



UNIVERSITÀ DEGLI STUDI DI PADOVA
FACOLTÀ DI INGEGNERIA

Consistenza di workflow temporali nel linguaggio YAWL

Laureando: Christian Marchi

Relatore: Prof. Silvana Badaloni
Correlatore: Dott.Ing. Marco Falda

Corso di Laurea Specialistica in Ingegneria Informatica

Anno accademico 2009/2010

9 marzo 2010

Indice

Introduzione	7
1. I Workflow Management Systems (WfMS).....	9
1.1 Introduzione.....	9
1.2 Business Process (Processo aziendale).....	11
1.2.1 Business Process Management (BPM).....	13
1.2.2 Business Process Reengineering (BPR).....	14
1.3 I workflow e i sistemi per gestirli.....	15
1.4 Modelli di un workflow.....	17
1.5 Categorie di workflow.....	18
1.6 Architettura WfMS.....	20
1.7 Interfacce.....	23
1.8 Temporalità nei WfMS.....	25
2. Fuzzy Logic.....	31
2.1 Introduzione.....	31
2.2 L'imperfezione.....	31
2.3 Teoria della probabilità.....	33
2.4 Storia della logica fuzzy.....	34
2.5 Principi della logica fuzzy.....	36
2.6 Fuzzy set (insiemi fuzzy).....	39
2.7 Concetti temporali.....	46
2.8 Modellazione dell'informazione temporale.....	48

3. YAWL.....	57
3.1 Introduzione.....	57
3.2 I pattern.....	58
3.3 Editor YAWL.....	61
3.4 Architettura di YAWL.....	65
3.5 Modellazione dati.....	70
3.6 Comunicazione tra editor ed engine tramite XML, XPath e XQuery...74	
3.7 Vincoli temporali.....	78
3.8 Modellazione degli aspetti temporali.....	80
3.9 YAWL Engine.....	85
4. Estensione fuzzy di YAWL.....	93
4.1 Introduzione.....	93
4.2 Inserimento vincoli temporali nell'editor.....	96
4.3 Passaggio tra editor ed engine.....	99
5. Consistenza del workflow temporale.....	101
5.1 Introduzione.....	101
5.2 TCSP (Temporal Constraint Satisfaction Problem).....	102
5.3 Classi di TCSP.....	104
5.4 Consistenza di una rete.....	106
5.4.1 STP (Simple Temporal Problems).....	107
5.4.2 Grafo delle distanze.....	107
5.4.3 Algoritmo Floyd-Warshall.....	108
5.4.4 Algoritmi di Path-Consistency.....	109

5.5 Implementazione del controllo di consistenza in YAWL.....	118
5.5.1 I Singleton in Java.....	118
5.5.2 La classe Sequenza.....	120
5.5.3 La classe Consistenza.....	121
5.5.4 Implementazione dell'algoritmo Floyd-Warshall.....	126
5.5.5 Implementazione dell'algoritmo P ³ C.....	128
5.5.6 Esempio di inconsistenza.....	132
6. Conclusioni e sviluppi futuri.....	135
Allegato.....	137
Bibliografia.....	143

Introduzione

L'obiettivo della tesi sperimentale sviluppata è il controllo della consistenza, a tempo di progettazione, di un workflow temporale creato con YAWL. Un workflow temporale può essere visto come un processo in cui devono essere svolti alcuni lavori in un tempo prestabilito. I workflow che sono stati presi in considerazione in questo lavoro di tesi sono workflow temporali estesi tramite la fuzzy logic [8][2], per permettere di rappresentare le nozioni di incertezza e vaghezza. Questa estensione fuzzy dei workflow è molto importante perchè permette di riprodurre in modo più flessibile il mondo reale che è affetto da incertezza e vaghezza, ciò è fatto rilassando i vincoli che sono presenti nel workflow che altrimenti sarebbero troppo rigidi e non riuscirebbero a modellare facilmente scenari realistici. Il software utilizzato per la creazione dei workflow è YAWL [1] (Yet Another Workflow Language), che si suddivide in due parti: un editor grafico per la creazione del workflow e un engine per l'esecuzione del workflow stesso. L'obiettivo principale della tesi è il controllo di consistenza a tempo di progettazione mentre in precedenza l'introduzione del controllo veniva effettuato a tempo di esecuzione. Per raggiungere questo obiettivo bisogna prima comprendere il linguaggio YAWL e poi ricercare e adattare al problema gli algoritmi che riguardano lo "stato dell'arte" del controllo di consistenza. Dato che il controllo viene effettuato in fase di progettazione del workflow nella tesi è stato considerato solo l'editor YAWL. Il raggiungimento dell'obiettivo principale permette di risparmiare tempo e rende più efficiente l'esecuzione dei workflow. L'algoritmo scelto per effettuare il controllo di consistenza è il P³C [4] per la sua miglior efficienza. Per l'implementazione dell'algoritmo scelto sono state implementate varie

operazioni su trapezi normalizzati che prima non erano previste nel modello dei workflow considerati [8].

La tesi è strutturata nel seguente modo: nel primo capitolo viene data una descrizione ad ampio raggio dei workflow, di come essi vengono gestiti tramite i WfMS (Workflow Management System); vengono poi introdotti i vari modelli di workflow, le loro categorie e viene descritta l'architettura generale di un WfMS. Infine viene trattata la dimensione temporale nei WfMS. Nel capitolo due si affronta il tema della fuzzy logic, descrivendone la storia, i principi generali, si descrivono gli insiemi fuzzy (Fuzzy set). Infine si spiega come vengono modellati gli aspetti temporali con questo tipo di logica. Il terzo capitolo descrive YAWL, ne specifica le caratteristiche principali e si focalizza soprattutto sull'editor, senza tralasciare comunque la spiegazione dell' engine. Il quarto capitolo riguarda l'estensione fuzzy dei workflow temporali tramite il software YAWL: si mostra come vengono modificati i vari tipi di vincolo per rendere possibile la gestione dell'incertezza e il passaggio dei vincoli dall' editor all'engine. Il quinto capitolo è il cuore del lavoro svolto: qui viene spiegato il concetto di consistenza, vengono descritti i TCSP (Temporal Constraint Satisfaction Problem) cioè il soddisfacimento di problemi con vincoli temporali, vengono riportati gli algoritmi principali per la soddisfazione dei vincoli e infine viene descritto come inserire il controllo di consistenza in YAWL. Questo capitolo contiene anche la spiegazione dell'implementazione dell'algoritmo di Floyd-Warshall e dell'algoritmo P^3C adattati a risolvere il problema che costituisce l'obiettivo di questa tesi. Vengono inoltre presentate e spiegate le varie modifiche effettuate al codice e le novità inserite. Infine vengono mostrati alcuni esempi di come funziona il controllo di consistenza in YAWL. Il sesto e ultimo capitolo riguarda le conclusioni tratte dal lavoro svolto e alcuni sviluppi futuri.

CAPITOLO 1

I Workflow Management Systems (WfMS)

1.1 Introduzione

Nel 1996 la Workflow Management Coalition (WfMC) ha definito i workflow come: *“l’automazione parziale o totale di un processo aziendale, durante il quale documenti, informazioni o compiti sono scambiati tra i vari partecipanti per essere eseguiti, rispettando un insieme di regole procedurali.”*

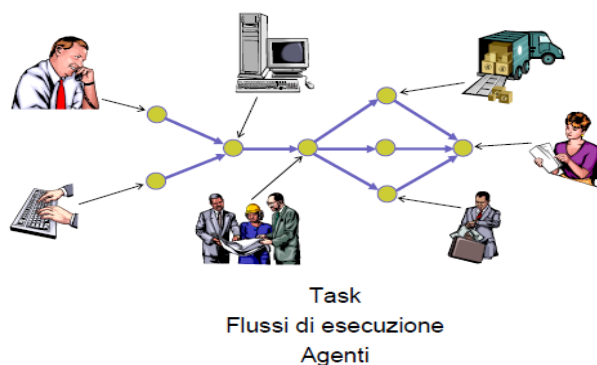


Figura 1.1 I workflow

Sempre la Workflow Management Coalition (WfMC) definisce i Workflow Management Systems (WfMS) come: *“Sistemi che definiscono, creano e*

gestiscono l'esecuzione di workflow attraverso l'uso di applicazioni software. Essi sono in grado di interpretare gli schemi di workflow, interagire con gli agenti e invocare l'uso di strumenti e applicazioni automatizzate." In pratica un sistema di gestione di Workflow (WfMS) è un sistema software che supporta l'esecuzione coordinata di semplici attività, assegnandole a esecutori umani o automatizzati, per raggiungere un obiettivo prefissato di un processo di business. Un processo definisce alcune azioni che devono essere eseguite da una persona, da un sistema software o da entrambi. Le specifiche di un workflow concernono la descrizione degli aspetti che sono rilevanti per il controllo e il coordinamento delle attività durante la loro esecuzione, così come i rapporti tra le attività stesse. Le informazioni richieste per l'esecuzione di un workflow sono memorizzate da un sistema di gestione di database (DBMS). Nella maggior parte dei casi, il DBMS è un tradizionale database relazionale che non prevede alcuna funzionalità volta alla gestione degli aspetti temporali relativi all'informazione memorizzata, costringendo il progettista a gestire tali aspetti esplicitamente. L'informazione temporale che dovrebbe essere gestita in un workflow coinvolge vari campi:

- gli istanti in cui inizia e termina l'esecuzione di un'attività;
- le deadline per il completamento di un'attività;
- il validity time di un dato;
- gli intervalli temporali tra l'esecuzione di due attività differenti.

Per gestire al meglio questi aspetti, si può adottare un DBMS temporale (T-DBMS) che è basato su un modello in cui il tempo e le dimensioni temporali sono facilmente rappresentabili e che può migliorare in modo consistente lo sviluppo di un sistema rivolto alla gestione automatica di workflow. Sfortunatamente, a causa della mancanza di sistemi di database temporali affidabili e ad alte prestazioni, non esistono WfMS che giacciono al di sopra di un database temporale.

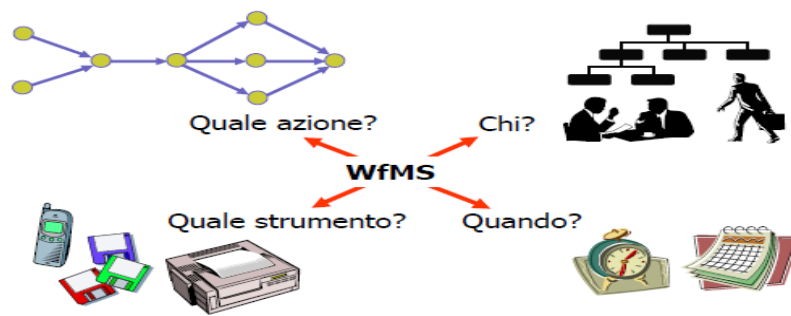


Figura 1.2. I WfMS

I WfMS vengono utilizzati principalmente per le seguenti ragioni:

Benefici tangibili:

- Riduzione dei costi
- Aumento della produttività
- Riduzione del tempo di esecuzione

Benefici non tangibili:

- Miglioramento della qualità dei servizi
- Gestione più efficace delle informazioni
- Maggior supporto alle decisioni e alla pianificazione

1.2 Business Process (Processo aziendale)

Il processo aziendale (o business process) è un insieme di attività correlate, svolte all'interno dell'azienda che creano valore trasformando delle risorse (input del processo) in un prodotto (output del processo) destinato ad un soggetto interno o esterno all'azienda (cliente). Il processo è teso al raggiungimento di un obiettivo aziendale, determinato in sede di pianificazione se questa è presente. Tanto le risorse quanto il prodotto

possono essere beni, servizi o informazioni oppure una combinazione di questi elementi. La trasformazione dell'input in output può essere eseguita con l'impiego di lavoro umano, di macchine o di entrambi. Nelle aziende dotate di un sistema di gestione della qualità, in accordo alla norma ISO 9001, i processi aziendali devono essere misurabili e monitorabili nel tempo mediante l'utilizzo di indicatori di prestazione chiave. All'interno del processo si possono distinguere tre elementi principali:

1. **Attività:** parte di un processo che non include decisioni e che quindi non è utile decomporre ulteriormente (sebbene la scomposizione sia di per sé possibile). Le attività, quindi, possono dividersi in operazioni su oggetti fisici o informativi oppure in una decisione assunta da un attore coinvolto nel processo.
2. **Sotto-processo:** parte del processo che comprende più attività e ha propri attributi in termini di obiettivo, input e output, contribuendo però nel contempo al raggiungimento dell'obiettivo più generale del processo.
3. **Progetto:** particolare tipo di processo aziendale, volto al conseguimento di un obiettivo specifico in un determinato tempo e con determinate risorse, che non è la sostanziale ripetizione di processi già svolti.

In un processo vengono normalmente coinvolti più organi aziendali e il loro apporto è coordinato attraverso un flusso di lavoro da cui il termine workflow. Il coordinamento può essere eseguito in vari modi:

- formalizzando in procedure i compiti e le responsabilità degli organi aziendali che intervengono nel processo.
- attribuendo la necessaria autorità funzionale ad un'apposita figura manageriale (process manager).
- raggruppando in un'unica unità organizzativa tutti gli organi coinvolti nel processo definito.

Vengono considerati clienti tutti coloro ai quali è destinato l'output di un processo, anche se interni all'azienda. Da questo punto di vista si distinguono:

- i processi primari, che hanno come clienti soggetti esterni all'azienda;
- i processi di supporto, che hanno come clienti soggetti interni all'azienda e che, quindi, supportano i processi primari.

Un modo semplice e diffuso di rappresentazione dei processi è quello che si serve del *diagramma di flusso (flow chart)*. Si tratta di un grafo in cui i nodi rappresentano le attività e i cui archi orientati la sequenza cronologica delle azioni stesse, nel senso che se un arco esce da un'attività ed entra in un'altra, la seconda viene eseguita dopo la prima. Nei nodi diverse forme geometriche (*blocchi*) rappresentano diversi tipi di attività. La rappresentazione può essere arricchita disponendo le attività su diverse colonne o righe (a seconda che il diagramma si sviluppi verticalmente o orizzontalmente) in corrispondenza dei diversi organi aziendali che le svolgono; in alternativa (o in aggiunta) si possono differenziare le attività svolte dai vari organi con una diversa colorazione dei blocchi (questa soluzione ovvia al problema per cui, nel caso di incolonnamento, si tende a percepire come maggiormente importanti le attività poste nelle colonne più a sinistra). Oltre ai diagrammi di flusso sono stati sviluppati anche altri metodi di rappresentazione grafica dei processi, più sofisticati, tra i quali si può citare la Business Process Modeling Notation (BPMN).

1.2.1 Business Process Management (BPM)

Il Business process management è l'insieme di attività necessarie per definire, ottimizzare, monitorare e integrare i processi aziendali, al fine di creare un processo orientato a rendere efficiente ed efficace il business dell'

azienda. Il *BPM* è una via intermedia fra la gestione d'impresa e l'Information Technology, ed è riferito a processi operativi, che interessano variabili quantitative e sono ripetuti su grandi volumi quotidianamente. Il *BPM* differisce dal *BPR* (Business Process Re-engineering), perché mira ad un miglioramento incrementale dei processi, mentre il secondo ad un miglioramento radicale. I software di *BPM* dovrebbero velocizzare e semplificare la gestione e il miglioramento dei processi aziendali. Per ottenere questi obiettivi, un software di *BPM* deve monitorare l'esecuzione dei processi, consentire ai manager di fare analisi e cambiare tecnologia e organizzazione sulla base di dati concreti, piuttosto che in base ad opinioni soggettive. Tali operazioni sono talora svolte da software differenti che comunicano tra loro, da programmi che misurano i dati e altri che contengono la descrizione dei processi "aggiornabile" con i dati dell'operatività. I programmi che si occupano della rilevazione degli indicatori di prestazione chiave (*KPI*) forniscono dei resoconti sintetici sull'operatività dei processi, e consentono un dettaglio dell'indicatore che può arrivare dal livello globale della società al singolo operatore/macchina. L'utilizzo di un workflow è una delle tecniche più diffuse nell'ambito del Business Process Management (BPM).

1.2.2 Business Process Reengineering (BPR)

La riprogettazione dei processi aziendali o *Business Process Reengineering (BPR)* è un intervento organizzativo di profonda revisione dei procedimenti operativi che non risultano più adeguati alle necessità aziendali. Per processo si intende un insieme di attività interconnesse che portano ad un risultato finale identificabile dal cliente, che quindi contribuisce alla formazione di valore per l'azienda. Lo stimolo per un intervento può venire dai risultati di un benchmarking, da una analisi di customer satisfaction, da un mutamento del quadro operativo, dalla ridefinizione degli obiettivi

aziendali, dall' evidenza di altri parametri critici che indichino la necessità o l'opportunità di migliorare l'efficacia e l'efficienza di un processo aziendale, dalla necessità di introdurre nuove metodologie di lavoro, o da altre situazioni. Lo scopo del reengineering non è il raggiungimento di miglioramenti marginali per l'organizzazione, quanto piuttosto ottenere prestazioni aziendali eccellenti ed effettuare una svolta sostanziale nelle performance. Il mezzo con cui giungere ad un miglioramento sostanziale non consiste nella modifica degli schemi esistenti, ma in una totale riprogettazione delle modalità con le quali si compie il lavoro. La progettazione infatti è di primaria importanza, un processo non ben pianificato, anche se svolto da persone esperte e motivate, non potrà essere eseguito in maniera soddisfacente.

1.3 I workflow e i sistemi per gestirli

Un sistema per la gestione di workflow è un software in grado di fornire supporto per la definizione, la coordinazione ed l'esecuzione di processi, in termini di attività da svolgere. Il processo di controllo svolge il proprio lavoro schedulando le attività e assegnandole agli agenti esecutivi. I WfMS sono sistemi molto complessi che includono componenti differenti tra cui: l'interprete del linguaggio di definizione di un processo (PDL) utilizzato per modellare formalmente il processo di business; il progettista del modello di processo, che aiuta l'utente a definire in modo corretto un modello in accordo con il PDL supportato; l'unità di gestione delle risorse che si occupa di assegnare i processi agli agenti esecutori, l'unità di connessione verso il database per accedere ai dati memorizzati e il gestore delle transazioni. Questa è solamente una delle definizioni che sono state proposte nel tempo, con il termine workflow nei vari contesti ci si può riferire :

- ad un processo di business.

- alla specifica o alla mappatura di un processo.
- Ad un'applicazione che facilita il coordinamento e la collaborazione delle persone che lavorano per la realizzazione di un processo.

Alcune definizioni tendono ad enfatizzare il fatto che il workflow è un concetto fortemente legato al Business Process Reengineering, altre invece si limitano a definirlo come un semplice strumento di routing, cioè di assegnazione e consegna automatica di compiti da svolgere da un utente ad un altro. In realtà il workflow si occupa di entrambi gli aspetti tecnologici e organizzativi: la confusione sulla sua definizione è principalmente dovuta al fatto che la tecnologia relativa al workflow è ancora ad uno stadio iniziale di sviluppo; negli ultimi anni, però, grazie al lavoro di quella associazione che ha preso il nome di Workflow Management Coalition, sono stati fatti notevoli passi avanti nella standardizzazione e nella omogeneizzazione delle varie concezioni di workflow presenti sul mercato. La Workflow Management Coalition (WfMC) nasce nel 1996 come una tavola rotonda fra esperti e studiosi di workflow. Gli sforzi sono orientati a fornire linee guida e standard per lo sviluppo di prodotti per la gestione di workflow. La formalizzazione apportata dalla WfMC ha lo scopo ultimo di realizzare la massima compatibilità tra i vari strumenti, in modo da permettere alle organizzazioni di adottare differenti modelli di workflow a seconda delle esigenze senza che ci si trovi di fronte a problemi di incomunicabilità fra i dati trattati da ognuno di essi. Secondo la sua definizione ufficiale, il workflow rappresenta il concetto di automazione di una parte o dell'intero processo gestionale, dove documenti, informazioni e compiti vengono passati da un attore ad un altro per ricevere una qualsiasi forma di trasformazione, seguendo un determinato insieme di regole procedurali. Il workflow riassume dunque una serie di processi che, posti in relazione tra loro, definiscono un contesto di lavoro strutturato, dove più unità lavorative, siano esse costituite da risorse umane, gruppi di lavoro o sistemi computerizzati, lavorano in concomitanza per un determinato obiettivo scambiandosi in modo automatico informazioni e compiti. Oltre alla sequenza delle varie attività che costituiscono un

processo, il workflow rappresenta ulteriori elementi fondamentali che servono per comprendere e gestire al meglio l'unità di lavoro. Nasce quindi l'esigenza da parte di un workflow di definire e rappresentare più modelli, non solo quello relativo al processo di business.

