disporre di una componentistica hardware con capacità adattive, l'unica soluzione possibile per risolvere il problema della reale attuazione di una azione di riparazione è risultata quella di simulare manualmente le modifiche logiche e strutturali richieste.

Quando un client riceve un messaggio JSON con le indicazioni necessarie per riportare il sistema al rispetto dei vincoli prefissati, esso non fa altro che stampare a video il nome dell'azione, così come essa è codificata all'interno del modello Asset-Event-Treatment, e il componente sul quale va messa in pratica. A questo punto, un utente esperto cercherà di attuare le operazioni richieste dall'azione di riparazione, in sostituzione della procedura automatizzata che non è stato possibile implementare.

Ad esempio, qualora un client violasse il vincolo relativo all'indicatore KPI "Response Time", la procedura di adattamento lato server terminerebbe suggerendo il Treatment "VM migration". Ciò che l'utente lato client potrebbe fare è splittare il carico di lavoro della macchina che ha causato la violazione del vincolo KPI in questione su di un'altra: il modo concreto per fare tutto questo consiste nell'accedere ai file WSDL, descrittivi dei servizi invocati, e fare in modo che alcuni di essi vengano gestiti su macchine diverse.

Da una soluzione di questo genere, per quanto rudimentale, ci si aspetta, comunque, di non riscontrare più la violazione precedente, almeno nel breve periodo. Il tool, e, parallelamente, il modello da cui si estraggono le informazioni necessarie, non sono influenzati dalle caratteristiche del sistema da modificare o dalle modalità di attivazione delle azioni di riparazione; quindi, le difficoltà incontrate fino ad ora non dovrebbero avere alcun peso sulla valutazione della correttezza dei suggerimenti proposti dal codice sviluppato, a patto che, ovviamente, l'esecuzione delle varie azioni di riparazione venga portata a termine con successo.

Una volta che l'utente avrà terminato con le modifiche al sistema, ne sarà testata l'efficacia rieseguendo le stesse attività che hanno portato alla violazione del vincolo di funzionamento con la precedente configurazione.

6.4 Risultati ottenuti

6.4.1 Analisi dei dati pre-adattamento

Partendo da una configurazione di rete che prevede l'invocazione di tutti i servizi del processo GAMES-BPEL-Simple su di un'unica macchia virtuale da parte dell'orchestratore, situato sulla stessa macchina fisica "host" della VM, si è riuscito ad ottenere un discreto quantitativo di dati di monitoraggio per l'indicatore KPI "Response Time" e per il GPI "Server Energy Consumption".

Una parte dell'elaborazione dati compiuta all'interno del CEP&Prediction-enhanced Adaptivity Tool per quel che riguarda l'indicatore "Response Time" è riportata in Figura 6.5.

Come è possibile notare dalla Figura, in blu è rappresentata la serie temporale relativa ai dati realmente misurati, mentre in verde sono riportate le predizioni calcolate. La soglia massima per l'indicatore, contraddistinta dalla linea rossa, è stata fissata a 1.4 secondi.

Una prima analisi visiva del grafico mostra come la serie reale sia caratterizzata da parecchi valori anomali, più frequentemente detti "outlier". Tali valori indicano una predominanza della parte stocastica della serie su quella sistematica. Come già anticipato nella Sezione 4.1.1, una serie di questo tipo è, purtroppo, difficilmente predicibile.

Ciò nonostante, si nota come, all'aumentare dei campioni di riferimento, il predittore fornisca dei valori quasi sempre molto vicini a quelli reali. Per quel che riguarda gli outlier che