1.4 Modelli di un workflow

I workflow si possono suddividere essenzialmente in tre modelli:

Modello organizzativo:

Il modello organizzativo formalmente raffigura l'organizzazione della compagnia in cui il processo è stato rilasciato. Le informazioni riguardano principalmente gli agenti, i loro compiti e la gerarchia all'interno della compagnia, tutte queste informazioni vengono mantenute in tabelle di database, la cui struttura risulta essere indipendente sia dall'organizzazione che dal dominio di applicazione. Ogni agente appartiene ad un gruppo: un gruppo coinvolge agenti che cooperano nella realizzazione del medesimo progetto oppure agenti che lavorano nella stessa area geografica. Ogni agente è individuato da:

- Un insieme di abilità individuali.
- Un insieme di processi la cui esecuzione gli può essere assegnata.
- Uno o più gruppi.
- Un supervisore.

Modello di processo

Il modello di processo descrive lo schema da seguire; anche in questo caso tutte le informazioni vengono memorizzate in tabelle di database la cui struttura si determina staticamente ed è indipendente dal dominio applicativo. Il progettista deve specificare il primo task da eseguire, i task successivi, le

condizioni da verificare per la selezione di archi di uscita differenti e i criteri per l'assegnazione dei task agli agenti. Le tabelle sono:

- Workflow, contiene i nomi dei modelli di processo registrati nel WfMS con i corrispondenti primi task da attivare.
- Worktask, memorizza i nomi dei task di tutti i modelli di processo e le competenze richieste per la loro esecuzione.
- RoutingTask, descrive la tipologia di ogni processo di routing.
- Next, memorizza il task successore per tutti i task segnalati.
- AfterFork, memorizza i criteri adottati per attivare gli archi di uscita da un task di routing per ogni modello di processo.

Modello informativo

Il modello informativo descrive il dominio applicativo dei dati che ogni processo deve gestire e la storia delle esecuzioni avvenute. La struttura delle tabelle viene definita durante la progettazione del sistema e le tabelle successivamente vengono automaticamente popolate con le variabili di processo definite in precedenza. *Historical Data*. I dati storici contengono informazioni relative alle esecuzioni precedenti della specifica e sono indipendenti dal dominio applicativo attuale.

1.5 Categorie di workflow

I workflow si possono suddividere tre categorie dipendenti dalle loro funzioni e capacità:

Workflow produttivi

I workflow produttivi sono orientati alla produzione su larga scala e all'implementazione dei processi critici di un'azienda, e quindi coinvolgono un

grande numero di utenti e di sistemi. Gestiscono un gran numero di attività simili, ripetitive e prevedibili, ed il loro fine è ottimizzare la produttività intesa ad esempio come pezzi per unità di tempo. Tutti i processi di workflow sono orientati ad ottenere un alto livello di automazione, per trarre il massimo vantaggio dall'assenza di componente umana, necessaria spesso solo per gestire eventuali eccezioni. Solitamente hanno un canale dedicato per la comunicazione con gli utenti e la consegna del lavoro.

Workflow amministrativi

I workflow amministrativi sono caratterizzati dalla facilità di definizione e di modifica dei flussi dei processi. Spesso infatti si possono avere processi che evolvono secondo le necessità o l'esperienza acquisita durante l'utilizzo, come può essere il workflow relativo alla gestione di uffici comunali, dove le regole procedurali devono seguire l'evoluzione delle normative ministeriali. In questi workflow la flessibilità è molto più importante della produttività, e per questo motivo tipicamente gestiscono un numero di istanze ogni ora minore di almeno uno o due ordini di grandezza rispetto ai workflow produttivi.

Workflow collaborativi

I workflow collaborativi si focalizzano sugli strumenti di comunicazione e scambio di informazioni fra gruppi di lavoro. Non vi è necessità di avere un'alta produttività ed i processi sono tipicamente definiti solo in maniera vaga e molto semplice: ciò che conta è avere strumenti di confronto e condivisione. Al contrario degli altri tipi di workflow essi non si basano su un flusso che scorre in avanti verso uno stadio finale, ma prevedono una serie di iterazioni sullo stesso passo o addirittura ripercorrono quelli precedenti fino al raggiungimento di un accordo. I workflow collaborativi utilizzano spesso la posta elettronica per consegnare e negoziare il lavoro. Portato questo concetto agli estremi, si ha un workflow caratterizzato dalla più bassa produttività e dalla massima flessibilità nel definire processi: la situazione è

quella in cui ogni caso di lavoro richiede un processo diverso e quindi un workflow apposito, creato ad hoc (talvolta questa estremizzazione si traduce in una categoria a parte nota con il nome di *workflow ad hoc*).

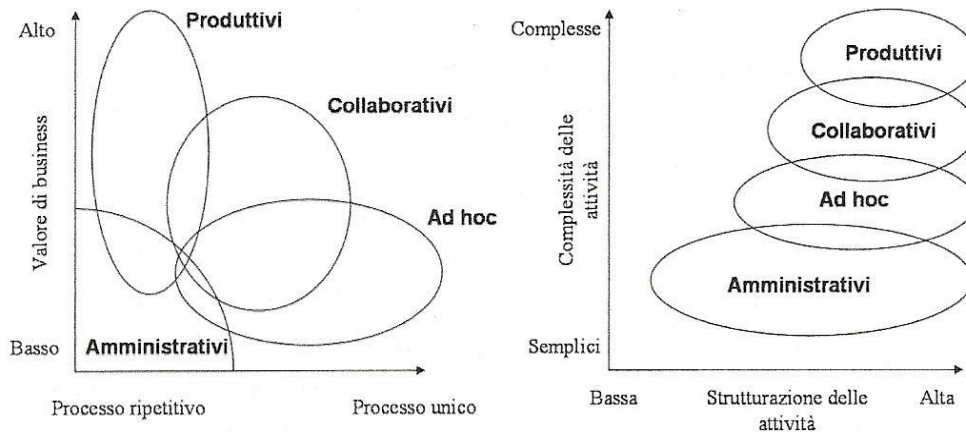


Figura 1.3 Categorie di workflow

1.6 Architettura WfMS

Ogni tipologia di WfMS ha delle caratteristiche in comune con tutti gli altri, per questo i WfMS sono scomponibili in tre grandi aree funzionali:

1. **Build time functions:** funzione volta alla definizione e modellazione del processo e delle relative attività
2. **Run time control functions:** funzione che interpreta il processo e gestisce le relative attività
3. **Run time interactions:** funzione che gestisce le interazioni con le persone e le applicazioni chiamate a svolgere le varie attività

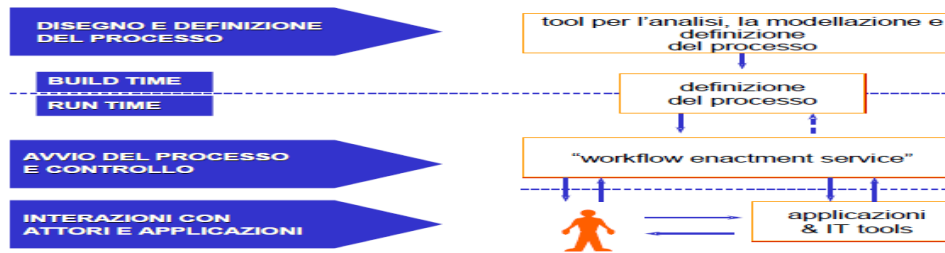


Figura 1.4 Fasi di progettazione di un workflow

In particolare, la prima fase relativa alla costruzione del workflow parte da un'analisi di Business Process Reengineering per studiare e ottimizzare la sequenza dei processi da definire e modellare. La fase successiva definisce invece il comportamento evolutivo dei processi, interpretandoli con soluzioni software che si occupano della creazione e del controllo delle istanze operazionali tramite la schedulazione delle attività e l'invocazione di appropriate risorse umane e tecnologiche. L'interazione con tali risorse viene infine gestita nell'ambito della terza fase, che si occupa delle soluzioni specifiche per gestire il trasferimento del controllo, l'evoluzione dello stato di processi e attività, il passaggio dei dati e le interfacce.

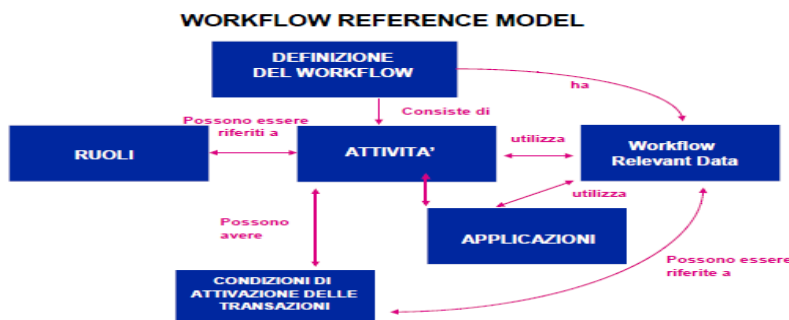


Figura 1.5 Modello di definizione del processo

Il supporto fisico allo sviluppo di workflow è garantito dai Workflow Enactment Services (WES), che consistono di uno o più motori capaci di creare, gestire ed eseguire le istanze di workflow.

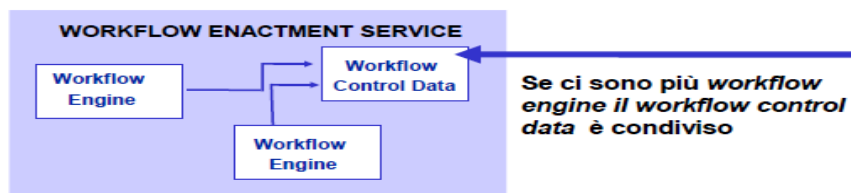


Figura 1.6 Servizi del workflow

Il “workflow engine” (motore) è la componente base del sistema, solitamente viene implementato su un server e riceve istruzioni da altri componenti del sistema. Il suo ruolo consiste nello stabilire le priorità dei diversi compiti, porli nel giusto ordine ed eseguire l'assegnazione delle risorse. Per poter fare ciò deve essere in grado di interpretare le regole assegnate in fase di definizione del processo. Inoltre è responsabile di:

1. Creazione ed eliminazione del processo
2. Controllo delle attività e scheduling
3. Attivazione delle risorse umane e tecnologiche

Quando lo sviluppo del processo, o altre operazioni di controllo da parte del workflow engine, sono basate su dati generali o aggiornati del WfMS, questi prendono il nome di “*WORKFLOW RELEVANT DATA*”. Le interazioni con le risorse esterne al motore di workflow avvengono attraverso due tipologie di interfacce:

1. *Client application*: interagisce con un worklist handler, cioè un componente software che organizza il lavoro di gestione delle risorse, selezionando e facendo progredire singoli lavori contenuti nella lista;
2. *Invoked application*: abilita il workflow engine a intraprendere una particolare attività, in genere priva di interfaccia utente. Nel caso una particolare attività utilizzi strumenti che richiedono una qualche forma di interazione con l'utente, essa viene in genere invocata tramite un'interfaccia per garantire maggiore flessibilità nella schedulazione delle operazioni.

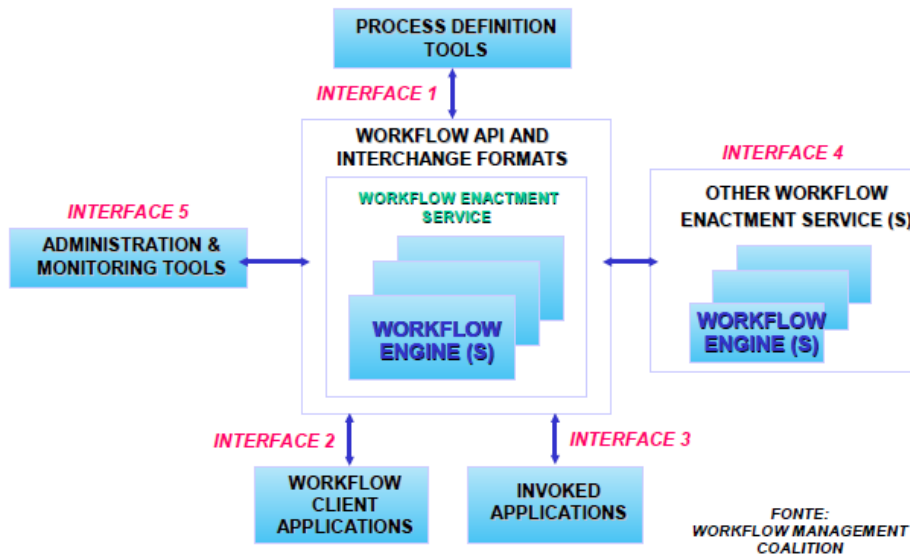


Figura 1.7 Modello di workflow: componenti e interfacce

Molti WfMS prevedono alcune funzioni di supervisione:

- Ridefinire le regole di distribuzione del lavoro tra i partecipanti
- Controllare lo svolgimento del processo
- Controllare il rispetto delle scadenze (deadlines)
- Verificare gli output di ogni singola attività
- Produrre statistiche

1.7 Interfacce

Nel modello di workflow sviluppato dalla WfMC esistono 5 interfacce ciascuna con caratteristiche diverse:

Interfaccia 1: i definition tool possono essere presenti all'interno del prodotto di workflow, oppure essere parte di un prodotto di business process design che include anche altre componenti di analisi e modellazione; in quest'ultimo caso il sistema di workflow dovrà solo importare ed interpretare (senza definire) i dati relativi al processo con il software per la definizione di processi, consentendo l'aggiornamento run-time delle regole di workflow.

Interfaccia 2: con l'applicazione client le API specificano come l'utente può interagire con il motore di workflow, visionando e completando le unità di lavoro presenti in coda. In sostanza è il responsabile del worklist handler per ciò che riguarda la selezione e l'aggiornamento dei singoli work item, cioè i lavori all'interno della worklist che devono essere eseguiti dai partecipanti del workflow. Un worklist handler può essere una semplice applicazione desktop affiancata da altri servizi come le mail e la cartella dei lavori in svolgimento (work-in-progress folder), fornisce un unico sistema di gestione e assegnazione dei compiti all'utente finale, oppure essere parte di un prodotto per la gestione di workflow e svolgere funzioni molto più sofisticate, come ad esempio il controllo sull'allocazione dei work item ai partecipanti, fornendo soluzioni come il bilanciamento dei carichi e la riassegnazione dei compiti.

Interfaccia 3: permette l'integrazione del motore di workflow con le eventuali applicazioni esterne che deve invocare. Consente di chiamare altre applicazioni mediante RPC (Remote ProcedureCall) o altre procedure.

Interfaccia 4: definisce i meccanismi per interagire con altri motori di workflow. Possono infatti esserci due sistemi ciascuno dei quali controlla solo una parte di un processo ed interagisce solo con quel sottogruppo di utenti e con i tool di applicazioni che riguardano le attività coinvolte nella parte di processo di cui è responsabile. In questo caso è necessaria sia una comune interpretazione della definizione di processo sia uno scambio reciproco di informazioni sulle attività. La natura e il flusso delle informazioni che i due sistemi si possono scambiare, sono rappresentati soprattutto da invocazioni di attività e sottoprocessi, lettura o scrittura o controllo di processi o attività.

Interfaccia 5: permette l'analisi ed il monitoraggio dei dati relativi all'esecuzione dei processi su diversi sistemi di workflow tramite strumenti esterni al motore di workflow. È utilizzata dagli amministratori per controllare le prestazioni del sistema globale e delle singole risorse. Permette inoltre di

- Ciascun agente lavora solo durante alcune ore del giorno;
- Ciascun agente lavora solo durante alcuni giorni della settimana;
- Ciascun agente può assentarsi, con un preavviso più o meno anticipato, dal luogo di lavoro a causa di malattie o vacanze;
- Ciascun agente (artificiale) può in qualsiasi momento smettere di essere operativo.

In questo caso si parla di criteri statici per la selezione di un agente; al contrario si parla di selezione dinamica nel momento in cui si deve tener conto del bilanciamento del lavoro assegnato ai vari agenti per evitare carichi troppo elevati. L'adozione di un modello organizzativo che esplicitamente considera questi aspetti temporali può migliorare l'utilizzabilità del sistema da parte del progettista il quale può facilmente rappresentare i vincoli temporali dell'organizzazione. Occorre definire due tipologie di espressioni per rappresentare in modo compiuto la disponibilità/indisponibilità di un agente:

1. *Daily time* rappresenta un intervallo temporale di ore per ciascun giorno;
2. *Periodic time* rappresenta un intervallo di ore della giornata specificato da un'espressione periodica.

Nel modello di processo vengono distinti vari aspetti temporali che sono:

- Versioning
- Eventi esterni
- Durate e ritardi
- Vincoli temporali
- Ritardi richiesti

Versioning

Le istanze di un workflow sono particolari occorrenze del processo. Istanze diverse dello stesso workflow possono eseguire un'attività sottostante di

task. Una classe di istanze è definita come un insieme di istanze che possono essere rappresentate dallo stesso sottografo in esecuzione. Un'istanza che rappresenta una deviazione dal modello di processo è un'eccezione. Una distinzione importante nel cambiamento del business che influenza i sistemi di workflow sottostanti riguarda la scelta di registrare le modifiche sul modello di workflow, su particolari istanze, o su entrambi. I cambiamenti effettuati al modello indicano un cambiamento del processo aziendale.

Se un task nell'istanza non si completa con successo, ciò può accadere per due ragioni:

- *System failure*: il recupero dopo un guasto del sistema, sono generalmente gestiti dall'attività del workflow che ha a che fare con le capacità di recovery del sistema sottostante (la base di dati);
- *Semantic failure*: un insuccesso semantico avviene quando un'istanza non è in grado di procedere in accordo al dato modello di workflow.

I cambiamenti nel modello di workflow vengono registrati come conseguenza di un insuccesso semantico. Si possono identificare 5 politiche di modifica che l'amministratore può eseguire:

Flush. Nelle situazioni di flush a tutte le istanze correnti è permesso terminare in accordo con il vecchio modello di processo, ma le nuove istanze sono pianificate per seguire il nuovo modello. Le nuove istanze potrebbero essere messe in attesa, fino a che tutte le istanze correnti raggiungono uno stato finale. Potrebbe anche essere permesso alle due specifiche di esistere simultaneamente.

Abort. Le istanze di workflow attive potrebbero essere abortite quando il modello di processo viene modificato. L'abort è più comunemente usato per l'adattamento di istanze singole, tutto ciò può essere il risultato di un cambiamento radicale nell'organizzazione.

Migrate. Il cambiamento coinvolge tutte le istanze correntemente in esecuzione, ma deve essere introdotto senza permettere alle istanze l'abort o il flush. Le istanze correnti sarebbero normalmente in fasi differenti

dell'esecuzione del processo. Il problema principale sorge quando un'istanza è in una fase in cui i task già completati hanno influenzato il processo in modo tale che i task seguenti non sono in grado di procedere in accordo alla nuova specifica. Tale migrazione potrebbe anche implicare il disfacimento o la compensazione dei task completati, per adattare l'istanza in accordo con la nuova specifica. Il caso peggiore avviene quando il processo completato deve essere riportato all'inizio, cioè tutto il lavoro è perso. Questo caso particolare è equivalente all'abort.

Adapt. L'adattamento comprende i casi di errore e le eccezioni, dove il modello di processo potrebbe non cambiare in modo definitivo, ma alcune istanze devono essere trattate in modo differente per alcune circostanze eccezionali e impreviste. Tali cambiamenti mirati nei workflow altrimenti ripetitivi e predicibili sono destinati a succedere una volta ogni tanto.