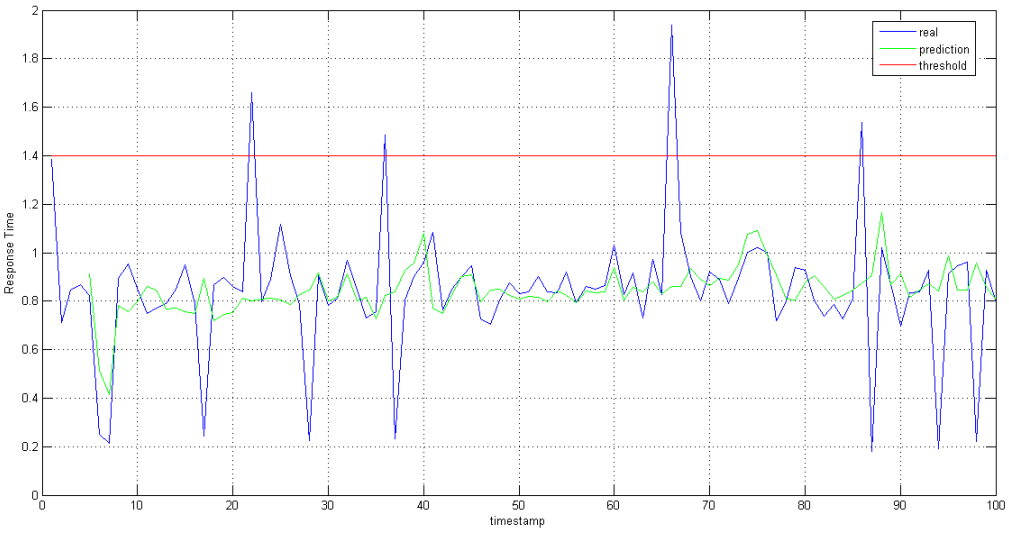

Figura 6.5: Analisi e predizione dei dati di monitoraggio per l'indicatore KPI "Response Time"

corrispondono a violazioni della soglia di funzionamento dell'indicatore KPI, è del tutto normale che non possano essere anticipati: in quanto valori anomali, il modello predittivo non perverrà mai a tali misurazioni.

Quando viene riscontrata una violazione per l'indicatore "Response Time", la procedura di adattamento fornisce le seguenti azioni di riparazione come possibile soluzione al problema riscontrato, basandosi sulle indicazioni che è possibile estrarre dal modello Asset-Event-Treatment:

- "*Skip optional task (Rating: 2)*" - nasce come specializzazione del Treatment "SLA/QoS renegotiation" e consiste nell'individuazione all'interno del sistema di tutti i task che non sono strettamente legati all'attività principale svolta e alla loro conseguente terminazione forzata;
- "*VM migration (Rating: 2)*" - con questa azione di riparazione si vuole migrare parte o tutto il contenuto della macchina virtuale su di un'altra attiva nel sistema;
- "*VM increase allocated resources (Rating: 2)*" - come suggerisce il nome, si intende aumentare le risorse allocate per la macchina virtuale;
- "*Execute optional flow (Rating: 2)*" - essa è la seconda specializzazione del Treatment "SLA/QoS renegotiation" e suggerisce di eseguire un flusso opzionale di operazioni;
- "*Reduce (Rating: 0)*" - l'azione di riparazione "Reduce" è una delle due possibili attuazioni del Treatment "Dynamic CPU frequency scaling" e sottintende, appunto, la riduzione della frequenza di funzionamento della CPU.

Il Rating indica numericamente il concetto, già espresso nella Sezione 5.5, di differenza tra il numero di impatti positivi sul sistema di una specifica azione di riparazione rispetto a quelli negativi, che potrebbero portare ad ulteriori malfunzionamenti.

Per la raccolta dei dati di monitoraggio relativi all'indicatore GPI "Server Energy Consumption", si è scelto di monitorare il consumo energetico di ciascuna delle macchina virtuali attive nella rete tramite lo script di monitoring introdotto nella Sezione 6.3.1.

Come specificato prima, ogni servizio offerto dal processo GAMES-BPEL-Simple è stato eseguito su di un'unica VM; sulla seconda, invece, era attiva solamente l'istanza locale del server GlassFish, simulando così il fatto che essa non fosse impegnata in attività collegate a quella monitorata. La somma dei consumi di entrambe le macchine, una attiva nell'invocazione di servizi e una in uno stato che si potrebbe definire "di riposo", darà luogo alla serie temporale dei valori dell'indicatore "Server Energy Consumption", simulando così la loro provenienza da un'unica macchina fisica, come dovrebbe essere all'interno di un data centre.

A differenza del caso precedente, qui la raccolta di dati è stata più problematica. Il tempo di attesa per ottenere ogni singola misurazione era di cinque minuti, durante i quali doveva essere ovviamente sempre attivo, almeno su una macchina, l'orchestratore dei processi BPEL per invocare i vari servizi. Considerando, inoltre, che lo script è in grado di raccogliere dati consistenti solo durante il funzionamento a batteria degli elaboratori, si capisce come, con una autonomia media delle macchine di circa un'ora e mezza, sia stato difficile ottenere dei dati, posto che durante tutto questo lasso di tempo se ne ottengono solamente diciotto.

Per di più, la serie dei dati ha presentato sin da subito una forte componente di stagionalità. Tutto ciò è normale se si pensa che è stato periodicamente invocato sulle stesse macchine lo stesso script e, approssimativamente, lo stesso numero di servizi per svariate decine di volte. Non c'è da stupirsi, quindi, se i consumi registrati sono stati pressoché identici durante tutto l'arco di tempo delle simulazioni.

Una componente di stagionalità è, tuttavia, un qualcosa che bisogna eliminare dalla serie temporale per poter ottenere un modello predittivo corretto e consistente, così come è già stato specificato nella Sezione 4.1.1. Ciò significa che non è stato possibile avvalersi di porzioni di dati superiori alle venticinque unità, pari a circa due ore di computazione, per il calcolo del modello autoregressivo AR.

Il risultato dell'analisi e del tentativo di predizione da parte del CEP&Prediction-enhanced Adaptivity Tool è riportato in Figura 6.6.

Esattamente come nel caso precedente, in blu è rappresentata la serie dei dati reali, somma dei consumi delle due macchine virtuali VM1 e VM2, mentre in verde sono riportate le predizioni sui valori futuri dell'indicatore e in rosso la soglia massima, scelta pari a 40 Watt.

I suggerimenti del tool per rispondere alla violazione sono i seguenti:

- "*VM migration (Rating: 2)*" - già incontrata nell'ambito della risoluzione di una violazione dell'indicatore "Response Time", essa consiste nella migrazione di parte o tutto il contenuto della macchina virtuale su di un'altra attiva nel sistema;

- "*Hibernate server (Rating: 1)*" - si propone di portare il server dal quale sono stati estratti i dati in uno stato di ibernazione;

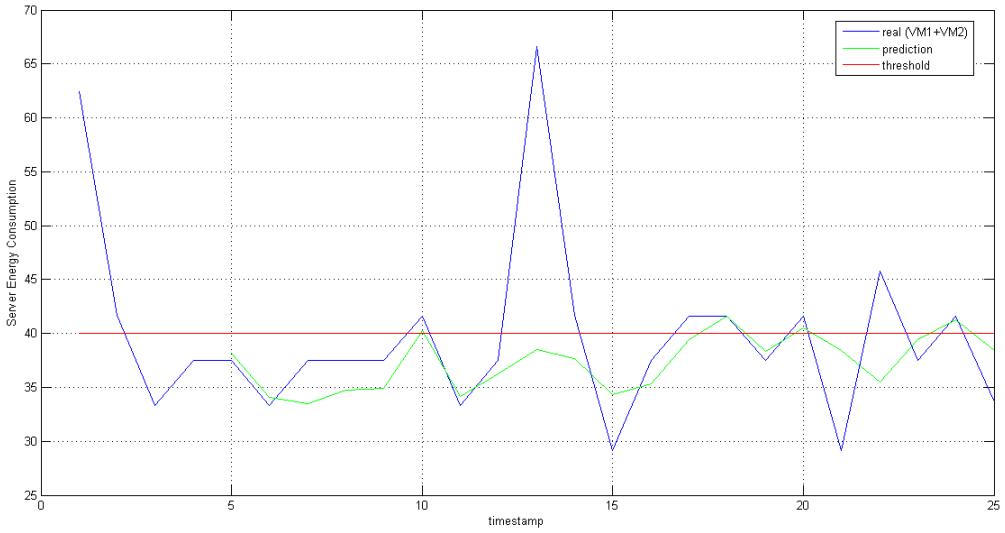

Figura 6.6: Analisi e predizione dei dati di monitoraggio per l'indicatore GPI "Server Energy Consumption"

- "*Increase (Rating: 0)*" - costituisce una specializzazione del Treatment "Dynamic CPU frequency scaling" e sottintende la possibilità di aumentare la frequenza di funzionamento della CPU per far fronte al problema riscontrato;
- "*Reduce (Rating: 0)*" - il modo opposto di attuare il Treatment "Dynamic CPU frequency scaling".

Come già più volte sottolineato, data l'impossibilità di disporre di un sistema con reali capacità adattive, l'azione di riparazione di cui è risultato più facile simulare il comportamento è "VM migration". La scelta è ricaduta, ovviamente, su una delle azioni con il punteggio più alto e si spera che le modifiche apportate con essa corrispondano ad un miglioramento della situazione attuale del sistema. Essa è sembrata, in ogni caso, la scelta più logica possibile, indipendentemente dalle capacità del sistema; "VM migration", infatti, è l'unica che raggiunge un punteggio di Rating uguale a 2 sia nell'ambito della risoluzione della violazione dell'indicatore KPI "Response Time" che in quello dell'indicatore GPI "Server Energy Consumption", il che significa che il modello Asset-Event-Treatment la ritiene, in questo caso, la migliore possibile in termini di efficacia ed efficienza.

6.4.2 Analisi dei dati post-adattamento

Per simulare l'esecuzione del Treatment "VM migration" la configurazione della rete locale è stata cambiata, modificando il file WSDL del servizio relativo alla fase "Browse products" di modo che esso venga invocato da una seconda macchina virtuale, poiché è risultato quello computazionalmente più oneroso. In questo modo tutte le macchine partecipano attivamente alla raccolta dei dati di monitoraggio.

Al fine di evidenziare il più possibile l'effetto dell'azione di riparazione, si è, inoltre, cercato di rieseguire le stesse operazioni del caso di analisi precedente, in modo da sottoporre complessivamente il sistema allo medesimo carico di lavoro nel tempo.