Build. Anche costruire un nuovo processo è un modo di cambiare il processo. La differenza è che il punto di partenza non è un modello preesistente dettagliato, ma una descrizione elementare, che coinvolge solo le basi, oppure un processo vuoto. Il vantaggio di includere il build come una delle classi del cambiamento del processo, sta nel fatto che, in questo modo, si permette l'inserimento di processi che possono non essere completamente definiti nell'insieme delle parti del processo che devono essere modificate. Quindi essenzialmente lo stesso meccanismo può rivolgersi alla definizione dinamica (build) come alla modifica dinamica (migrate e adapt).

Eventi esterni

Un evento è qualcosa che potrebbe accadere durante l'esecuzione di un processo, quindi è semanticamente non trascurabile. Gli eventi esterni rappresentano occorrenze di fatti non modellati all'interno del workflow e sono caratterizzati da un timestamp. Per questo motivo viene rappresentato in modo esplicito.

Durate e ritardi.

La durata è una proprietà temporale sia delle attività che dei flussi tra due attività distinte. La durata di un'attività rappresenta l'ampiezza temporale che separa l'istante iniziale e finale di esecuzione; la durata sull'arco, diversamente, indica l'intervallo tra l'istante finale del predecessore e l'istante iniziale del successore, in altre parole il ritardo introdotto dal sistema per svolgere le operazioni di schedulazione tra due attività successive.

Vincoli temporali.

Un modello di processo realistico deve comprendere anche varie tipologie di vincoli temporali tra i quali i vincoli assoluti, i vincoli periodici nonché i vincoli relativi.

Ritardi richiesti.

I ritardi richiesti rappresentano la distanza temporale minima che deve necessariamente intercorrere tra l'istante finale della prima attività e l'istante iniziale della seconda attività.

CAPITOLO 2

FUZZY LOGIC

2.1 Introduzione

Prima di addentrarsi nella spiegazione e nello studio della fuzzy logic bisogna aprire una parentesi sul concetto di imperfezione nei dati con relative cause. Spiegato questo concetto si può passare alla spiegazione della logica e dei fuzzy set associati e infine si applicherà questa teoria ai workflow. Un insieme fuzzy è caratterizzato dal fatto che il grado di appartenenza di ogni elemento all'insieme può essere un qualunque numero reale tra 0 e 1. Un insieme fuzzy A è definito quindi da una *funzione di appartenenza* $\mu_A : X \rightarrow [0,1]$, essendo X l'*universo di definizione*. L'universo X è un insieme convenzionale (o *crisp*). Infine verrà approfondito il discorso che riguarda l'applicazione del fuzzy agli aspetti temporali di un workflow.

2.2 L'imperfezione

L'imperfezione, sia che essa riguardi l'imprecisione o l'incertezza, è intrinseca nella realtà e deve essere trattata in ogni sistema informativo il cui scopo sia quello di fornire un modello completo e accurato. Essa può essere dovuta a tre fattori principali:

1. Vaghezza
2. Inconsistenza
3. Incertezza

Gli aspetti riguardanti le informazioni imperfette sono sempre stati modellati sfruttando la teoria della probabilità, ma negli ultimi 20 anni sono stati proposti e sviluppati nuovi modelli per trattare questa tipologia di dati. Il gran numero di soluzioni prese in considerazione dimostra che l'imperfezione è costituita da molti aspetti e per questo motivo la teoria della probabilità, per quanto ben formalizzata, non può e non deve essere l'unico modello normativo per l'analisi di dati affetti da imperfezione. Le cause di imperfezione sono molto diverse tra loro, infatti imprecisione e inconsistenza sono due proprietà relative al contenuto di un'affermazione, mentre l'incertezza è una proprietà che deriva dalla mancanza di informazione riguardante la realtà presa in considerazione. È quindi lecito affermare che imprecisione e inconsistenza sono essenzialmente proprietà dell'informazione stessa mentre l'incertezza è una proprietà che riguarda il rapporto tra l'informazione e la conoscenza del mondo analizzato.

Vaghezza

La vaghezza può essere descritta come una proprietà relativa ad un oggetto, un'informazione che ne indica la non conoscenza, o meglio la non perfetta conoscenza. Quando si fa riferimento ad un'informazione definita vaga si vuole intendere che tale informazione, nel suo insieme, non è nota completamente. A volte la vaghezza caratterizza la conoscenza vera e propria di un oggetto, altre volte può fare riferimento al contesto nel quale inserire tale oggetto.

Inconsistenza

Solitamente l'inconsistenza è una proprietà che fa riferimento alla combinazione di due o più componenti informative. È vero anche che una

singola affermazione può essere inconsistente, ma il più delle volte essa è il risultato di una manipolazione inappropriata di realtà verificate più elementari. Il concetto di errore è intrinseco nella definizione di inconsistenza, in quanto altrimenti si parlerebbe di vaghezza o incertezza, a seconda dei casi. Un'affermazione conflittuale porta ad una incoerenza nelle conclusioni che si traduce nell'inconsistenza dei dati. Esistono affermazioni più o meno inconsistenti dovute all'imprecisione nei dati di partenza oppure all'erroneità dei dati stessi o ancora alla non capacità di combinare dati corretti.

Incerteza

Il terzo aspetto dell'imperfezione, l'incerteza, concerne il livello di conoscenza di un agente (umano o artificiale) nei confronti del rapporto tra il mondo e le affermazioni relative al mondo. L'affermazione potrebbe essere vera o falsa, ma la conoscenza da parte dell'agente riguardo al mondo non gli consente di avere la certezza su quell'affermazione.

2.3 Teoria della probabilità

Prima dell'avvento della logica fuzzy per descrivere i dati affetti da imperfezione e quindi la realtà veniva utilizzata la teoria della Probabilità, che comunque è ancora utilizzata. I matematici si riferiscono alle probabilità come a numeri nell'intervallo da 0 a 1, assegnati ad "eventi" la cui occorrenza è casuale. Le probabilità $P(E)$ sono assegnate ad eventi E secondo gli assiomi della probabilità. La probabilità che un evento E avvenga dato il verificarsi noto di un evento F è la probabilità condizionata di E dato F . Se la probabilità condizionale di E dato F è la stessa della probabilità ("non condizionale") di E , allora E ed F sono detti eventi indipendenti. Due concetti cruciali nella teoria della probabilità sono quelli di variabile casuale e di distribuzione probabilistica di una variabile casuale. Sono almeno 4 le interpretazioni principali scaturite dalla definizione della teoria probabilistica:

La teoria classica. In questa definizione, sostenuta anche da Laplace, si assume l'esistenza di un insieme fondamentale di eventi equiprobabili. La probabilità di un evento particolare diventa di conseguenza il rapporto tra i casi favorevoli e tutti i possibili casi.

La teoria della frequenza relativa. La probabilità è essenzialmente il limite di convergenza delle frequenze relative dopo un numero considerevole di esperimenti indipendenti.

La probabilità soggettiva(Bayesiana). Per la scuola di pensiero Bayesiana la misura di probabilità riflette la credibilità, espressa dall'agente che sta studiando l'universo, che un evento occorra o che una proposizione sia vera. É quindi una misura del tutto soggettiva, personale senza alcun fondamento scientifico.

La probabilità logica. Keynes (1962) definisce la probabilità come la relazione logica tra una proposizione e un dato di fatto. Dal momento che le proposizioni possono essere in ultima analisi vere o false (senza l'introduzione di proposizioni fuzzy) esse vengono giudicate probabili in relazione con la nostra conoscenza corrente. Una proposizione è probabile rispetto ad un dato di fatto indipendentemente dal fatto che qualcuno lo pensi o meno.

2.4 Storia della logica fuzzy

Nei primi anni '60, Lotfy A. Zadeh, professore all'Università della California a Berkeley e, molto noto per i suoi contributi alla teoria dei sistemi, cominciò ad avvertire che le tecniche tradizionali di analisi dei sistemi erano eccessivamente ed inutilmente accurate per molti dei problemi tipici del

mondo reale. L'idea di "grado d'appartenenza", concetto divenuto poi la spina dorsale della teoria degli insiemi fuzzy, fu da lui introdotta nel 1964, e lo portò in seguito, nel 1965, alla pubblicazione di un primo articolo, ed alla nascita della logica fuzzy. Il concetto di insieme fuzzy, e di logica fuzzy, attirò le aspre critiche della comunità accademica; nonostante ciò, studiosi e scienziati di tutto il mondo appartenenti a diversi campi di studio divennero seguaci di Zadeh. In Giappone la ricerca sulla logica fuzzy cominciò con due piccoli gruppi universitari fondati sul finire degli anni '70: il primo era guidato, a Tokio, da T. Terano e H. Shibata, mentre l'altro si stabilì a Kanasai sotto la guida di K. Tanaka e K. Asai. Al pari dei ricercatori americani, questi studiosi si scontrarono, nei primi tempi, con un'atmosfera fortemente avversa alla logica fuzzy. E tuttavia, la loro tenacia e il duro lavoro si sarebbero dimostrati estremamente fruttuosi già dopo un decennio: i ricercatori giapponesi, i loro studenti, e gli studenti di questi ultimi produssero molti importanti contributi sia alla teoria che alle applicazioni della Logica Fuzzy. Nel 1974, Seto Assilian ed Ebrahim H. Mamdani svilupparono, in Gran Bretagna, il primo sistema di controllo di un generatore di vapore, basato sulla logica fuzzy. Nel 1976, la Blue Circle Cement e il SIRA idearono la prima applicazione industriale della logica fuzzy, per il controllo di una fornace per la produzione di cemento. Il sistema divenne operativo nel 1982. Nel corso degli anni '80, diverse importanti applicazioni industriali della logica fuzzy furono lanciate con pieno successo in Giappone. Dopo otto anni di costante ricerca, sviluppo e sforzi di messa a punto, nel 1987 S. Yasunobu ed i suoi colleghi della Hitachi realizzarono un sistema automatizzato per il controllo operativo dei treni metropolitani della città di Sendai. Un'altra delle prime applicazioni di successo della logica fuzzy fu un sistema per il trattamento delle acque di scarico sviluppato dalla Fuji Electric. Queste ed altre applicazioni motivarono molti ingegneri giapponesi a dedicarsi ad applicazioni inedite, e ciò ha portato ad un vero boom della logica fuzzy. Una tale esplosione era peraltro il risultato di una stretta collaborazione, e del trasferimento tecnologico, tra Università ed Industria. Due progetti di ricerca nazionali su larga scala furono decisi da agenzie governative giapponesi nel 1987, il più noto dei quali

sarebbe stato il Laboratory for International Fuzzy Engineering Research (LIFE). Alla fine di gennaio del 1990, la Matsushita Electric Industrial Co. diede il nome di “Asai-go” (moglie adorata) Day Fuzzy alla sua nuova lavatrice a controllo automatico, e lanciò una campagna pubblicitaria in grande stile per il prodotto fuzzy. Tale campagna si è rivelata essere un successo commerciale non solo per il prodotto, ma anche per la tecnologia stessa. Il termine d'origine estera “fuzzy” fu introdotto nella lingua giapponese con un nuovo e diverso significato: “*intelligente*”. Molte altre aziende elettroniche seguirono le orme della Panasonic e lanciarono sul mercato elettrodomestici progettati con la fuzzy logic. E esplose una vera mania per tutto quanto era etichettato come fuzzy: tutti i consumatori giapponesi impararono a conoscere la parola fuzzy, che vinse il premio per il neologismo dell'anno nel 1990. I successi giapponesi stimolarono un vasto e serio interesse per questa tecnologia in Corea, in Europa e, in misura minore, negli Stati Uniti, dove pure la logica fuzzy aveva visto la luce. La logica fuzzy ha trovato parimenti applicazione in campo finanziario. Il primo sistema per le compravendite azionarie ad usare la logica fuzzy è stato lo Yamaichi Fuzzy Fund. Esso viene usato in 65 aziende e tratta la maggioranza dei titoli quotati dell'indice Nikkei e consiste approssimativamente in 800 regole. Tali regole sono determinate con cadenza mensile da un gruppo di esperti e, se necessario, modificate da analisti finanziari di provata esperienza. Il sistema è stato testato per un periodo di due anni, e le sue prestazioni in termini di rendimento hanno superato l'indice Nikkei Average di oltre il 20%. Il sistema è divenuto operativo nel 1988. Il primo chip VLSI (Very Large Scale Integration) dedicato alla computazione d'inferenze fuzzy fu sviluppato da M. Togai e H. Watanabe nel 1986: chip di tal genere sono in grado di migliorare le prestazioni dei sistemi fuzzy per tutte le applicazioni in tempo reale. Diverse imprese sono state costituite allo scopo di commercializzare strumenti hardware e software per lo sviluppo di sistemi a logica fuzzy. Allo stesso tempo, anche i produttori di software, nel campo della teoria convenzionale

del controllo, cominciarono ad introdurre pacchetti supplementari di progettazione dei sistemi fuzzy.

2.5 Principi della logica fuzzy

I modelli per il trattamento di dati imperfetti possono essere suddivisi in modelli di tipo simbolico-qualitativo e in modelli di tipo numerico-quantitativo. La logica fuzzy è una teoria che permette di identificare un modello a metà strada tra quelli qualitativi e quelli quantitativi. Molte volte gli studi teorici effettuati non garantiscono un riscontro esatto nella pratica a causa delle numerose forme di imperfezione dei dati in esame inoltre risulta essere inconveniente una ricerca assidua per aumentare la precisione laddove non ve ne sia bisogno; al contrario l'approccio migliore consiste nel descrivere un modello che a partire da informazioni imprecise, incerte o errate sia in grado di manipolare i dati giungendo ad una soluzione coerente. È sempre meglio ottenere dei risultati caratterizzati da un grado di approssimazione, coerenti e in tempi rapidi piuttosto che ottenere dei dati che all'apparenza sembrerebbero precisi ma che si rivelano essere incoerenti e di conseguenza totalmente inutilizzabili. La teoria fuzzy permette di approssimare il pensiero umano riguardo alcuni aspetti in cui la logica tradizionale si è rivelata impotente, spesso per mancanza di espressività, e ha costruito un ponte di comunicazione tra i sistemi di modellazione quantitativi e i sistemi di modellazione qualitativi permettendo nel frattempo di esprimere concetti vaghi in modo abbastanza rigoroso. Un ambito di applicazione molto importante è quello dei controllori fuzzy, nei quali la capacità di descrivere in modo efficace concetti vaghi rende la logica fuzzy un utile strumento per gestire situazioni altrimenti impossibili da definire a causa della loro complessità. Il controllo fuzzy (fuzzy control in inglese) è l'applicazione della logica fuzzy al controllo di un sistema, sia un impianto, un agente artificiale ecc. Le entità che hanno a che fare con un controllo sono generalmente gli ingressi (i sensori, gli input) e le azioni che possono essere intraprese in risposta a tali ingressi. Nel controllo fuzzy gli ingressi sono input

fuzzy, cioè non precisi. Dunque non avremo la temperatura è 15C, bensì “fa caldo”, “fa molto caldo”, ecc. Tali input vengono processati usando le regole fuzzy (chiamate in inglese fuzzy rules, che possono essere descritte mediante un linguaggio naturale, ad esempio: SE fa caldo, ALLORA abbassa la temperatura”), dopodichè il risultato di tutto il processo viene “defuzzicato” (cioè reso di nuovo un valore preciso) e l'output utilizzato per il controllo (ad esempio, appunto, abbassare la temperatura). Come già accennato la nascita di una teoria di tipo fuzzy si è resa necessaria per far fronte ad alcuni problemi per i quali i modelli matematici non riuscivano ad elaborare risultati sufficientemente accurati. Ciò può avvenire per due motivi: le assunzioni fatte dal modello non rispecchiano completamente la realtà e quindi il risultato prodotto è affetto da problemi di approssimazione, oppure il modello matematico è talmente complesso da risultare inutilizzabile. La logica fuzzy è nata per formalizzare concetti espressi in linguaggio naturale e che non possono essere oggettivamente riconosciuti come veri o falsi, ma rispecchiano un certo grado di verità. Tutta la teoria si basa sulla definizione del concetto di Fuzzy Set introdotto da Zadeh. La peculiarità risiede nella possibilità di un oggetto di far parte di più insiemi contemporaneamente, specificando però per ogni insieme il grado di appartenenza dell'oggetto. È ovvio come un modello basato su questa idea sia ideale per gestire le informazioni vaghe e incerte espresse in linguaggio naturale con annessa proprietà. La teoria classica afferma che se un oggetto appartiene all'insieme A, questo non può appartenere anche all'oggetto B. Risulta immediato sapere se un oggetto appartiene o meno ad un insieme dato che è sufficiente gestire una variabile che vale 1 se l'oggetto è contenuto nell'insieme, mentre vale 0 se tale oggetto appartiene all'insieme complemento. I problemi dell'insiemistica classica sorgono quando si devono descrivere concetti che non sono ben definiti, quando cioè si entra nel mondo dell'imperfezione. Se si considera ad esempio il problema di stabilire se una persona è giovane o meno già si incontrano diverse difficoltà. Innanzitutto si definisce una soglia oltre la quale una persona viene considerata non giovane, ma cosa dire di una persona la cui età è vicina alla soglia stabilita? Un sistema basato su

insiemi di tipo crisp definirebbe giovane una persona di 29 anni e di mezza età una persona di 31 anni, fatto che chiaramente un umano non farebbe vista la poca differenza tra i due individui. Siccome lo scopo di un sistema del genere dovrebbe essere quello di emulare il pensiero umano è ovvio come tale approccio non sia consistente. La teoria fuzzy cerca proprio di emulare i meccanismi del ragionamento umano, utilizzando un numero limitato di regole rendendo il problema gestibile al calcolatore. Le regole si basano su variabili multi-valore, non binarie, ed è per questo motivo che è stato introdotto il concetto di **funzione di appartenenza**. Un oggetto può appartenere ad un insieme A con un certo grado di appartenenza e contemporaneamente all'insieme B con un altro grado di appartenenza.

Funzione di appartenenza: $\mu_A : X \rightarrow [0,1]$

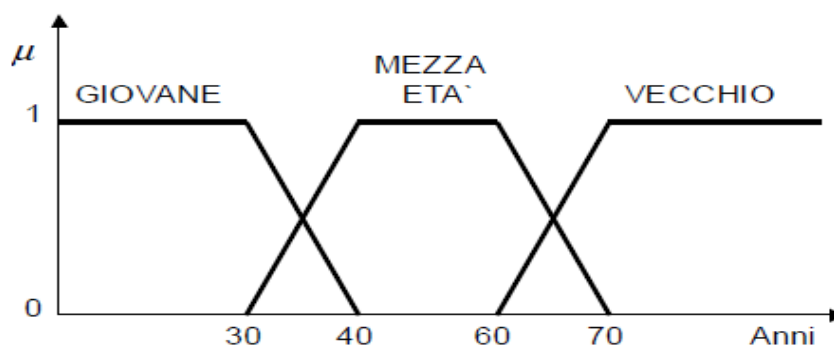


Figura2.1. Esempio di funzione d'appartenenza

2.6 Fuzzy Set (Insiemi fuzzy)

Un fuzzy set (insieme fuzzy) è un insieme senza un confine chiaramente delineato [3]. Esso contiene elementi con un grado di appartenenza compreso in $[0,1]$. Una *funzione di appartenenza* (membership function) è una funzione che assegna ad ogni valore del parametro in ingresso un valore di appartenenza (o grado di appartenenza) compreso tra 0 e 1. Quindi più formalmente un insieme fuzzy è definito nel seguente modo :

- $A = \{(x, \mu_A(x)) \mid x \in X\}$

dove $\mu_A(x)$ è la funzione di appartenenza di x in A , e X è il dominio di interesse.

Proprietà dei fuzzy set

- Dati due fuzzy set A e B , questi sono uguali se le loro funzioni di appartenenza sono uguali

$$A = B \Leftrightarrow \mu_A(x) = \mu_B(x), \quad \forall x \in X;$$

- Il set universale U è definito come:

$$\mu_U(x) = 1, \quad \forall x \in X;$$

- L'altezza di un fuzzy set A è il massimo valore che un qualche elemento assume:

$$hgt(A) = \max_{x \in X}(\mu_A(x))$$

- Se l'altezza è 1, il set è chiamato normale o normalizzato, altrimenti è chiamato subnormale. Un'insieme fuzzy può essere sempre normalizzato dividendo per $\sup \mu_A(x)$;
- Il supporto di un fuzzy set A è costituito dall'insieme crisp che contiene tutti gli elementi di X che hanno un grado di appartenenza non nullo:

$$supp(A) = \{x \in X \mid \mu_A(x) > 0\};$$

- Il nucleo di un fuzzy set normale è l'insieme crisp che contiene tutti gli elementi di X che hanno grado di appartenenza 1 in A :

$$\text{core}(A) = \{x \in X \mid \mu_A(x) = 1\};$$

- Il confine è l'insieme crisp che contiene tutti gli elementi di X che hanno grado di appartenenza compreso tra 0 e 1:

$$\text{bnd}(A) = \{x \in X \mid 0 < \mu_A(x) < 1\};$$

- Due fuzzy set sono simili, se:

$$\text{core}(A) = \text{core}(B) \quad \text{AND} \quad \text{supp}(A) = \text{supp}(B);$$

- Se il supporto di un fuzzy set A consiste in un unico elemento di valore 1, il set prende il nome di set singolare o singleton:

$$\text{supp}(A) = \text{core}(A) = \{x_0\};$$

- Un fuzzy set A è più specifico di un fuzzy set B se:

$$\text{core}(A) \subset \text{core}(B) \vee \text{supp}(A) \subseteq \text{supp}(B);$$

- Un fuzzy set A è più preciso di un fuzzy set B se:

$$\text{core}(A) = \text{core}(B) \vee \text{supp}(A) \subset \text{supp}(B);$$

- La generalizzazione del concetto di supporto è dato dall' α -cut di un fuzzy set A, costituito da tutti gli elementi dell'insieme con grado di appartenenza maggiore di un fissato valore reale :

$$\alpha - \text{cut}(A) = \{x : \mu_A(x) > \alpha\}.$$

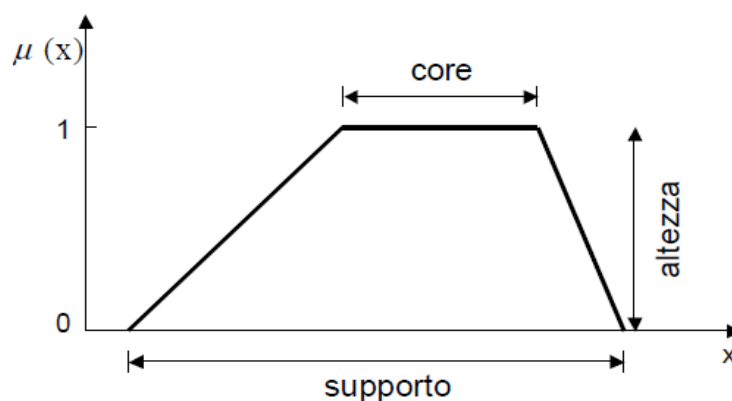


Figura 2.2 Supporto di un Fuzzy Set

Operazioni sui fuzzy set

Gli operatori principali della logica tradizionale sono l'intersezione, l'unione, il complemento e l'inclusione. Nella logica crisp esiste solo una definizione di questi operatori. Nella logica fuzzy invece esistono più possibilità di calcolo del risultato. Una prima definizione degli operatori fa uso degli operatori max e min, altre utilizzano la definizione algebrica, che sfrutta appunto le operazioni algebriche di somma e prodotto:

1. L'operatore di intersezione(AND) applicato a due fuzzy set A e B con funzioni di appartenenza:

$$\mu_{A \cap B}(x) = \min_{x \in X} \left\{ \mu_A(x), \mu_B(x) \right\};$$

$$\mu_{A \cap B}(x) = \mu_A(x)\mu_B(x);$$

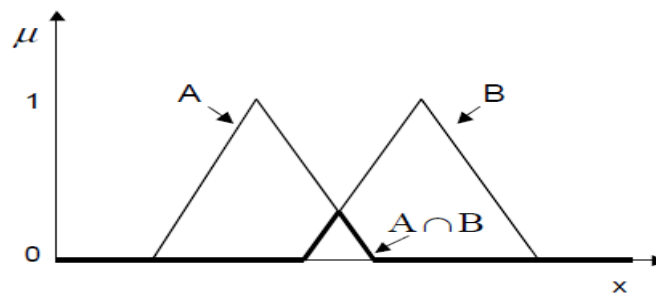


Figura 2.3. Intersezione tra due Fuzzy set

2. L'operatore di unione(OR) applicato a due fuzzy set A e B con funzioni di appartenenza:

$$\mu_{A \cup B}(x) = \max_{x \in X} \left\{ \mu_A(x), \mu_B(x) \right\};$$

$$\mu_{A \cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x);$$

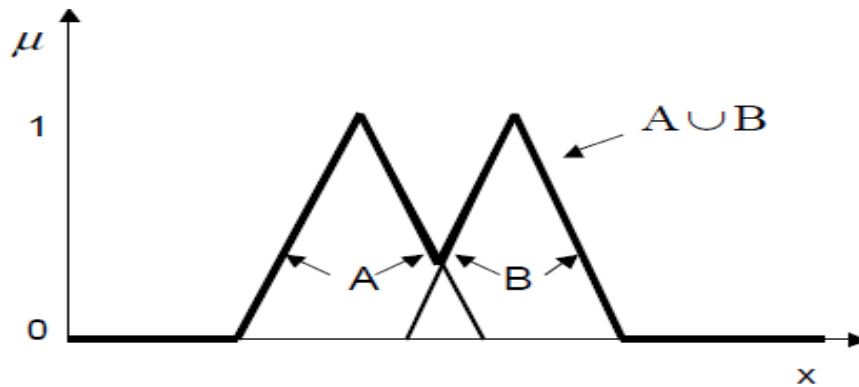


Figura 2.4 Unione di due Fuzzy set

3. L'operatore complemento (NOT) applicato ad un fuzzy set A con funzione di appartenenza:

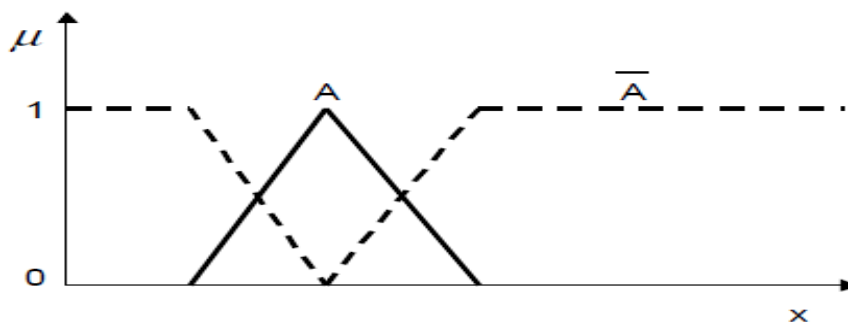


Figura 2.5 Complementazione nei Fuzzy set

4. Esiste anche l'inclusione di insiemi fuzzy
 $A \subset B$ se e solo se $\mu_A(x) \leq \mu_B(x)$

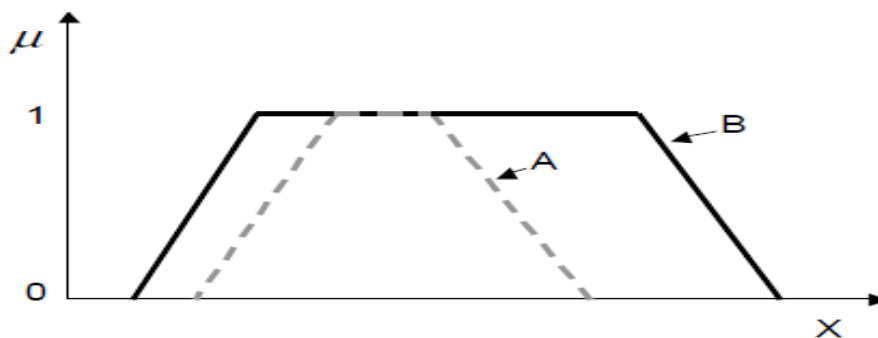


Figura 2.6 Inclusione nei Fuzzy set

Vista la diversità di definizione non tutte le proprietà insiemistiche sono valide, infatti mentre con l'uso di max e min sono valide le proprietà di commutazione, associazione, distribuzione e le leggi di De Morgan, con la definizione algebrica non è più valida la proprietà distributiva (mentre le altre continuano a valere).

Relazioni fuzzy

Dati n fuzzy set il loro *prodotto cartesiano* è dato dalla seguente formula:

$$A = A_1 \times \dots \times A_n = \left\{ (x, \mu_A(x)) : x \in X_1 \times \dots \times X_n \right\}$$

Se un elemento non appartiene ad uno degli insiemi il suo grado di appartenenza all'insieme prodotto sarà nullo. Una *relazione fuzzy* è un sottoinsieme degli elementi del prodotto cartesiano e quindi è un insieme fuzzy dato dalla seguente formula:

$$R = \left\{ ((x, y), \mu_R(x, y)) : (x, y) \in X \times Y \right\}$$

A differenza della classica logica booleana nella fuzzy logic tanto più alta è la funzione di appartenenza della coppia (x,y) tanto più è vero che esiste il legame tra x e y espresso da R.

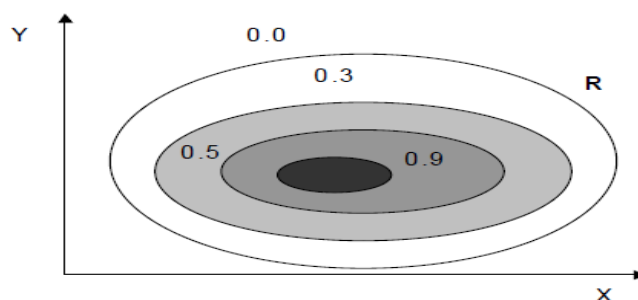


Figura 2.7 Relazioni Fuzzy

Regole Fuzzy

Una regola è concepita come una regola di produzione, cioè contenente una coppia condizione-azione nella forma *If* condizione *then* azione, e ogni regola è un'entità a tre valori. Ciascun caso dovrebbe essere codificato tramite l'utilizzo di diversi tipi di valori di verità. I primi due casi corrispondono agli usuali valori di verità *true* = *vero* e *false* = *falso*; il terzo caso corrisponde al terzo valore di verità che può essere interpretato come *unknown* = *sconosciuto*. Le regole *If-Then* vengono utilizzate per formulare le dichiarazioni condizionali che contengono la logica fuzzy. La parte *if* della regola "x is A" è detta *antecedente* o *premessa*, mentre la parte *then* della regola "y is B" è detta *conseguente* o *conclusione*. Generalmente, l'input per una regola *if-then* è il valore corrente per la variabile di input e l'output è un intero fuzzy set. Questo insieme verrà poi defuzzificato, assegnando un valore all'output. Nel caso della logica a due valori l'interpretazione di una regola if-then non presenta particolari difficoltà: se la premessa è vera allora la conclusione è vera. Nel caso in cui invece si operi nell'ambito della logica fuzzy se l'antecedente è vero con un certo grado di appartenenza, allora il conseguente è altrettanto vero con lo stesso grado. È possibile trovarsi di fronte a regole costituite da antecedenti multipli: in tal caso le parti dell'antecedente sono calcolate simultaneamente e associate ad un singolo valore utilizzando gli operatori logici. Analogamente anche il conseguente di una regola può avere parti multiple, in tal caso tutti i conseguenti sono equamente influenzati dal risultato dell'antecedente. L'interpretazione delle regole *if-then* è un processo che consta di tre parti:

1. **Fuzzificazione degli input** : si associa a tutte le affermazioni fuzzy nell'antecedente un grado di appartenenza compreso tra 0 e 1. Questo è il cosiddetto grado di supporto per la regola;
2. **Applicazione degli operatori fuzzy ad antecedenti con parti multiple**: si applicano gli operatori fuzzy e si associa così l'antecedente ad un singolo numero compreso tra 0 e 1;

3. **Applicazione del metodo di implicazione** : se l'antecedente è solo parzialmente vero, allora il fuzzy set di output è troncato secondo il metodo di implicazione prescelto.

2.7 Concetti temporali

Ora verranno analizzati i concetti relativi alla rappresentazione del tempo caratterizzato da incertezza che stanno alla base dell'implementazione effettuata in YAWL. Inoltre verranno analizzate le situazioni in cui un istante temporale o un intervallo temporale è affetto da una qualche forma di imperfezione, che si manifesta sotto forma di vaghezza e incertezza, oppure con istanti temporali definiti su diversi livelli di granularità.

Dominio temporale

Un dominio temporale è una coppia $(T; \leq)$ dove T è un insieme non vuoto di istanti temporali e " \leq " è un ordinamento universale su T . Il dominio temporale è l'insieme delle entità primitive temporali utilizzate per definire e interpretare i concetti relativi al tempo. L'insieme viene ordinato attraverso una relazione, " \leq ", su tali entità [10]. Si può scegliere tra due tipi di dominio:

1. Un dominio si definisce *denso* se è un insieme infinito e $\forall t, t' \in T$ con $t < t'$, esiste sempre un $t'' \in T$ tale che $t < t'' < t'$
2. Un dominio è *discreto* se ciascun elemento ad eccezione dell'ultimo possiede un successore immediato, e tutti gli elementi ad eccezione del primo possiedono un predecessore immediato.

Quindi se l'applicazione impone una granularità minima prefissata ne consegue che il modello di dominio più appropriato risulta essere un dominio discreto. Se, al contrario, all'utente interessa rappresentare arbitrariamente

granularità più fini, allora è richiesto un dominio denso. Inoltre un dominio temporale è *limitato* se prevede un limite superiore e/o inferiore nel rispetto della relazione interna di ordinamento. Formalmente (nel caso di presenza di entrambi i limiti), il dominio temporale T è limitato se esistono $t', t'' \in T$ tali che $t' \leq t \leq t'' \forall t \in T$, con t' limite inferiore e t'' limite superiore.

Granularità

Una granularità è una mappatura G dagli interi (insieme di indici) ai sottoinsiemi del dominio temporale tale che:

1. se $i < j$, $G(i)$ e $G(j)$ sono non vuoti, allora ciascun elemento di $G(i)$ è minore di tutti gli elementi di $G(j)$;
2. se $i < k < j$, $G(i)$ e $G(j)$ sono non vuoti, allora $G(k)$ è non vuoto.

La prima condizione stabilisce che i granuli relativi ad una granularità non si sovrappongono e che il relativo ordinamento degli indici è lo stesso rispetto all'ordinamento del dominio temporale. La seconda condizione dice che il sottoinsieme dell'insieme degli indici che mappa verso sottoinsiemi non vuoti del dominio temporale è contiguo. Mentre il dominio temporale può essere discreto, denso o continuo, una granularità semplicemente definisce l'insieme dei granuli; ciascun granulo viene identificato da un intero. Una granularità si dice *limitata* se esistono un limite inferiore ed un limite superiore k' e k'' nell'insieme indice tali che $G(i) = \emptyset ; \forall i$ con $i < k'$ o $k'' < i$. Ciascun sottoinsieme non vuoto $G(i)$ viene chiamato **granulo** della granularità G . Intuitivamente, un granulo è un insieme di istanti temporali che viene percepito come atomico. Un granulo può essere costituito da un istante singolo, un insieme di istanti contigui (intervallo temporale), o ancora da un insieme di istanti non contigui. Due granuli $G(i)$ e $G(j)$ sono *contigui* se non esiste un $t \in T : G(i) < t < G(i + 1)$. Un elemento importante è l'**origine** di una granularità G che consiste in un granulo creato per identificare l'origine rappresentato da $G(0)$. Un altro elemento fondamentale è l'**anchor** di G che rappresenta il più grande limite inferiore dell'insieme degli elementi

appartenenti al dominio temporale corrispondente all'origine di G . In pratica l'anchor è l'elemento più remoto che si avvicina maggiormente all'origine. Ad esempio una granularità ancorata è “7 Giugno 1985”, una non ancorata è “1 ora”. Esistono inoltre delle *relazioni* definite tra granularità con lo stesso dominio temporale:

- **Raggruppamento.** Una granularità G si raggruppa in una granularità H , se per ogni indice j esiste un sottoinsieme (eventualmente infinito) S di interi tale che $H(j) = \bigcup_{i \in S} G(i)$.
- **Più fine di.** Una granularità G è più fine di una granularità H , se per ogni indice i , esiste un indice j tale che $G(i) \subseteq H(j)$.
- **Sotto-granularità.** Una granularità G è una sotto-granularità di H se per ogni indice i , esiste un indice j tale che $G(i) = H(j)$;
- **Shift-equivalente.** Le granularità G e H sono shift-equivalenti, se esiste un intero k tale che $G(i) = H(i + k) \forall i$ nell'insieme degli indici.

2.8 Modellazione dell'informazione temporale

Il modello proposto si basa sulla struttura descritta da Bettini e colleghi per la gestione delle granularità temporali che può essere efficientemente incorporata con la rappresentazione della conoscenza temporale (fuzzy).

Dominio temporale e granularità

Il dominio temporale fondamentale \mathbf{T} , chiamato anche asse temporale, è isomorfo rispetto ai numeri reali con la classica relazione di ordinamento “ \leq ”. L'insieme *Gran* delle funzioni di granularità equivale alle granularità del Calendario Gregoriano (anni, mesi, giorni, ore, minuti e secondi). La mappatura considera entrambe le tipologie di granularità, ancorate e non

ancorate. Ciascuna funzione di granularità G è una funzione che mappa valori appartenenti ad un insieme di indici I_G verso l'insieme costituito da 2^T elementi [10]. Più precisamente:

$$Gran = \{Y, mean_Y, M, mean_M, D, mean_D, H, mean_H, Mi, mean_Mi, S, mean_S\}.$$

Le funzioni Y , M , etc.. rappresentano le classiche granularità basate sul Calendario Gregoriano (devono gestire anni bisestili, mesi da 28, 29, 30 o 31 giorni e così via). Le funzioni $mean_Y$, $mean_M$, ecc.. forniscono invece una mappatura regolare, che si presta molto meglio ad essere utilizzata nelle operazioni di modellazione di durate. Al posto di utilizzare la notazione $\mu(i)$ per identificare l'*i-esimo tick* della mappatura, viene utilizzato il simbolo " $< >$ ". Altre notazioni vengono utilizzate per i singoli insiemi di indici, e.g. YY/MM/DD/HH/Mi/SS per l'insieme dei secondi I_S , YY/MM/DD per l'insieme I_D , n_1 y n_2 m per l'insieme I_{mean_M} delle durate espresse utilizzando mesi e anni. Alcuni esempi per chiarire meglio il concetto:

- $Y(1996) = < 1996 >$
- $M(1996 * 12 + 4) = < 1996/4 >$
- $mean_Y(1) = < 1y >$
- $mean_M(1 * 12 + 3) = < 1y3m >$

Le funzioni I e $u: 2^T \rightarrow T$, restituiscono, rispettivamente, il limite inferiore ed il limite superiore del granulo considerato. La notazione $granule(I_G, t)$, dove I_G è uno degli indici temporali descritti sopra, è un troncamento che fa riferimento al granulo $G(i)$ su T contenente l'istante t : ad esempio, $granule(I_M, I < 1997/1/13/0/0/0 >) = < 1997/1 >$.