In Figura 6.7 è riportato l'esito dell'analisi e predizione dei dati dell'indicatore KPI "Response Time". Anche questa volta in blu è rappresenta la serie dei dati reali, in verde le varie predizioni calcolate e in rosso la soglia di funzionamento stabilita.

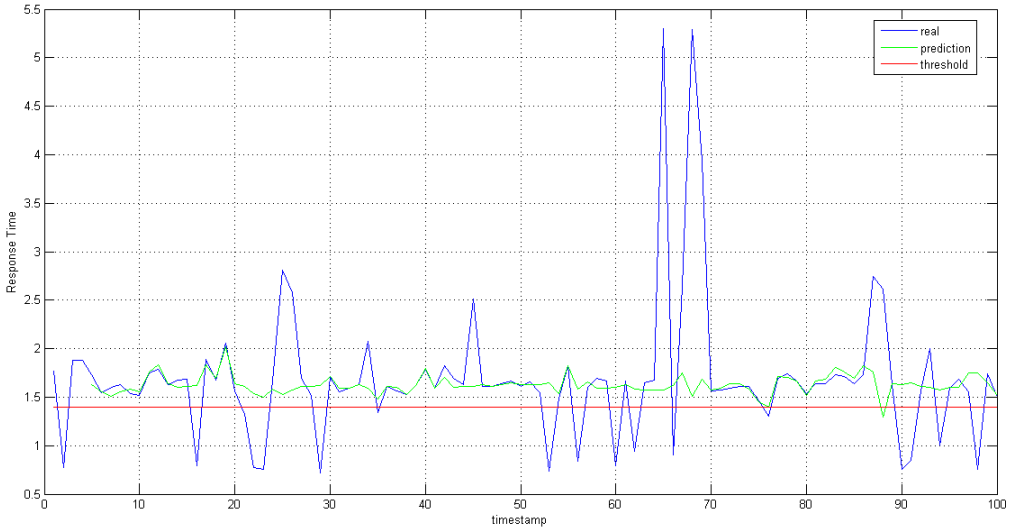

Figura 6.7: Analisi e predizione dei dati di monitoraggio per l'indicatore KPI "Response Time" dopo l'esecuzione di "VM migration"

Fin da subito è facile vedere il notevole peggioramento della situazione: prima dell'esecuzione dell'azione di riparazione il sistema violava il vincolo relativo alla soglia di funzionamento solamente nel caso di un outlier, un comportamento accettabile e che, a causa della sua imprevedibilità, era impossibile correggere.

Ora, l'invocazione di un servizio deployato su una seconda macchina virtuale ha aumentato considerevolmente il tempo di risposta di tutti gli altri. Tale effetto si sarebbe potuto intuire, considerando che il servizio "Browse products" è la prima e imprescindibile delle fasi del processo GAMES-BPEL-Simple. Un ritardo nella sua esecuzione, quindi, produce un ritardo nel tempo di risposta totale che non è eliminabile dalla velocità di computazione della macchina su cui risiede il resto del processo, nonostante essa si sia liberata del servizio più oneroso.

La media del tempo di risposta si è notevolmente alzata, passando da 0.848 a 1.688 secondi, e il predittore ne ha risentito, tanto è vero che segnala quasi sempre, con esattezza o meno, una possibile violazione futura.

Prima di esprimersi riguardo alla correttezza dell'azione di riparazione eseguita, è bene valutare anche il suo effetto sull'altro indicatore campionato durante le varie simulazioni. In Figura 6.8 è, dunque, riportato l'effetto dell'azione di riparazione "VM migration" sui dati e sulle predizioni dell'indicatore GPI "Server Energy Consumption".