Istanti e durate

Un istante temporale fa riferimento ad un punto sull'asse temporale, la cui collocazione può essere caratterizzata da una componente di incertezza. Gli istanti vengono rappresentati dal dominio **Inst** [10]. La posizione di un

istante sull'asse viene fornita da una funzione chiamata *distribuzione di possibilità* $\pi : \text{Inst} \times \mathbf{T} \rightarrow [0; 1] \subset \mathbb{R}$. $\forall t \in \pi_i(t)$ è il valore numerico che stima la possibilità che l'istante i sia precisamente t . Quando $\pi_i(t) = 0$, è certo che i e t differiscono tra loro. A partire dalla funzione “ π ” sono state derivate due ulteriori funzioni di possibilità chiamate “*poss_after_*” e “*poss_before_*”:

- $\text{poss_after_}\pi_i(t) = \sup_{s \leq t} (s)$
- $\text{poss_before_}\pi_i(t) = \sup_{s \geq t} (s)$

Le ultime due funzioni, dato un'istante i , forniscono un valore numerico stimante la possibilità che un punto nel tempo sia dopo (prima) l'istante i . Una *durata* è un intervallo temporale non-ancorato: rappresenta la distanza tra due punti nel tempo. Come accade per gli istanti, le durate possono essere fornite con vaghezza, indeterminatezza o utilizzando differenti granularità. Formalmente, le durate vengono rappresentate dal dominio **Dur**. La distribuzione di possibilità in questo caso stima la possibilità che una durata sia rappresentata da differenti distanze tra punti nel tempo. La funzione $\pi : \text{dur} \times \mathbf{T} \rightarrow [0; 1] \subset \mathbb{R}$ restituisce un valore rappresentante la possibilità che differenti punti nel tempo su \mathbf{T} rappresentino una certa durata.

Istanti e durate con diverse granularità

Gli istanti e le durate vengono presentati attraverso un formato standard utilizzato per rappresentare le date e gli archi temporali come YY/MM/DD, YY/MM/DD/HH, o YY/MM/DD/HH/Mi/SS per gli istanti, e N_1 yy N_2 mm, N_1 yy N_2 mm N_3 dd, per le durate, dove N_i è il numero di unità temporali considerate. In modo più formale si può dire che tali costanti corrispondono all'applicazione della funzione **g2i** : $2^T \rightarrow \text{Inst}$ [10]; l'argomento di questa funzione sono i granuli ottenuti dalle granularità prese dal calendario. Ad esempio, la costante 2008/11/19 è il risultato dell'applicazione della funzione **g2i** (<2008=11=19 >). Allo stesso modo è possibile definire la funzione inversa **i2g** la quale, dato un istante appartenente a **Gran_i**, ritorna il corrispondente granulo. Bisogna ora definire una distribuzione di possibilità verosimile ed efficace in grado di rappresentare gli istanti e le durate forniti

con diverse granularità. La funzione trapezoidale proposta e analizzata è la seguente:

- $\forall i \in \mathbf{Gran_I} \pi_i \equiv_{df} (l(i2g(i))-f(i); l(i2g(i)),u(i2g(i)),u(i2g(i)) + f'(i)).$

A questo punto sono diverse le definizioni possibili che si possono dare per le funzioni “ f ” ed “ f ’ ” ; l'idea di base è che queste due funzioni devono dipendere dall'ampiezza del corrispondente granulo, in modo da ottenere una fuzzificazione dipendente dallo spessore della granularità corrispondente. Una possibile scelta è la seguente facendo in modo che f ed f ' coincidano:

- $\forall i \in \mathbf{Gran_i} f(i) \equiv_{df} [u(i2g(i)) - l(i2g(i))] / k(i)$

Per chiarire al meglio il concetto è necessario fornire un esempio di definizione: si consideri l'insieme degli indici ID dei giorni. Ponendo $l(i2g(YY/MM/DD)) = x$ e $u(i2g(YY/MM/DD)) = y$, la distribuzione di possibilità associata agli istanti definiti su quest'indice è:

- $\pi_{YY/MM/DD} \equiv_{df} (x-(86400/12),x,y,y+(86400/12)$

dove 86400 è il numero di secondi nel granulo dei giorni e 12 è un valore scelto in modo verosimile per rappresentare la costante k(i), per ogni i. La figura seguente illustra le funzioni di possibilità associate all'istante temporale 10:00:00 a.m. del 15 Novembre 2008 (2008/11/15/10/0/0, alla granularità dei secondi), 10:00 a.m. del 15 Novembre 2008 (2008/1/15/10/0, alla granularità dei minuti) e 10 a.m. del 15 Novembre 2008 (2008=11=15=10, alla granularità delle ore).

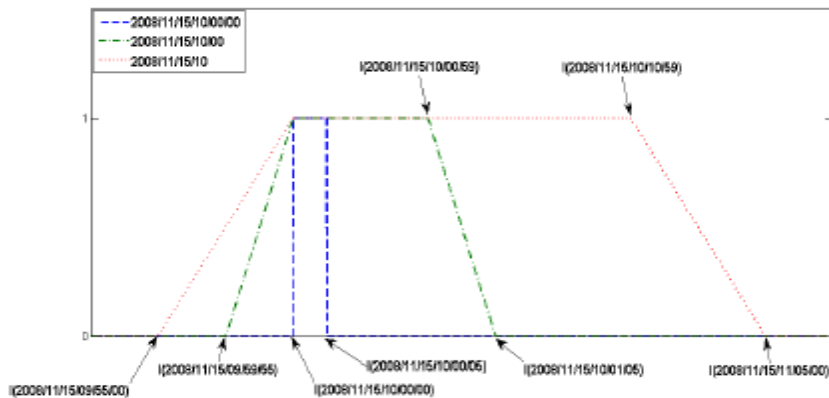


Figura 2.8 Esempio di trapezio di un vincolo

É semplice notare come la funzione di possibilità associata ad un istante definito alla granularità dei secondi (la granularità più fine del modello) non rappresenta alcuna fuzziness in quanto non avrebbe senso e non sarebbe coerente con le definizioni fornite finora; perciò i punti all'interno del granulo definito hanno possibilità pari a 1, mentre i punti al di fuori del granulo hanno possibilità 0 in quanto non possono in alcun modo rappresentare l'istante specificato. Un discorso analogo può essere fatto per quanto riguardano le durate. In questo caso la distribuzione trapezoidale definita formalmente è la seguente:

- $\forall i \in \mathbf{Gran_I} \pi_i \equiv_{df} (x - f(i), x, y, y + f'(i))$.

Fino a questo punto non ci sono differenze rispetto al modello definito per gli istanti. Le differenze emergono nel momento in cui bisogna definire le funzioni $f(i)$ ed $f'(i)$. Nella costruzione della funzione bisogna tenere conto del fatto che non si parla più di istanti assoluti che individuano un punto sull'asse temporale ma di punti relativi, frutto della distanza tra due istanti. Per questo motivo è opportuno che la funzione tenga presente dell'estensione della durata per calcolare il massimo spostamento dovuto al processo di fuzzificazione: più l'intervallo è ampio e più esso dovrebbe risentire della componente incertezza, se presente. “ $f(i)$ ” deve contenere al suo interno una

componente che dipende dall'estensione della durata sia in termini di granularità che in termini di valore numerico. Ecco la funzione “f(i)” proposta:

- $\forall i \in \mathbf{Gran_I} \ f(i) \equiv_{df} x / k(i)$

La funzione “f'(i)”:

- $\forall i \in \mathbf{Gran_I} \ f'(i) \equiv_{df} y / k(i)$

Grazie alle ultime definizioni è possibile semplificare la notazione della distribuzione trapezoidale relativa alle durate:

- $\forall i \in \mathbf{Gran_I} \ \pi_i \equiv_{df} ([(k(i) - 1) / k(i)] * x , x, y, y * [(k(i) + 1) / k(i)])$.

Le osservazioni relative alla granularità dei secondi valgono anche per le durate.

Istanti caratterizzati da vaghezza

Si consideri ora il secondo caso in cui si definiscono istanti con un certo grado di vaghezza [10]. Una scelta semplice è quella di adottare funzioni che permettono di ottenere una partizione fuzzy del granulo considerato: una partizione fuzzy per un granulo X è composta dalle funzioni di possibilità π_i tali che:

- $\forall t \in X \ \sum_i \pi_i(t) = 1$

Si considerino le seguenti costanti, che identificano un istante all'inizio, in mezzo o alla fine del Novembre 2008: *beginning(2008/11)*, *middle(2008/11)* e *end(2008/11)*. Le seguenti funzioni di possibilità possono aiutare a definire tali istanti ponendo $I(< 2008/11 >) = x$ e $I(h2008=11i) = y$:

- $\pi_{beginning(2008/11)} \equiv_{df} (x , x , x+a , x+2a)$
- $\pi_{middle(2008/11)} \equiv_{df} (x+a , x+2a , x+3a , x+4a)$
- $\pi_{end(2008/11)} \equiv_{df} (x+3a , x+4a, y , y)$

con $a = [u(i2g(2008/11)) - l(i2g(2008/11))] / 5$.

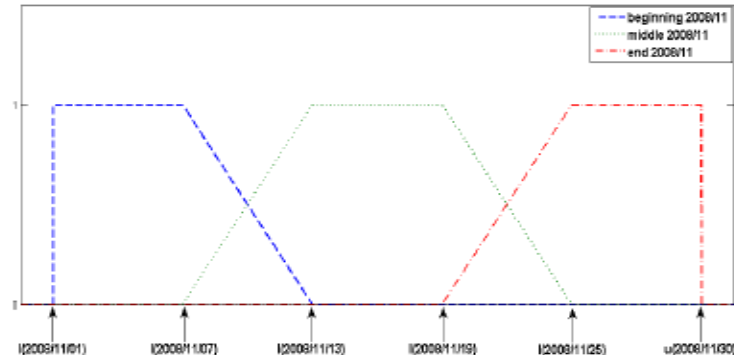


Figura 2.9 Esempio di vaghezza

Quindi in generale per gli istanti appartenenti a **Vag_I** si può scrivere:

- $\pi_{beginning(i)} \equiv_{df} (L, L, L + (U-L) / 5, L + [2 * (U-L) / 5])$
- $\pi_{middle(i)} \equiv_{df} (L + (U-L) / 5, L + [2 * (U-L) / 5], L + [3 * (U-L) / 5], L + [4 * (U-L) / 5])$
- $\pi_{end(i)} \equiv_{df} (L + [3 * (U-L) / 5], L + [4 * (U-L) / 5], U, U)$

con $L = l(i2g(i))$ e $U = u(i2g(i))$

Istanti caratterizzati da indeterminatezza

Il terzo e ultimo caso concerne la definizione di istanti caratterizzati dalla componente di indeterminatezza [10]. In questo caso le funzioni di possibilità possono essere ottenute combinando le funzioni che caratterizzano gli istanti sia che essi siano definiti secondo una determinata granularità, sia che essi siano definiti con la componente vaghezza. Più precisamente, ciò che viene fatto è combinare le funzioni di possibilità dei due istanti utilizzati come confine grazie al classico operatore di intersezione tra funzioni di possibilità:

- $\forall i, j \in \text{Gran_I} \cup \text{Vag_I} \pi_{in_in(i,j)} \equiv_{df} \text{poss_after_}\pi_i \cap \text{poss_before_}\pi_j$

Si consideri l'istante $\langle \text{end}(2008/11) \ 2009/01/15 \rangle$, che rappresenta un punto nel tempo situato tra la fine del Novembre 2008 e il 15 Gennaio 2009.

La funzione di possibilità corrispondente può essere ottenuta come l'intersezione tra le funzioni $\text{end}(2008/11)$ e $2009/01/15$, come mostrato in figura:

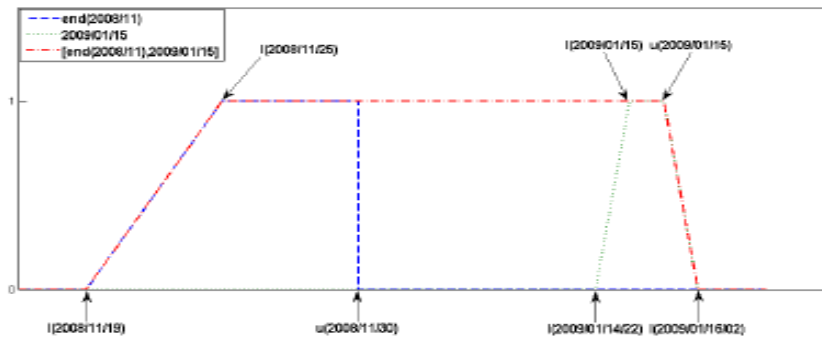


Figura 2.10 Indeterminatezza in un vincolo

Esiste un'ulteriore condizione che dev'essere rispettata in caso di presenza della componente indeterminatezza:

- $\inf (t : \pi_i(t) = 1) < \inf (u : \pi_j(u) = 1)$

Tale condizione semplicemente controlla che il primo istante non sia successivo al secondo in modo tale che la creazione del trapezio si concluda con successo.

Intervalli temporali

Un intervallo temporale fa riferimento ad un insieme di punti nel tempo contigui. Un istante iniziale, un istante finale ed una durata identificano un'intervallo. **Itvl** è il dominio delle variabili e delle costanti presenti in un intervallo [10]. Per le variabili di tipo intervallo le funzioni $\text{from}()$ e $\text{to}()$ restituiscono valori appartenenti al dominio **Inst** mentre la funzione $\text{dur}()$ restituisce un risultato nel dominio **Dur**. Tali funzioni prendono il nome in letteratura di costruttori di intervalli utilizzando la notazione “< >” per esprimere intervalli attraverso diverse espressioni. Alcuni esempi possono essere:

1. $\langle YY \rangle$, $\langle YY/MM \rangle$ o ancora $\langle YY/MM/DD \rangle$ quando l'intervallo è un granulo appartenente al Calendario; questa notazione viene utilizzata per modellare intervalli definiti da frasi come l'anno 2007, Gennaio 2006. Il costruttore è la funzione $gran\ int: 2^T \rightarrow Itvl$;
2. $\langle from(x), to(x) \rangle$ quando gli istanti iniziale e finale sono noti. Questa è la classica notazione per esprimere gli intervalli nei database temporali (con la limitazione di poter utilizzare solo una granularità e di non poter esprimere la vaghezza). Il costruttore è la funzione $from\ to: 2^{Inst} \rightarrow Itvl$;
3. $\langle from(x), dur(x) \rangle$ quando sono conosciuti l'istante iniziale e la durata. Il costruttore è $from\ for: Inst \times Dur \rightarrow Itvl$.
4. $\langle dur(x), to(x) \rangle$, in questo caso sono noti l'istante finale e la durata. Il costruttore è $for\ to: Dur \times Inst \rightarrow Itvl$;
5. $\langle in, dur(x) \rangle$ dove "in" è un granulo; questa notazione permette di esprimere intervalli. Il costruttore è $in\ for: 2^T \times Dur \rightarrow Itvl$.
6. $\langle from(x), dur(x), to(x) \rangle$ quando tutte le variabili sono conosciute; è il caso più generale grazie al quale è possibile esprimere tutte le notazioni precedentemente discusse. Il costruttore è $from\ for\ to: 2^{Inst} \times Dur \rightarrow Itvl$.

CAPITOLO 3

YAWL

3.1 Introduzione

YAWL [1] (Yet Another Workflow Language) è un WfMS (workflow management system) che, fornisce un linguaggio semplice ma potente per modellare processi i quali descrivono complesse relazioni di flussi tra business processes. La specifica del linguaggio ed il software sono stati sviluppati dalla collaborazione tra la Queensland University of Technology e l'Eindhoven University of Technology. Oggi YAWL è distribuito come software Open Source ed è reperibile al sito www.yawl-system.com. Yawl è nato come strumento di modellazione e analisi avanzata dei grafi di workflow, basato sui fondamenti delle Reti di Petri: poi è stato arricchito dei moduli software caratteristici dei WfMS. YAWL ora consiste di un motore open source e un editor GUI, entrambi scritti in Java e realizzati con licenza LGPL. Al suo interno contiene e utilizza XML Schema, XPath, XQuery e Xforms, ed è compatibile con SOAP e WSDL. Le caratteristiche principali di YAWL sono:

- Supporto completo dei Workflow Pattern;
- Sintassi e semantica del flusso di controllo (control-flow perspective) dotate di una definizione formale.

- Supporto per la persistenza dei dati (interfacciandosi con PostgreSQL 8), generazione automatica di form e consolle di amministrazione.
- Supporto per sofisticate estensioni basate su Service-Oriented Architecture (SOA).
- Uso della tecnologia XML.

I punti di Forza del software sono:

- **Espressivo**, supporta direttamente tutti gli schemi di workflow e gestisce in modo avanzato i workflow pattern.
- **Visuale**, rende più intuitiva la visione dei processi grazie alla loro rappresentazione a grafi
- **Flessibile**, permette la creazione di servizi da integrare nel linguaggio
- **Affidabile**, formalmente fondato sulla teoria delle reti di Petri.

La definizione di pattern per workflow mira a stabilire un approccio più strutturato per specificare le dipendenze dei flussi di controllo nei linguaggi di workflow. E' stato individuato un insieme di pattern che corrisponde alla tipica dipendenza dei flussi di controllo incontrati nelle specifiche. Inoltre le Reti di Petri, nonostante abbiano un'enorme potere espressivo, non permettono di gestire tutti i pattern individuati, tra i quali la cancellazione, l'unione sincronizzata e le attività multiple. YAWL è stato sviluppato proprio per eliminare queste carenze dei linguaggi per workflow: unisce la conoscenza ottenuta dai pattern per workflow con il potenziale delle reti di Petri.

3.2 I Pattern

Un pattern può essere definito come un modello: ne esistono 20 raggruppabili in 6 categorie e servono per valutare i vari linguaggi di workflow:

- **Pattern di base per i flussi di controllo:** Sono costrutti di base presenti in molti linguaggi di workflow per modellare l'instradamento sequenziale, parallelo e condizionale.
 1. **Sequenza:** un task nel processo è attivato dopo il completamento di un altro task nello stesso processo.
 2. **Split parallelo:** punto in cui un singolo thread si divide in più thread eseguibili in parallelo.
 3. **Sincronizzazione:** punto in cui più task convergono in uno unico di controllo, ottenendo quindi una sincronizzazione multipla.
 4. **Scelta esclusiva:** scelta di una delle strade possibili in base al verificarsi di alcune condizioni.
 5. **Merge semplice:** due o più rami si uniscono senza sincronizzazione, nessun ramo può essere eseguito in parallelo.

- **Pattern di ramificazione avanzata e sincronizzazione:** Permettono comportamenti avanzati per divisioni (split) e congiunzioni (join).
 6. **Scelta multipla:** scelta di uno o più rami da eseguire in base a decisioni prese o a dati di controllo del workflow.
 7. **Merge di sincronizzazione:** se più cammini convergono in un unico thread, e più di un ramo è attivo, è necessaria una sincronizzazione. Se invece è attivo un solo cammino i rami alternativi possono convergere di nuovo senza sincronizzazione. Inoltre un ramo che è già stato attivato non può essere riattivato prima che gli altri rami attivati nella stessa congiunzione non siano stati completati.
 8. **Merge multiplo:** due o più rami convergono senza sincronizzazione.
 9. **Discriminatore:** punto dove si aspetta che un ramo in ingresso sia completo prima di attivare il task seguente. Quando tutti i rami sono stati valutati si inizializza di nuovo il workflow in modo che questo possa essere eseguito nuovamente (utile per i cicli).

- **Pattern strutturali:**
 10. **Cicli arbitrari:** due o più task possono essere eseguiti ripetutamente.
 11. **Terminazione implicita:** un sottoprocesso termina quando non ha più nulla da fare adesso o in futuro.
- **Pattern per istanze multiple:** utilizzati quando alcune parti del processo devono essere istanziate più volte.
 12. **Istanze multiple senza sincronizzazione:** si possono creare nuovi thread indipendenti all'interno di una singola istanza senza utilizzare la sincronizzazione.
 13. **Istanze multiple con conoscenza a priori durante la progettazione:** il numero di istanze di ogni task per un data istanza di processo è conosciuto al tempo di progettazione.
 14. **Istanze multiple con conoscenza a priori a tempo di esecuzione:** il numero di istanze di un dato task variano ma è noto ad un certo punto dell'esecuzione, prima che le istanze del task vengano create.
 15. **Istanze multiple senza conoscenza a priori a tempo di esecuzione:** a differenza del pattern 14 non si conosce mai il numero di istanze per task e quindi se ne possono creare di nuove mentre è in corso l'esecuzione di altre.
- **Pattern basati sullo stato**
 16. **Scelta differita:** scelta di uno o più rami ma a differenza dello split XOR la scelta non è basata su dati o decisioni ma sono possibili più alternative. Solo una delle alternative viene eseguita e una volta che è stato attivato il ramo gli altri vengono disattivati. Inoltre la scelta viene fatta il più tardi possibile.
 17. **Interleaved Parallel Routing:** un insieme di task è eseguito in un ordine arbitrario, l'ordine è deciso a tempo di esecuzione.
 18. **Milestone:** il task viene attivato se si raggiunge un certo punto cardine.