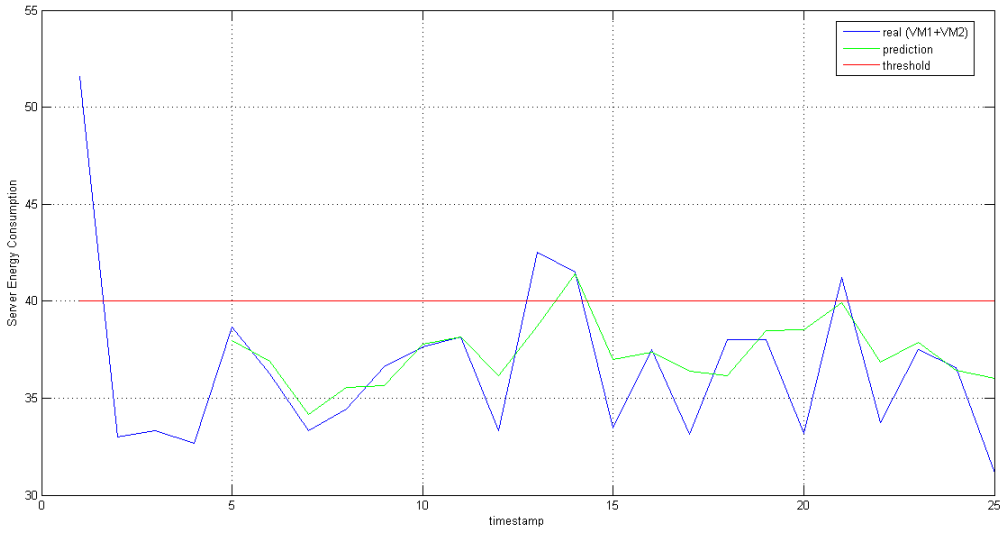

Figura 6.8: Analisi e predizione dei dati di monitoraggio per l'indicatore GPI "Server Energy Consumption" dopo l'esecuzione di "VM migration"

In questo caso, la modifica della rete sembra aver avuto un effetto positivo. Calcolando la potenza totale consumata, in rispetto della formula in Tabella 5.1, come l'approssimazione dell'integrale definito della serie discreta, si nota che il valore passa da 473.71 a 403.9 Wh.

Tralasciando il valore iniziale della serie dei dati reali, che non può essere considerato una vera e propria violazione, quanto invece un costo di avviamento del sistema, per il resto il numero di volte in cui l'indicatore "Server Energy Consumption" supera la propria soglia di funzionamento è inferiore a quello precedente.

Dal punto di vista predittivo, pare che il tool abbia avuto delle difficoltà ad anticipare le violazioni riscontrate. La spiegazione è, ancora una volta, legata alle caratteristiche della serie, la quale, con un andatura così irregolare, non costituisce certo un buon punto di partenza per un modello di predizione.

In conclusione, nonostante il ridotto campione di dati che è stato possibile analizzare per l'indicatore in questione, va riconosciuto l'impatto positivo dell'azione di riparazione.

A questo punto, considerando sia il peggioramento dei valori del tempo di risposta del processo, ma soprattutto l'aumento dell'efficienza energetica grazie al risparmio in potenza che si ottiene sulle macchine virtuali, bisogna riconoscere la correttezza del modello Asset-Event-Treatment, almeno per il caso in oggetto. A fronte di un peggioramento del sistema dal punto d vista prestazionale, si ottiene, comunque, un risparmio energetico non indifferente.

La considerevole variazione dei dati di monitoraggio del "Response Time" è quasi sicuramente imputabile alle ridotte capacità dell'hardware di cui si dispone. All'interno di un data centre attrezzato per sopportare alti carichi di lavoro, si presume che il divario tra le due analisi possa assottigliarsi di parecchio. Per quel che riguarda la "Server Energy Consumption", la diminuzione della media del consumo energetico è il miglior risultato a cui si potesse ambire.

Capitolo 7

Conclusioni

Questo lavoro di tesi ha riguardato la progettazione e realizzazione di uno strumento software a servizi adattivi per l'analisi e la predizione di indicatori fondamentali all'interno di un data centre, allo scopo di lo scopo di diminuire il loro consumo energetico. Accanto ai più classici Key Performance Indicator (KPI), sono stati definiti indicatori di Green Performance Indicator (GPI) all'interno dei quali sono stati mappati valori di sistema critici per il raggiungimento degli obiettivi in termini di efficienza operativa ed energetica.

Come strumento di supporto a questo tool, è stata utilizzata una nuova metodologia di progettazione degli applicativi definita nell'ambito del progetto europeo GAMES (Green Active Management of Energy in IT Service centres). Ad essa è legata la definizione di un modello Goal-based per la risoluzione di eventuali anomalie, quali, ad esempio, il superamento delle soglie di funzionamento pattuite da parte degli indicatori GPI e KPI.

La componente di innovazione della soluzione proposta riguarda la capacità di predizione implementata nel tool, così da poter provare ad anticipare le eventuali violazioni delle soglie massime e minime.

Per realizzare questa specifica funzionalità ci si è basati sulla teoria delle serie temporali e alcune basi di statistica; attraverso queste è stato possibile ottenere dei modelli di predizione per gli indicatori di interesse, semplicemente osservando il loro andamento passato.

Il risultato dell'analisi e della predizione dei dati di monitoraggio è una serie di eventi che vengono poi esaminati da un engine CEP (Complex Event Processing), uno strumento software, nato negli anni Sessanta, che si basa sul meccanismo di *continuous query*.