- **Pattern di cancellazione:**

19. **Cancellazione di un'attività:** un task in esecuzione viene disabilitato.
20. **Cancellazione di un processo:** una istanza di processo viene completamente rimossa.

3.3 Editor YAWL

YAWL, come già affermato, è composto da due elementi principali: l'editor grafico e l'engine. In questo paragrafo verrà introdotto l'editor grafico e il suo utilizzo. Questo strumento ha quattro funzionalità principali:

1. Modellazione degli schemi di workflow, tramite tutti gli oggetti presenti all'interno dell'editor
2. Analisi e validazione in build-time degli schemi di workflow
3. Esportazione degli schemi in formato XML, per permettere all'engine di YAWL di elaborarli
4. Importazione di specifiche in formato XML e loro traduzione nel formato proprietario YAWL.

L'interfaccia iniziale dell'editor è mostrata in figura:

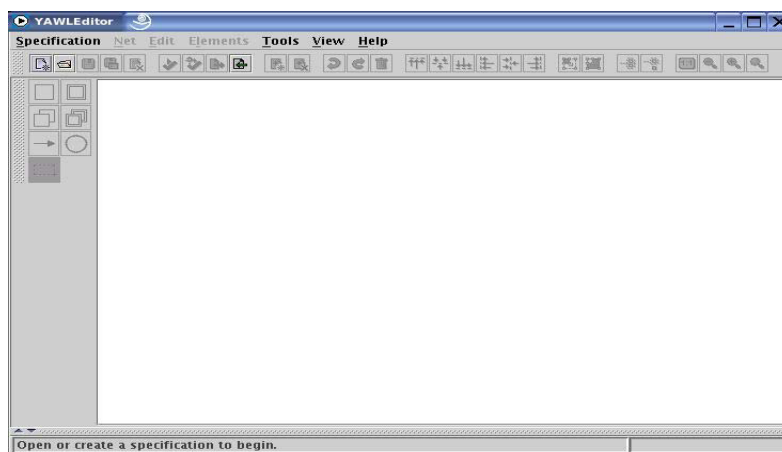


Figura 3.1. Editor Yawl

L'editor contiene la rappresentazione grafica di tutti gli elementi necessari per la costruzione di uno schema di workflow. I principali sono i seguenti:

1. **Atomic Task**, rappresenta un singolo task, può essere eseguito da un utente o da un'applicazione esterna
2. **Composite task**, contenitore di una sottorete
3. **Multiple Atomic Task**, permette di eseguire più istanze di un atomic task contemporaneamente
4. **Multiple Composite Task**, permette di eseguire più istanze di un composite task
5. **Flow relation**, crea una relazione tra i task
6. **Condition**, rappresenta uno stato nella rete
7. **Split**, permette la suddivisione del flusso di lavoro tra uno o più task a seconda del tipo scelto. Ci sono tre tipi di split: *AND* in cui il flusso di esecuzione passa attraverso tutti i task, *OR* dove il flusso di esecuzione passa attraverso almeno un task e infine lo *XOR* dove il flusso di esecuzione passa attraverso un solo task.
8. **Join**, in cui la condizione specifica il comportamento da adottare per l'esecuzione del task in relazione al completamento dei precedenti. Esistono tre tipi di join: *AND* che attiva il task quando sono stati completati tutti quelli in arrivo, *OR* che attiva il task quando sono stati completati uno o più tra quelli in arrivo (è praticamente una *AND* tra i task selezionati) ed infine il join *XOR* che attiva il task quando è stato completato uno solo tra quelli in arrivo.

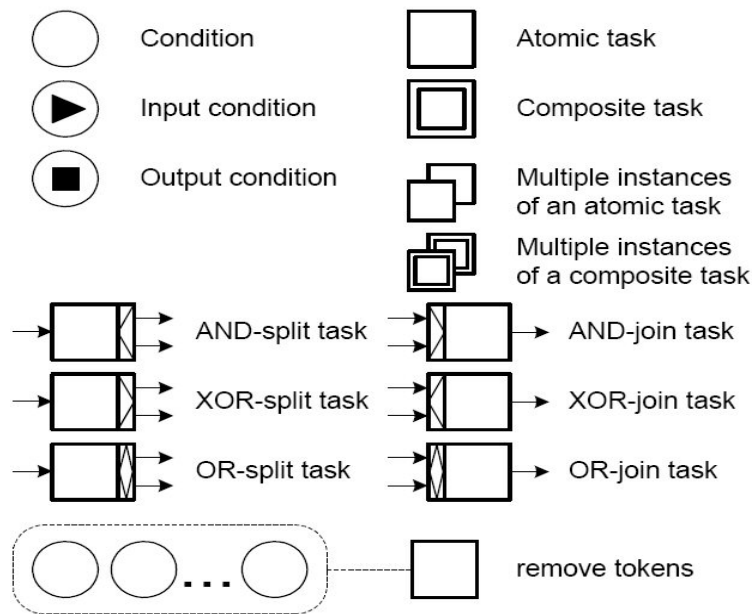






Figura 3.2 Elementi per la costruzione del workflow

Come si vede in figura i task possono essere atomici o composti, ogni task composto si riferisce a una definizione di processo a un livello più basso della gerarchia. I task atomici sono le foglie della struttura a grafo. Ogni processo è costituito da task e condizioni, ha un unico punto iniziale ed un unico punto finale che sono distinti tra loro. In YAWL sono le condizioni di input e output. Inoltre è possibile connettere due oggetti di transizione (task) direttamente senza l'utilizzo di oggetti intermedi (a differenza delle reti di Petri). Questo viene interpretato come una condizione nascosta, quindi implicita, che viene aggiunta ad ogni connessione diretta. Ogni task può avere istanze multiple di cui si specifica un limite inferiore o superiore al momento di inizializzare il task, inoltre si può decidere di terminare un task quando si raggiunge un certo numero di istanze completate. Si può decidere che il numero di istanze sia *statico*, cioè deciso all'inizializzazione del task, o *dinamico*, ossia vi è la possibilità di aggiungere ulteriori istanze mentre il task non è ancora completo. Ci sono poi i join che indicano il comportamento che viene assunto dal flusso in ingresso in un task, mentre lo split riguarda il comportamento del flusso in uscita da un task. Infine viene introdotto il task

di rimozione che elimina tutti i token nella sua area di cancellazione, grazie a questo costrutto vengono implementati i pattern di cancellazione. Riassumendo una rete di workflow di YAWL è formata da una condizione di input, una condizione di output, un insieme di task (con associata una molteplicità), un insieme di operatori di cancellazione, split e join, un insieme di condizioni e le relazioni di flusso che collegano i vari elementi in modo tale che partendo dalla condizione iniziale si arrivi a quella finale. Inoltre per ogni task esiste una “*task decomposition*” che rappresenta l'ambiente di visibilità di un task e permette di definire il tipo di attività che deve rappresentare tale task e l'uso delle variabili usate dai singoli task. Inoltre permette di scegliere quale servizio dello YAWL Engine verrà invocato dal task (ad esempio il servizio delle worklet, con worklet si intende un insieme di attività atte a svolgere un compito preciso). Nell'editor sono presenti altri pulsanti che permettono altre funzioni molto importanti:

-  • validare la specifica creata rispetto ai requisiti richiesti dallo YAWL Engine;
-  • analizzare la specifica alla ricerca di eventuali problemi o deadlock
-  • esportare il diagramma di workflow in un file xml per effettuare successivamente l'upload sullo YAWL Engine
-  • importare un file xml, specifico del motore, all'interno dell'editor per alcune modifiche.

Le proprietà del workflow che YAWL verifica sono:

- Correttezza del workflow (soundness)
 1. *Option to complete*: un processo deve poter terminare (no deadlock).
 2. *Proper completion*: quando un processo termina, non deve avere task in esecuzione.
 3. *No dead task*: il processo non deve contenere task che non possono essere eseguiti.
- Presenza di OR-Join non necessari.

- Presenza di Cancellation set non necessari.

3.4 Architettura di YAWL

Il sistema YAWL è suddiviso in due identità distinte tra loro e comunicanti: l'*editor* e l'*engine*. Le specifiche del workflow sono progettate utilizzando l'editor, mentre l'engine ha lo scopo di eseguire tutte le verifiche necessarie, registrare i task e memorizzarli in un repository che gestisce l'insieme di specifiche di workflow eseguibili. Le specifiche possono essere istanziate definendo così i case, in seguito l'engine è in grado di comunicare all'utente quali task sono pronti per andare in esecuzione. Nella figura viene rappresentata una prima architettura di YAWL.

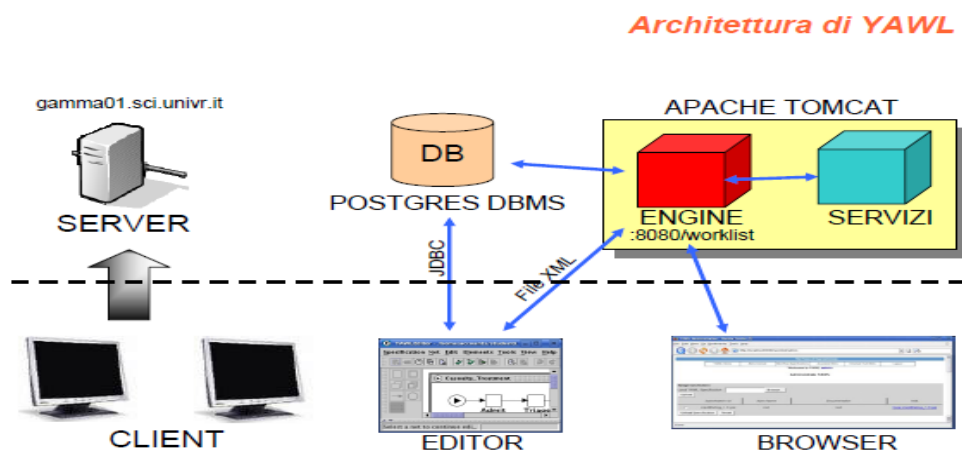


Figura 3.3 Architettura di YAWL

L'ambiente YAWL è composto da *servizi*, i principali sono quattro:

- gestore delle code di lavoro (YAWL worklist handler);
- agente per i servizi web (YAWL web services broker);
- agente di interoperabilità (YAWL interoperability broker);
- servizi personalizzati (custom YAWL services).

Il gestore delle code di lavoro corrisponde al classico gestore di worklist presente nella maggior parte dei WfMS. Si occupa di assegnare il lavoro agli utenti del sistema, grazie a questo servizio gli utilizzatori sono in grado di

accettare i workitem (unità di lavoro semplice, più workitem formano un'attività) e segnalare il loro completamento. In YAWL, a differenza dei sistemi di workflow tradizionali, il gestore delle code è considerato un servizio indipendente staccato dall'engine.

L'agente per i servizi web è lo strumento che permette il collegamento tra l'engine e tutti gli altri servizi web. Si comporta come un mediatore tra l'engine e i servizi web esterni.

L'agente di interoperabilità è un servizio utile per interconnettere diversi workflow engine. Ad esempio un task potrebbe essere ceduto ad un altro sistema dove lo stesso task corrisponde ad un intero processo.

L'ultimo servizio è il **servizio personalizzato** che connette l'engine con un'entità nell'ambiente del sistema, questo per rendere YAWL un sistema dinamico ed estendibile a piacimento. Le applicazioni esterne possono interagire con l'engine nei seguenti due modi:

1. **Direttamente**, utilizzando messaggi XML su protocollo HTTP, i servizi connessi in questa modalità vengono definiti Custom YAWL Services.
2. **Indirettamente**, sfruttando lo YAWL Web Services Invoker, in questo caso l'engine comunica con l'invoker che a sua volta inoltra le richieste ad un servizio esterno.

Questi servizi possono essere sviluppati in qualsiasi linguaggio di programmazione ed essere eseguiti su qualsiasi piattaforma capace di gestire messaggi XML via HTTP. Vengono registrati nel YAWL Engine specificando un URL dal quale risponde il servizio.

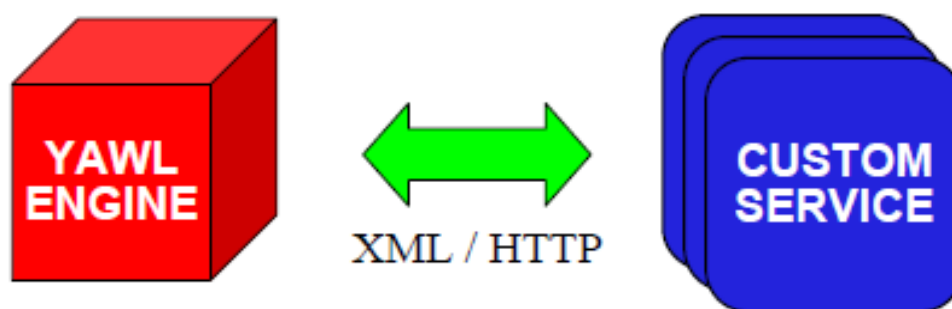


Figura 3.4 Dialogo tra Engine e Custom Service

In YAWL le specifiche di workflow sono gestite dal repository mentre le diverse istanze, o case, sono gestite direttamente dall'engine. In YAWL sono presenti anche cinque diverse interfacce ognuna delle quali ha un compito ben preciso:

1. **Interfaccia A:** gestisce le interazioni tra il progettista e il gestore di YAWL da un lato e l'engine dall'altro.
2. **Interfaccia B:** gestisce l'interazione tra i servizi di YAWL e l'engine.
3. **Interfaccia C :** permette l'interazione tra il servizio di gestione delle code di lavoro e la base di dati delle worklist;
4. **Interfaccia D:** usata per connettere il gestore delle code di lavoro ad applicazioni esterne
5. **Interfaccia X:** permette all'engine di notificare ai servizi personalizzati alcuni eventi fondamentali durante l'esecuzione di un'istanza di processo.

Ogni interfaccia di YAWL ha corrispondenze con le interfacce del WfMC (Workflow Management Coalition) ma a differenza degli altri WfMS esistenti l'engine ha a che fare con il flusso di controllo dei dati, non esplicitamente con gli utenti. Infatti nei WfMS classici l'engine si occupa un po' di tutti gli aspetti cioè del "cosa", del "come", del "quando" e di "chi", mentre in YAWL l'engine si occupa solo del "cosa" e del "quando" e delega ai servizi gli altri compiti, questo permette di realizzare un engine altamente efficiente e di personalizzare e aggiungere alcune funzionalità. L'Engine è gestito dal Java Web Server Tomcat, e per fare funzionare quest'ultimo sono necessari alcuni componenti riportati in seguito con l'estensione ".war" (web application):

- **YAWL Workflow Engine (yawl.war):** è il cuore dell'engine, dialoga con gli altri componenti attraverso messaggi XML.
- **Worklist Service(worklist.war):** alloca i workitem agli utenti e definisce chi è in grado di eseguire i diversi task all'interno di un processo

- **Web Service Invoker Service(YAWLWSInvoker.war):** permette all'engine di invocare servizi web
- **Administration Console (admintool.war):** comunica con l'engine tramite l'interfaccia "A". Permette la memorizzazione dei dati dell'organizzazione nella stessa base di dati del componente di logging e persistenza. Serve per aggiungere o modificare le risorse e le regole dell'organizzazione.
- **Worklet Service (WorkletService.war):** Permette la selezione dinamica del processo e un servizio di gestione delle eccezioni.

Lo YAWL WSInvoker è necessario quando un'applicazione esterna deve essere invocata dall'engine per eseguire il lavoro corrispondente ad un'istanza di task. Il task viene decomposto in un'invocazione al servizio esterno, e tale invocazione è espressa utilizzando l'interfaccia WSDL presente nella descrizione del processo.

Per eseguire il task via Web Service vengono effettuati i seguenti passi:

1. Quando viene attivata un'istanza del task, lo YAWL WSInvoker recupera l'interfaccia WSDL del task e, utilizzando il *Apache Web Services Invocation Framework (WSIF)*, ricava le informazioni necessarie per invocare il servizio.
2. Il passaggio all'applicazione esterna avviene immediatamente e lo stato del task evolve da "**enabled**" a "**processing**".
3. Quando il servizio termina di eseguire l'operazione richiesta, il task riprende il controllo passando in modalità "**completed**".

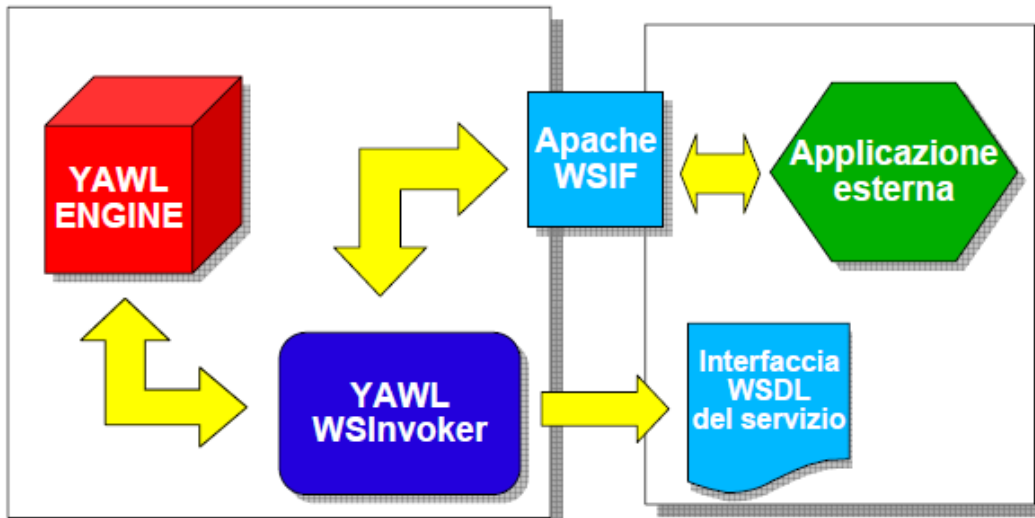


Figura 3.5 Dialogo tra i vari componenti di YAWL

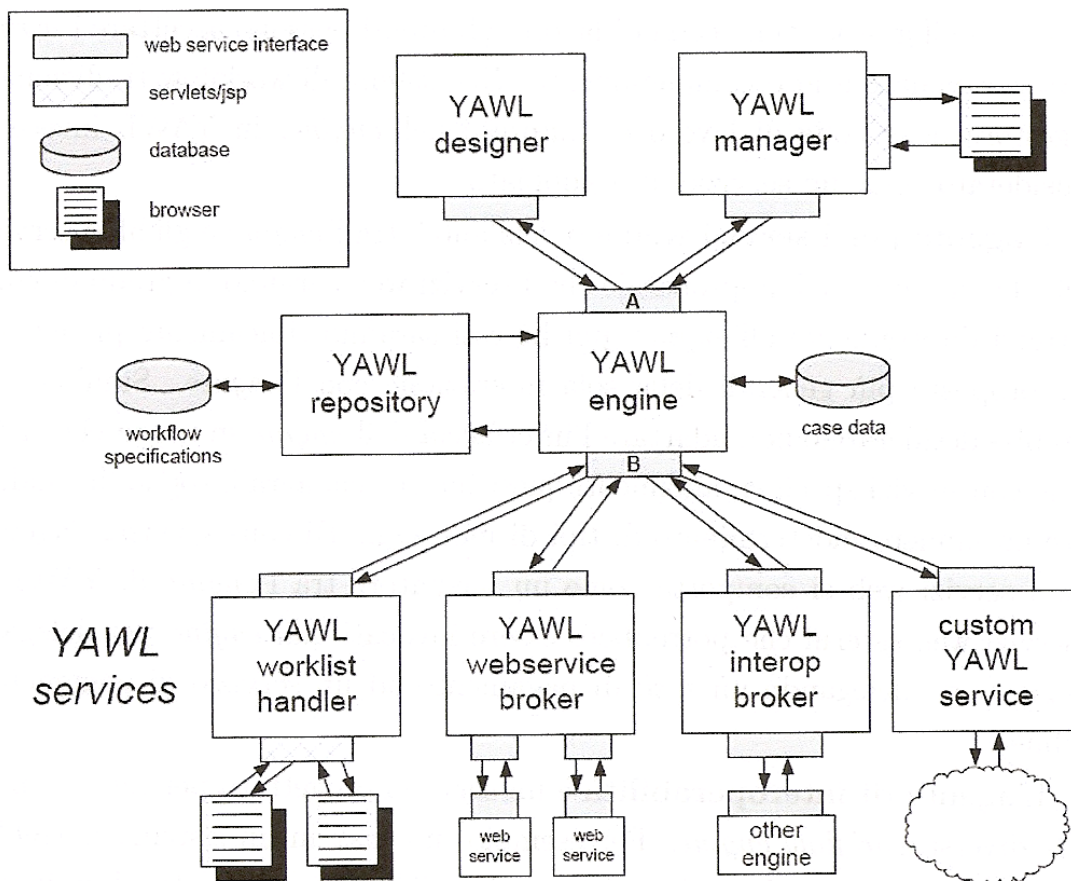


Figura 3.6 Architettura di YAWL

3.5 Modellazione dati

In YAWL le informazioni (data) sono memorizzate in documenti XML mentre i dati (data elements) sono memorizzati in variabili. Esistono due tipi di variabili:

1. **Variabili della rete:** dati accessibili dal task in una net
2. **Variabili di task:** dati che devono essere acceduti solo nel contesto di esecuzione di un task.