Ogniqualvolta l'engine CEP incontra un evento che corrisponde ad un comportamento scorretto, sia che esso si stia verificando concretamente oppure che sia frutto di una predizione, avvia una procedura, detta di adattamento, la quale ha il compito di suggerire all'utente incaricato di sorvegliare il sistema la migliore azione di riparazione possibile tra quelle implementate, dove con il termine "azione di riparazione" ci si riferisce ad una modifica strutturale o logica della macchina fisica che ospita l'applicazione da cui è nato il problema che si sta tentando di risolvere. Tali azioni di riparazione sono codificate all'interno del modello Goal-based prima introdotto, e a cui spesso ci si riferisce come modello Asset-Event-Treatment per via di come esso è composto.

Ci si aspetta che l'esecuzione di una qualunque delle azioni di riparazione specificate nel modello introduca un miglioramento a livello prestazionale per il sistema, a patto che, ovviamente, essa sia stata definita come capace di risolvere la violazione riscontrata.

A questo scopo sono state svolte una serie di simulazioni per la raccolta di dati di monitoraggio relativi ad alcuni degli indicatori GPI e KPI più significativi all'interno della rete locale, costruita ad-hoc per l'occasione, in modo da poter verificare gli effetti della scelta di una azione di riparazione sul sistema. Dopo una serie di analisi e predizioni sui valori di "Response Time" e "Server Energy Consumption" calcolati, è stato possibile approvare, anche se solo in parte, la correttezza del modello Asset-Event-Treatment. Nonostante un aumento medio del tempo di risposta dei processi, invocati durante le attività di testing, è stata riscontrata una diminuzione del consumo energetico delle due macchine virtuali tramite l'attivazione delle modifiche logiche proposte dal Treatment "VM migration".

Tra i possibili sviluppi futuri c'è senza dubbio la possibilità di eseguire dei test atti a verificare l'efficienza del modello Asset-Event-Treatment; le simulazioni svolte nel Capitolo 6, infatti, hanno provato solamente la correttezza di quest'ultimo. Per efficienza del modello si intende la verifica della veridicità dei punteggi di Rating, lo strumento attraverso il quale ogni azione di riparazione viene suggerita per risolvere una specifica violazione. Più alto è il valore di Rating, maggiore sarà l'effetto positivo dell'attuazione di quella specifica azione nel sistema. Dall'azione di riparazione con il punteggio più alto tra quelle selezionate per un particolare indicatore GPI o KPI, dunque, ci si aspetta che il miglioramento da essa introdotto sia superiore, in termini prestazionali e di greenness, a quello di qualunque altra.

All'interno di questo lavoro di tesi non è stato possibile testare tale caratteristica per via della limitatezza dell'hardware di cui si disponeva.

Appendice A

M-file sviluppato per il calcolo predittivo

predict.m

```
function [pv,prob] = predict(y,t,c);
% predict the t+1 value of a time series, given the previous t values.
% [PredictedValue, ProbabilityOfCorrectnessOfThePrediction] = predict(
    TimeSeries);
%
% y     Time series y(t)
% t     Timestamp series (the event y(n) occured in time t(n))
% c     Criterion with which calculate the AR(p) parameters
%
% pv    Predicted value y(t+1|t)
% prob  Probability of correctness of the prediction
%
% This function uses octave m-files contained in the statistical package
    and in the tsa-4.1.1 (Time Series Analysis). The link to download the
    tsa package is:
% http://biosig-consulting.com/matlab/tsa/download.html
% I strongly suggest to download both tsa and NaN package.

% Mean and variance of the time series
m=mean(y);
var=var(y);

% Number of observations
L=length(y);
if nargin<2
        t=[1:L];
```

```
endif
if nargin<3
        c='AKICc';
endif

% Elimination of the spikes
max_value=max(y);
abs_dev=mad(y);
K=1;
int_ts=[];
yprov=[];
ts=[];
treshold=(max_value+m)/2+K*abs_dev;
for i=1:L
        if (y(i)>treshold)
                int_ts=[int_ts t(i)];
        else
                yprov=[yprov y(i)];
                ts=[ts t(i)];
        endif
endfor
yrefined=y;
if(length(int_ts)>0)
        interp=[];
    for j=1:length(int_ts)
        interp(j)=interp1(ts,yprov,int_ts(j),'nearest');
                % NA case handling
                if(isnan(interp(j)))
                        interp(j)=interp1(ts,yprov,int_ts(j),'nearest','
                            extrap');
        endif
                for h=1:length(t)
                        if(int_ts(j)==t(h))
                                yrefined(h)=interp(j);
                        endif
                endfor
        endfor
endif

% Depolarization of the time series
[ytilde,trend]=detrendts(t,yrefined);

% AutoCovariance calculation
cxx=[];
for l=0:(L-1)
```

```
        temp=0;
        for n=1:(L-l)
                temp=temp+(y(n+l)-m)*(y(n)-m);
        endfor
        cxx(l+1)=(1/L)*temp;
endfor

% Calculation of all possible predictions with all possible criterions
prediction=[];
% Calculation of the final prediction error and the relative one
fpe=[];
sigma=[var];
for z=1:(L-3)
        % AKIc criterion
        [a1,v1,k1]=arburg(ytilde,z,'AKICc');
        cz1=a1(1);
        az1=[1 -a1(2:length(a1))];
        same_length1=false;
        while(same_length1==false)
                cz1=[cz1 0];
                if (length(cz1)==length(az1))
                        same_length1=true;
                endif
        endwhile
        diff1=cz1(2:length(cz1))-az1(2:length(az1));
        pvtilde1=0;
        for k1=1:length(diff1)
                pvtilde1=pvtilde1+diff1(k1)*ytilde(L-k1+1);
        endfor
        prediction(z,1)=pvtilde1+m;

        % KIC criterion
        [a2,v2,k2]=arburg(ytilde,z,'KIC');
        cz2=a2(1);
        az2=[1 -a2(2:length(a2))];
        same_length2=false;
        while(same_length2==false)
                cz2=[cz2 0];
                if (length(cz2)==length(az2))
                        same_length2=true;
                endif
        endwhile
        diff2=cz2(2:length(cz2))-az2(2:length(az2));
        pvtilde2=0;
        for k2=1:length(diff2)
```

```
        pvtilde2=pvtilde2+diff2(k2)*ytilde(L-k2+1);
endfor
prediction(z,2)=pvtilde2+m;

% AICc criterion
[a3,v3,k3]=arburg(ytilde,z,'AICc');
cz3=a3(1);
az3=[1 -a3(2:length(a3))];
same_length3=false;
while(same_length3==false)
        cz3=[cz3 0];
        if (length(cz3)==length(az3))
                same_length3=true;
        endif
endwhile
diff3=cz3(2:length(cz3))-az3(2:length(az3));
pvtilde3=0;
for k3=1:length(diff3)
        pvtilde3=pvtilde3+diff3(k3)*ytilde(L-k3+1);
endfor
prediction(z,3)=pvtilde3+m;

% AIC criterion
[a4,v4,k4]=arburg(ytilde,z,'AIC');
cz4=a4(1);
az4=[1 -a4(2:length(a4))];
same_length4=false;
while(same_length4==false)
        cz4=[cz4 0];
        if (length(cz4)==length(az4))
                same_length4=true;
        endif
endwhile
diff4=cz4(2:length(cz4))-az4(2:length(az4));
pvtilde4=0;
for k4=1:length(diff4)
        pvtilde4=pvtilde4+diff4(k4)*ytilde(L-k4+1);
endfor
prediction(z,4)=pvtilde4+m;

% FPE criterion
[a5,v5,k5]=arburg(ytilde,z,'FPE');
cz5=a5(1);
az5=[1 -a5(2:length(a5))];
same_length5=false;
```

```
while(same_length5==false)
        cz5=[cz5  0];
        if (length(cz5)==length(az5))
                same_length5=true;
        endif
endwhile
diff5=cz5(2:length(cz5))-az5(2:length(az5));
pvtilde5=0;
for k5=1:length(diff5)
        pvtilde5=pvtilde5+diff5(k5)*ytilde(L-k5+1);
endfor
prediction(z,5)=pvtilde5+m;

% FPE and RFPE
FPE0=((L+1)/(L-1))*sigma(1);
if (strcmp(c,'AKICc')==1)
        if (z+1<=length(a1))
                param=a1(z+1);
                sigma(z+1)=sigma(z)*(1-(z*param)^2);
                fpe(z,1)=((L+(z+1))/(L-(z+1)))*sigma(z+1);
                fpe(z,2)=fpe(z,1)/FPE0;
        endif
elseif (strcmp(c,'KIC')==1)
        if (z+1<=length(a2))
                param=a2(z+1);
                sigma(z+1)=sigma(z)*(1-(z*param)^2);
                fpe(z,1)=((L+(z+1))/(L-(z+1)))*sigma(z+1);
                fpe(z,2)=fpe(z,1)/FPE0;
        endif
elseif (strcmp(c,'AICc')==1)
        if (z+1<=length(a3))
                param=a3(z+1);
                sigma(z+1)=sigma(z)*(1-(z*param)^2);
                fpe(z,1)=((L+(z+1))/(L-(z+1)))*sigma(z+1);
                fpe(z,2)=fpe(z,1)/FPE0;
        endif
elseif (strcmp(c,'AIC')==1)
        if (z+1<=length(a4))
                param=a4(z+1);
                sigma(z+1)=sigma(z)*(1-(z*param)^2);
                fpe(z,1)=((L+(z+1))/(L-(z+1)))*sigma(z+1);
                fpe(z,2)=fpe(z,1)/FPE0;
        endif
else
        if (z==1)
```

```
                        param=a5(z+1);
                        sigma(z+1)=sigma(z)*(1-(z*param)^2);
                        fpe(z,1)=((L+(z+1))/(L-(z+1)))*sigma(z+1);
                        fpe(z,2)=fpe(z,1)/FPE0;
                endif
        endif
endfor
min_fpe=min(min(fpe));
[nr,nc]=size(fpe);
p_order=0;
for r=1:nr
        for c=1:nc
                if (fpe(r,c)==min_fpe)
                        p_order=r;
                endif
        endfor
endfor
criterion=0;
if (strcmp(c,'AKICc')==1)
        criterion=1;
elseif (strcmp(c,'KIC')==1)
        criterion=2;
elseif (strcmp(c,'AICc')==1)
        criterion=3;
elseif (strcmp(c,'AIC')==1)
        criterion=4;
else
        criterion=5; endif

pv=prediction(p_order,criterion);
prob=abs((1-min_fpe)*100);
```

Bibliografia

[AGM10] Y. Asnar, P. Giorgini, and J. Mylopoulos. Goal-driven risk assessment in requirements engineering. *Requirements Engineering*, volume 16:pages 101–116, 2010.

[Aka69] Y. Akaike. Fitting autoregressive models for prediction. *Annals of the Institute of Statistical Mathematics*, volume 21:pages 243–247, 1969.

[Bit05] S. Bittanti. *Serie Temporali e Processi Casuali*. Pitagora Editore, 2005.

[Cal10] L. Calaon. *Confronto funzionale e prestazionale di motori CEP attraverso un caso d'uso reale*. PhD thesis, Università degli Studi di Padova, 2010.

[CFP+11] C. Cappiello, A. Mello Ferreira, B. Pernici, P. Plebani, and A. Ciuca. Refined co-design methodology. Technical report, GAMES (Green Active Management of Energy in IT Service centres), 2011.

[CPP+10] C. Cappiello, B. Pernici, P. Plebani, D. Arnone, I. Salomie, A. Ciuca, J. Liu, and A. Kipp. Co-design methodology. Technical report, GAMES (Green Active Management of Energy in IT Service centres), 2010.

[EE11] Y. Engel and O. Etzion. Towards proactive event-driven computing. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, pages 125–136, New York, NY, USA, 2011. ACM.

[EN11] O. Etzion and P. Niblett. *Event Processing In Action*. Manning Publications Co, 2011.

[Esp12] EsperTech. Esper: Event processing for java, 2012. [Online; controllata il 8-febbraio-2012].

[GAM10] Games (green active management of energy in it service centres), 2010. [Online; controllata il 2-febbraio-2012].

[GR09] M. Gualtieri and J. R. Rymer. The forrester wave: Complex event processing (cep) platforms q3 2009. 2009.

[Hau05] F. Haugen. Discrete-time signals and systems, February 2005.

[JKC+10] T. Jiang, A. Kipp, C. Cappiello, M. Fugini, G.R. Gangadharan, A. Mello Ferreira, B. Pernici, P. Plebani, I. Salomie, T. Cioara, I. Anghel, W. Christmann, E. A. Henis, R. I. Kat, M. Lazzaro, A. Ciuca, and D. Hatiegan. Layered green performance indicators definition. Technical report, GAMES (Green Active Management of Energy in IT Service centres), 2010.

[Luc07] D. Luckham. A short history of complex event processing - part 1: Beginnings, 2007.

[MC11] A. Margara and G. Cugola. Processing flows of information: from data stream to complex event processing. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, pages 359–360, New York, NY, USA, 2011. ACM.

[McC02] D. W. McCoy. Business activity monitoring: Calm before the storm, 2002.

[Pal09] G. K. Palshikar. Simple algorithms for peak detection in time-series. In *Proceedings of the 1st internationl conference on Advanced Data Analysis, Business Analytics and Intelligence*, 2009.

[Pas08] A. Paschke. Design patterns for complex event processing, 2008.

[Pro12] ProgressSoftwareCorporation. Apama - event processing platform, 2012. [Online; controllata il 7-febbraio-2012].

[TFDB08] Y. Taher, M.C. Fauvet, M. Dumas, and D. Benslimane. Using cep technology to adapt messages exchanged by web services. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 1231–1232, New York, NY, USA, 2008. ACM.

[Wik11a] Wikipedia. Analisi delle serie storiche - wikipedia, l'enciclopedia libera, 2011. [Online; controllata il 18-febbraio-2012].

[Wik11b] Wikipedia. Application server - wikipedia, l'enciclopedia libera, 2011. [Online; controllata il 15-marzo-2012].

[Wik12a] Wikipedia. Correlogramma - wikipedia, l'enciclopedia libera, 2012. [Online; controllata il 25-febbraio-2012].

[Wik12b] Wikipedia. Serie storica - wikipedia, l'enciclopedia libera, 2012. [Online; controllata il 24-febbraio-2012].