Tutti i dati sono definiti con un tipo (data type), il linguaggio fornisce alcuni tipi primitivi come boolean, long, time, ecc... I tipi semplici possono essere poi usati per creare tipi di dati complessi [5].

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="insegnamento">
    <sequence minOccurs="1" maxOccurs="1">
      <element name="Nome_corso" type="string" />
      <element name="Anno" type="long" />
      <element name="Docente" >
        <complexType>
          <sequence>
            <element name="Nome" type="string"/>
            <element name="Cognome" type="string" />
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</schema>
```

Figura 3.7 Esempio di definizione dati

L'editor permette di creare tipi di dati complessi attraverso XML Schema.

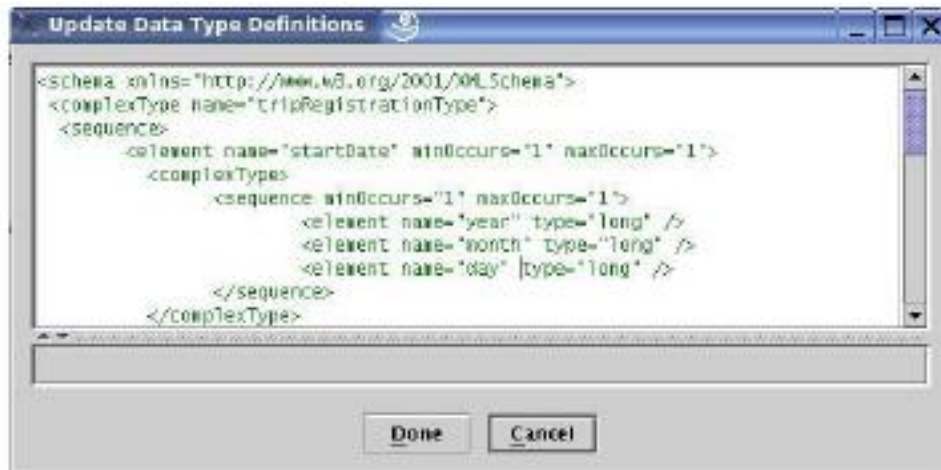


Figura 3.8 Esempio di definizione di tipi di dati complessi

Le variabili del linguaggio hanno diversi tipi di utilizzo:

- **Input only:** riceve i dati da una variabile della net
- **Output only:** fornisce i dati a una variabile della net
- **Input e output:** riceve e fornisce i dati da una variabile della net
- **Local variable:** applicabili solo a variabili della net ,utilizzate per manipolazioni interne alla net(quindi nel suo scope).

In YAWL il trasferimento di dati può avvenire attraverso *variabili* (passaggio interno di dati) o tramite *interazione tra processo e ambiente operativo* (passaggio esterno). Tutti i passaggi interni dei dati avvengono tra il singolo task e la net, utilizzando i parametri input e output del task [5].

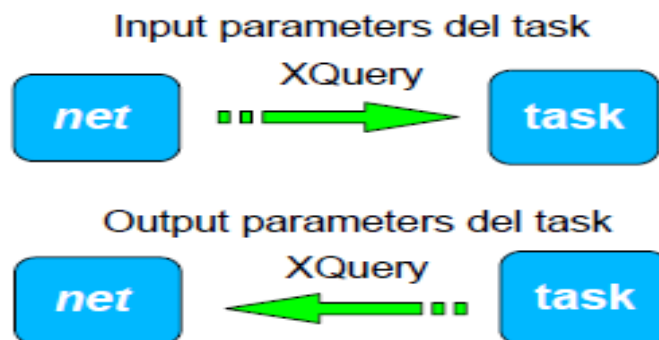


Figura 3.9 Dialogo tra task e rete

Non esiste il trasferimento interno dei dati direttamente tra task. I dati richiesti dall'ambiente esterno (variabili di input della net) sono gestiti da YAWL con la generazione automatica di form per richiedere informazioni da utenti o servizi associati, i form non sono però applicabili a variabili della net locali o alle variabili dei task.

The image shows a web form titled "Register". It contains the following elements:

- A "Customer" text input field containing "Mario Rossi".
- A section titled "Registr Info" containing:
 - A "Start Date" section with three input fields: "Year" (2007), "Month" (07), and "Day" (21).
 - An "End Date" section with three empty input fields: "Year", "Month", and "Day".
 - Three radio button options: "Want Flight" (true/false), "Want Hotel" (true/false), and "Want Car" (true/false).
 - A "Pay Acc Number" text input field.
- A "Submit" button.
- A legend at the bottom: "* - required | ? - help".

Figura 3.10 Esempio form

Inoltre YAWL fornisce alcuni servizi per supportare la modellazione del passaggio dei dati:

- Segnalazione automatica della mancanza di variabili necessarie per il trasferimento dati
- Generazione automatica di stringhe XQuery corrette compatibili con YAWL engine.

Per quanto riguarda gli split, nel caso in cui siano presenti le condizioni associate ai rami (branching condition), queste sono specificate come condizioni booleane XPath nei dettagli di flusso, per split OR e XOR. Il motore eseguirà la condizione che risulterà vera, o quella di default se tutte risultassero false. Solo le variabili della net sono interessate nelle specifiche delle condizioni di flusso.

Le variabili di task immagazzinano informazioni relative al task dove sono definite e hanno diverse tipologie di utilizzo tra cui:

- Trasferimento di informazioni tra task e net (tipo I/O)
- Memorizzazione dei dati localmente (input only)
- Presentazione dei dati in uscita del task (Output only)

Invece le variabili di net immagazzinano informazioni che riguardano la rete. Vengono principalmente usate per il passaggio di dati tra reti e task, reti e servizi dell'engine e per elaborazioni locali



Figura 3.11 Esempio descrizione dati

Infine i parametri di task permettono il passaggio di dati tra reti e task.

3.6 Comunicazione tra editor ed engine tramite XML, XPath e XQuery

In questa sezione si prenderà in considerazione la modalità con la quale editor ed engine comunicano tra loro, in particolare l'engine ha bisogno di caricare le specifiche create durante la fase di progettazione grazie alle potenzialità espressive dell'editor [5]. Per fare ciò una volta che il grafo rappresentante le dinamiche di uno specifico workflow è stato creato, esso deve essere salvato in un formato interpretabile dal motore di esecuzione. I progettisti di YAWL hanno deciso che il formato migliore fosse l'XML. XML è un meta-linguaggio per creare documenti arricchiti da informazioni aggiuntive. È un formato standard e flessibile per lo scambio di dati e informazioni tra applicazioni diverse (favorisce l'interoperabilità). L'editor, per mezzo della sua interfaccia grafica permette il salvataggio dello schema di workflow in due formati. Il formato YWL prevede il salvataggio non solo della struttura del workflow, ma anche delle informazioni grafiche. Questo è possibile grazie all'utilizzo della tecnologia JavaBeans, la quale permette di realizzare una descrizione XML dell'interfaccia sfruttando le classi XMLEncoder e XMLDecoder. I vantaggi offerti da questo tipo di approccio sono i seguenti:

- portabilità e indipendenza dalle versioni
- compattezza strutturale (XMLEncoder elimina le ridondanze)
- resistenza agli errori (se parte del file è danneggiata si può tradurre la parte non interessata all'errore)

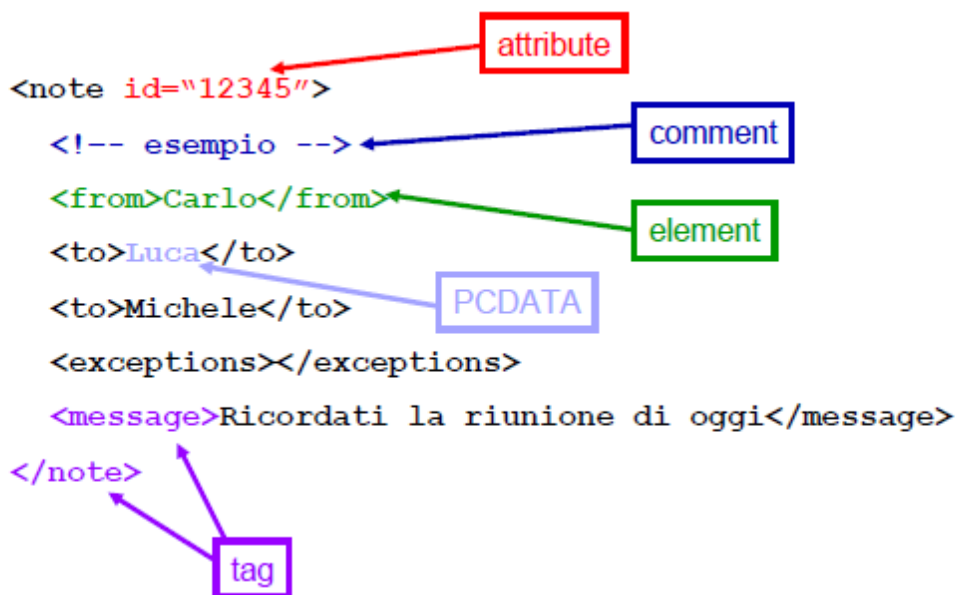


Figura 3.12 Spiegazione del linguaggio YWL

La Struttura del documento è rigorosamente ad albero, deve esistere un unico elemento radice e tutti i tag devono essere chiusi. Inoltre i valori degli attributi devono essere sempre inclusi tra apici singoli e doppi e bisogna prestare attenzione poiché XML è case sensitive. Inoltre la sintassi YWL è soggetta alle seguenti regole:

- ogni suo elemento rappresenta la chiamata ad un metodo;
- il tag object può contenere una expression il cui valore sarà usato come argomento per l'elemento in esso contenuto;
- il tag void può contenere uno statement che sarà eseguito, ma il cui risultato non sarà utilizzato come argomento per il metodo associato;
- gli elementi annidati sono utilizzati come argomenti, a meno che non siano preceduti dal tag void;
- il nome del metodo è indicato dall'attributo method;
- l'attributo classe è usato per specificare in modo esplicito l'obiettivo di un metodo statico o del costruttore;
- tutte le stringhe vengono codicate in UTF-8;

In XML esistono due tipi di relazione:

1. padre-figlio
2. fratello-fratello

In questo linguaggio non conta l'ordine in cui sono definiti i fratelli nel documento. Inoltre XML è uno standard che definisce la sintassi di un linguaggio facilmente interpretabile tramite, ad esempio, XLST, un parser standard che effettua le seguenti operazioni:

- Legge il documento XML e ne interpreta la sintassi
- Estrae tutte le informazioni e le fornisce all'applicazione
- Le applicazioni non devono preoccuparsi di gestire regole particolari
- Può leggere qualsiasi documento XML (indipendentemente dall'applicazione che lo ha creato)

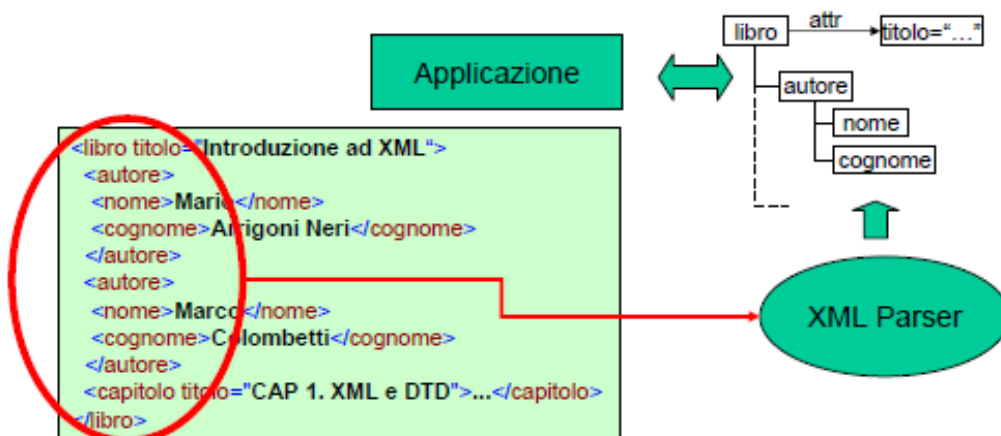


Figura 3.13 Esempio di XML

Il consorzio W3C nel 1998 ha avviato la definizione di un linguaggio standard per la navigazione dei documenti XML, chiamato XPath, e uno per l'interrogazione, chiamato XQuery.

XPath è un linguaggio tramite il quale è possibile esprimere delle espressioni per indirizzare parti di un documento XML. XPath opera su una rappresentazione logica del documento XML, che viene modellato con una struttura ad albero, e definisce una sintassi per accedere ai nodi di tale

albero. Le espressioni definite da XPath per accedere ai nodi dell'albero prendono il nome di *Location Path* (percorsi di localizzazione). Questi cammini possono essere assoluti se viene utilizzata la “/”, in questo caso la sequenza degli elementi inizia dalla radice fino al nodo selezionato. Se il cammino inizia con // vengono restituiti tutti gli elementi che soddisfano il criterio di selezione indipendentemente dal cammino usato per raggiungerli. Inoltre con il simbolo “*” si va a scegliere un elemento qualsiasi di una categoria selezionata.

Xpath permette di lavorare con:

- Stringhe
- Numeri
- Espressioni booleane

grazie all'utilizzo di una libreria di funzioni. Per selezionare i diversi rami si devono utilizzare le parentesi quadre “[]”, mentre grazie all'utilizzo dell'operatore “|” si possono selezionare diversi cammini. Per andare a selezionare uno specifico attributo di un elemento si utilizza l'operatore “@”. Come già detto XPath supporta espressioni numeriche, relazionali e booleane:

- Gli operatori numerici sono: +, -, *, div e mod
- Gli operatori di uguaglianza sono: = e !=
- Gli operatori di relazione sono: <, <=, >, >=
- Gli operatori booleani sono: or e and

L'ultimo linguaggio da presentare è XQuery, lo standard di interrogazione dei dati XML. Insieme a XPath forma un linguaggio completo: *XML Query*, che può essere accostato a SQL per i database.

L'interrogazione XQuery consta di 4 parti:

1. La clausola **for**, per la dichiarazione di variabili che permettono l'iterazione su elementi di un documento

2. La clausola **let** consente la dichiarazione di nuove variabili, eventualmente correlate con quelle presenti nel *for*
3. La clausola **where** per esprimere predicati e selezionare elementi che compongono il risultato dell'interrogazione
4. La clausola **return** che consente di definire il contenuto e la struttura del risultato.

Per la loro forma le interrogazioni XQuery sono dette espressioni FLWR (For-Let-Where-Return). Un esempio può essere:

```
for $x
```

```
IN document (catalogo.xml) // Autore
```

```
Return $x
```

Viene dichiarata la variabile (\$x) il cui valore è il risultato dell'espressione XPath *document (Catalogo.xml) // Autore* che trova tutti gli elementi Autore presenti nel documento catalogo.xml [5].

3.7 Vincoli temporali

Ora vengono analizzati i vincoli temporali che possono essere introdotti nello schema di workflow:

- **Durata di un task:** oltre all'attributo name relativo al task è presente anche l'attributo duration all'interno del quale è memorizzato il vincolo di durata con la seguente sintassi: [minDuration, maxDuration] granularity (es. <duration>[30,60] min</duration>);
- **Durata di un flusso:** l'attributo flowsInto di un task che si occupa di registrare i flussi uscenti può memorizzare anche la durata del flusso

stesso con la seguente sintassi: [minDuration, maxDuration] granularity (es. <duration>[5,6] h</duration>);

- **Vincoli relativi:** tra i vincoli temporali memorizzabili all'interno del tag timeConstraints di uno specifico task vi sono i vincoli relativi; la sintassi da rispettare è la seguente: S/E[minDuration, maxDuration] S/E granularity (es. <relativeConstrainttoTask=administer_mannitol_18 duration=S[3,5]E d/>);
- **Vincoli assoluti:** per memorizzare i vincoli assoluti relativi ad un task la sintassi del frammento XML è la seguente: [timestamp iniziale, timestamp finale] (es. <absoluteConstraint> not[25/12/2008 00:00:00, 25/12/2008 23:59:59] </absoluteConstraint>);
- **Vincoli periodici:** per memorizzare i vincoli periodici relativi ad un task la sintassi del frammento XML è la seguente: [p..q]/Days_In_Weeks (es. <periodicConstraint> [2..7]/Days_In_Weeks </periodicConstraint>);
- **Ritardi richiesti:** in questo caso si può avere un esempio della sintassi XML direttamente dal frammento riportato in precedenza.

In caso di un task con uno split di tipo OR o XOR, per determinare quale dei flussi riceverà un token si usa un predicato (elemento predicate figlio dell'elemento flowsInto) che contiene una espressione XPath da verificare per seguire questo determinato flusso. Il campo ordering, che fa parte dell'elemento predicate, dà un ordinamento, e quindi una priorità, a questi flussi, utile in caso si utilizzi uno XOR-split. Il funzionamento di OR e XOR split è quindi il seguente: nel caso di XORsplit verrà mandato un token all'elemento con ordering più basso tra quelli il cui campo predicate risulta essere vero, e se nessun predicate risulta verificato, il token verrà mandato comunque nel flusso che ha l'elemento che lo contraddistingue come flusso di default, ovvero l'elemento isDefaultFlow (figlio anche questo dell'elemento flowsInto). L'OR-split invece manderà un token a tutti i flussi il cui campo predicate risulta essere verificato, e nel caso nessuno lo fosse verrà mandato al flusso di default, ovvero quello con l'elemento isDefaultFlow.

3.8 Modellazione degli aspetti temporali

In questo paragrafo vengono presentati i concetti temporali di base utilizzati in un processo. Il modello su cui si basa il modello temporale implementato in YAWL considera gli istanti e le durate come tipologie temporali elementari. Gli istanti sono punti su un dominio temporale, mentre le durate sono lunghezze su un dominio temporale. Gli intervalli sono tipi derivati e possono essere definiti ciascuno come una durata dopo un istante o come una durata prima di un istante o ancora come la distanza tra due istanti di inizio e fine. Gli istanti sono rappresentati attraverso timestamp, ogni timestamp è definito con una granularità. Una granularità temporale intuitivamente partiziona il dominio temporale in una sequenza di granuli, ossia insiemi di punti su un dominio temporale. Ecco nel dettaglio quali sono gli aspetti temporali che possono essere gestiti in YAWL:

EVENTI ESTERNI

In YAWL un evento esterno è raffigurato da un'icona a forma di orologio rotondo e indica un istante di tempo che potrebbe influenzare l'esecuzione delle attività. Infatti, gli eventi potrebbero essere collegati ad attività attraverso vincoli relativi (relative constraints) o ritardi richiesti (required delays).

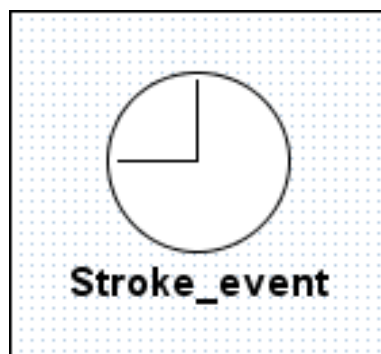
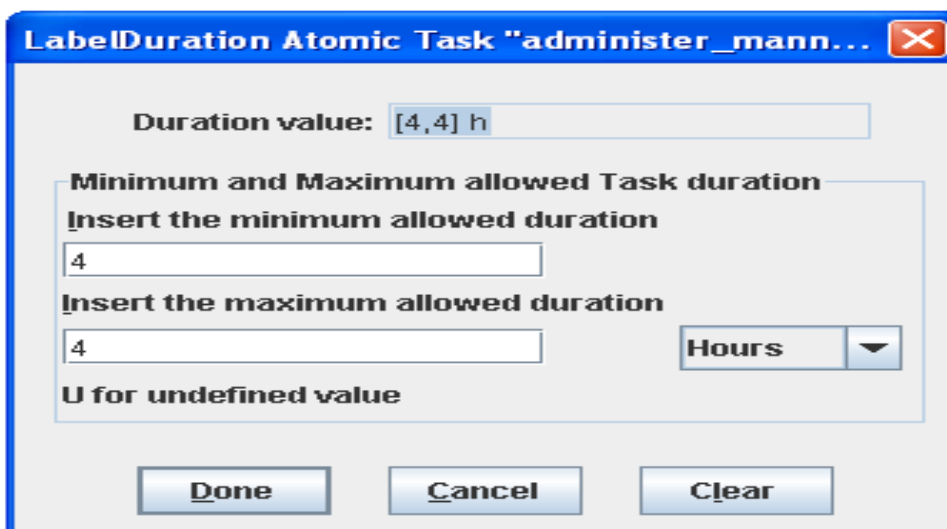


Figura 3.14 Evento esterno in YAWL

DURATE E RITARDI

La durata è una proprietà temporale sia degli archi che dei nodi. La durata di un nodo è la distanza temporale tra gli istanti di inizio e di fine dell'attività. La durata sull'arco, chiamata delay, denota l'intervallo tra l'istante di fine del predecessore e l'istante di inizio del successore, quindi ad esempio la durata a disposizione del WfMS per organizzare l'inizio dell'attività dopo la fine di una precedente. Le durate vengono espresse implementando un modello concettuale che sfrutta la nozione di intervallo: l'attributo di conseguenza ha la seguente forma: [MinDuration,MaxDuration] Granule, dove i vincoli rappresentano la durata minima e la durata massima permessa. In alcuni casi viene definita solo la durata minima o massima di conseguenza gli intervalli potrebbero essere solo parzialmente definiti. Se l'intervallo è completamente specificato durate e ritardi sono espressi attraverso numeri interi, altrimenti si utilizza il simbolo U per rappresentare valori indefiniti. Se il progettista di workflow non imposta la durata su un componente, il valore di default è [0,0] MinGranularity dove MinGranularity è la granularità minima dello schema. In questo caso, l'attività è considerata come istantanea invece se il vincolo è ad esempio [1,1] min; l'intervallo lecito viene inteso [1 min, 1 min 59 s] e quindi l'attività risulta non istantanea.



The image shows a software dialog box for configuring an atomic task. The title bar reads "LabelDuration Atomic Task 'administer_mann...". The main content area includes a "Duration value:" label followed by a text input field containing "[4,4] h". Below this is a section titled "Minimum and Maximum allowed Task duration" which contains two sub-sections: "Insert the minimum allowed duration" with an input field containing "4", and "Insert the maximum allowed duration" with an input field containing "4". To the right of the second input field is a unit selection dropdown menu currently set to "Hours". At the bottom of the dialog, there are three buttons: "Done", "Cancel", and "Clear".

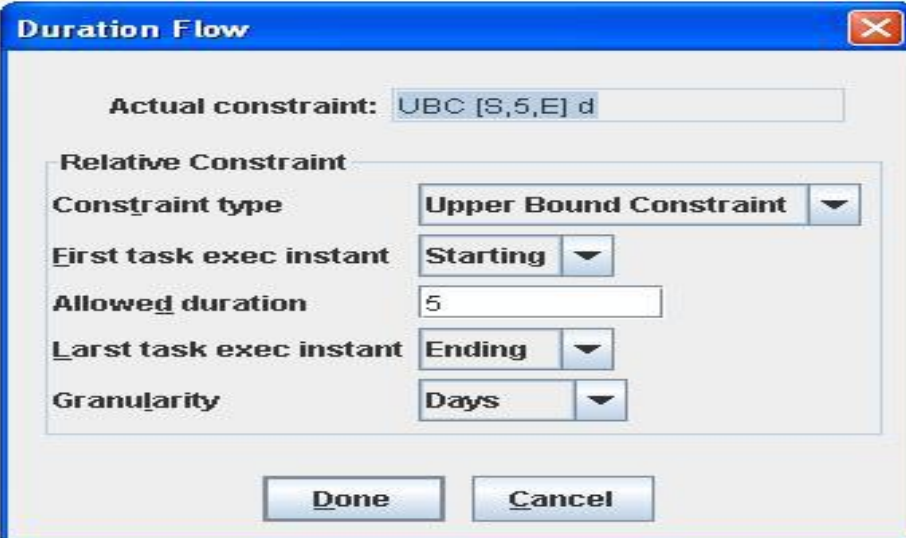
Figura 3.15 Inserimento task atomico

VINCOLI TEMPORALI

Diversamente dagli attributi sulle durate, i vincoli temporali non sono obbligatori per ogni componente del workflow, modellano proprietà temporali addizionali e devono essere controllati dal WfMS.

Essi sono:

- **Vincoli relativi:** un vincolo relativo limita la distanza di tempo (durata) tra gli istanti di inizio e di fine di task non successivi. $S/E [TimeDistanceMin, TimeDistanceMax] S/E$ Granule denota che non deve passare meno di TimeDistanceMin e più di TimeDistanceMax tra l'inizio o la fine del primo task e l'inizio o la fine dell'ultimo task. Un vincolo upper bound può modellare inoltre una deadline: questa corrisponde al tempo massimo che si ha a disposizione per l'esecuzione dell'attività. Invece, se la prima attività di un vincolo upper bound è lo startcase (task di partenza) di uno schema, allora il valore del TimeDistance rappresenta la deadline rispetto all'ultima attività rappresentata nel vincolo upper bound.



The image shows a dialog box titled "Duration Flow" with a close button in the top right corner. Inside the dialog, there is a text field labeled "Actual constraint:" containing the text "UBC [S,5,E] d". Below this is a section titled "Relative Constraint" containing several fields:

- "Constraint type" is a dropdown menu set to "Upper Bound Constraint".
- "First task exec instant" is a dropdown menu set to "Starting".
- "Allowed duration" is a text input field containing the number "5".
- "Last task exec instant" is a dropdown menu set to "Ending".
- "Granularity" is a dropdown menu set to "Days".

At the bottom of the dialog are two buttons: "Done" and "Cancel".

Figura 3.16 Inserimento vincolo relativo

- **Vincoli assoluti:** Un vincolo assoluto rappresenta un intervallo di tempo durante il quale un'attività può essere eseguita. I limiti dell'intervallo sono espressi attraverso timestamp: i limiti di un vincolo assoluto sono indipendenti dal timestamp che rappresenta l'inizio dell'esecuzione del case. L'ampiezza dell'intervallo di tempo per un vincolo assoluto deve essere più grande della durata minima definita per la corrispondente attività. Un vincolo assoluto può essere usato anche per rappresentare un intervallo di tempo durante il quale un'attività non può essere eseguita: in questo caso, la parola chiave not deve essere inserita prima della specifica dell'intervallo.

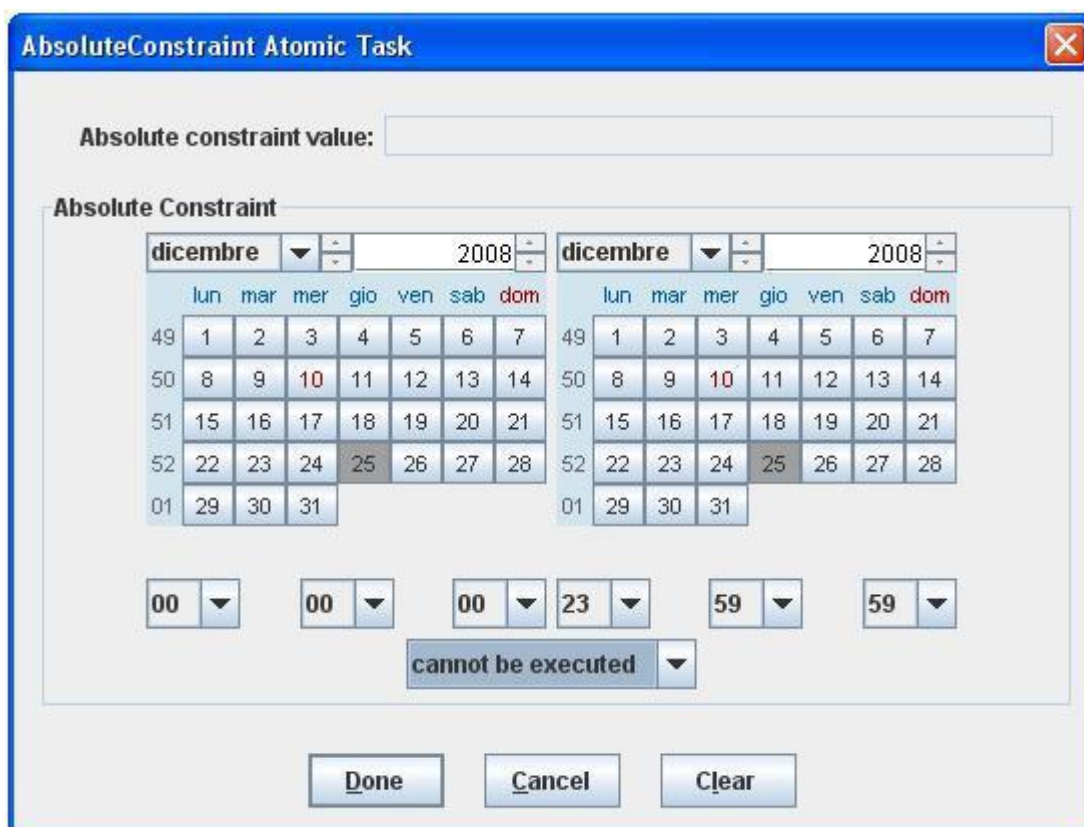


Figura 3.17 Inserimento vincolo assoluto

- **Vincoli periodici:** Un vincolo periodico rappresenta un intervallo di tempo periodico durante il quale un'attività può iniziare ad essere eseguita. Inoltre il vincolo periodico può essere usato anche per

rappresentare un intervallo di tempo durante il quale un'attività non può essere eseguita: in questo caso, la parola chiave not viene inserita prima della specifica dell'intervallo di tempo.

Figura 3.18 Inserimento vincolo periodico

- Ritardi richiesti:** I ritardi richiesti (required delays) rappresentano la distanza di tempo minima tra due attività non successive: non è un vincolo temporale perchè non può essere violato, in quanto il WfMS attende finchè il ritardo specificato non è trascorso. I ritardi richiesti sono raffigurati nell'editor come archi a tratti grandi che connettono due nodi non successivi. Gli archi sono etichettati dal periodo tra le due attività e la corrispondente granularità [TimeDistance] Granule. I ritardi richiesti sono concettualmente differenti dai vincoli relativi: infatti questi ultimi rappresentano dei limiti che hanno bisogno di un meccanismo di gestione delle eccezioni se vengono violati. Invece, i ritardi richiesti rappresentano un intervallo di tempo che il WfMS deve in ogni caso attendere finchè il ritardo richiesto non è trascorso.

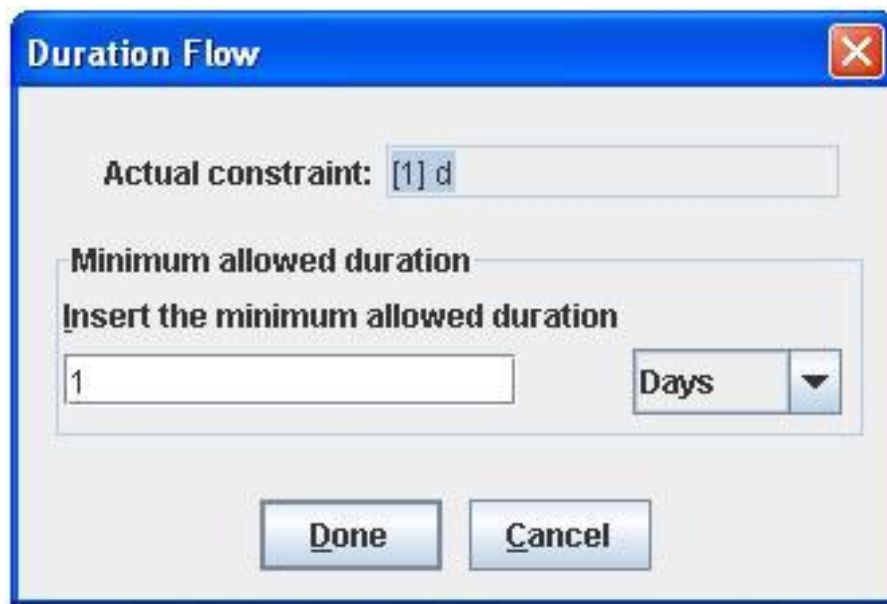


Figura 3.19 Inserimento vincolo di flusso

3.9 YAWL Engine

In questa sezione verrà introdotto lo YAWL Engine che permette l'esecuzione dello schema di workflow creato con lo YAWL Editor. L' Engine è in esecuzione presso una macchina server a cui si può accedere utilizzando un browser. L'indirizzo a cui risponde lo YAWL Engine, ad esempio, è: <http://gamma01.sci.univr.it:8080/worklist>. Per loggarsi come amministratore basta inserire come user *admin* e come password *YAWL* ricordando che YAWL è case sensitive. La Worklist si presenta come in figura:

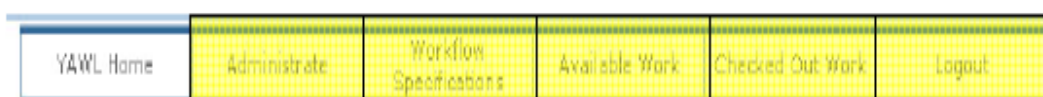


Figura 3.20 Schede Engine

Ogni scheda permette varie operazioni:

- **Administrate:** si accede alla pagina di caricamento delle specifiche di workflow. Riservata agli amministratori.
- **Workflow Specifications:** informazioni sulle specifiche presenti e i rispettivi casi in esecuzione, lancio di nuovi casi di una specifica, terminazione manuale di un caso. Vengono visualizzate tutte le specifiche di workflow caricate nell'engine ed i relativi casi già in esecuzione di cui può essere visualizzato lo stato. Da questa pagina è possibile lanciare nuovi casi di una specifica o eliminare quelli già in esecuzione.

The screenshot shows the YAWL admin interface. At the top, there is a navigation bar with the following tabs: YAWL Home, Administrate, Workflow Specifications, Available Work, Checked Out Work, and Logout. Below the navigation bar, the text 'Welcome to YAWL admin' is displayed. The main content area is titled 'Active YAWL Specifications' and contains a table with the following columns: Specification ID, Spec Name, Documentation, XML, and Cases. The table has one row with the following data: Specification ID: creditRating_1.3.ywl, Spec Name: null, Documentation: null, XML: View creditRating_1.3.ywl, Cases: 6. Below the table, there are two buttons: 'Launch Case' and 'Reset'. At the bottom of the page, there is a footer that reads 'YAWL is distributed under the LGPL'.

Specification ID	Spec Name	Documentation	XML	Cases
creditRating_1.3.ywl	null	null	View creditRating_1.3.ywl	6

Figura3.21 Esempio Engine

- **Available Work:** task disponibili pronti per essere attivati dall'utente. L'utente vede la lista dei task disponibili per la risorsa (ruolo o utente) che rappresenta a seguito dell'autenticazione. A seguito del check out il task scompare dalla lista e lo ritroveremo tra i Checked Out Work.

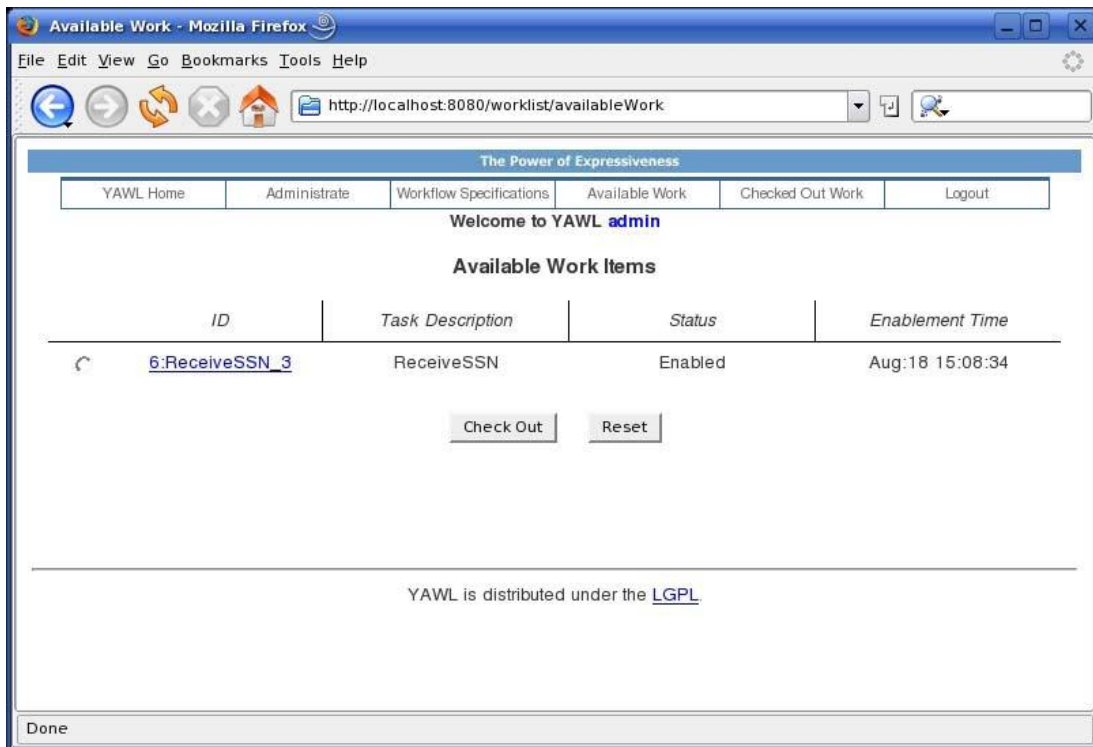


Figura 3.22 Esempio di work avviabili

- **Checked Out Work:** task attivati in attesa di esecuzione da parte dell'utente (immissione dati). Dalla pagina "Checked Out Work Items" è anche possibile lanciare nuove istanze di un Multiple Instance Task oppure sospendere un task di cui è stato eseguito il Check Out riportandolo così nella lista degli "Available Work Items".

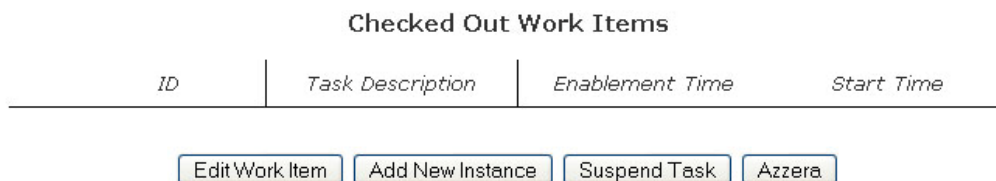


Figura 3.23 Esempio task in attesa

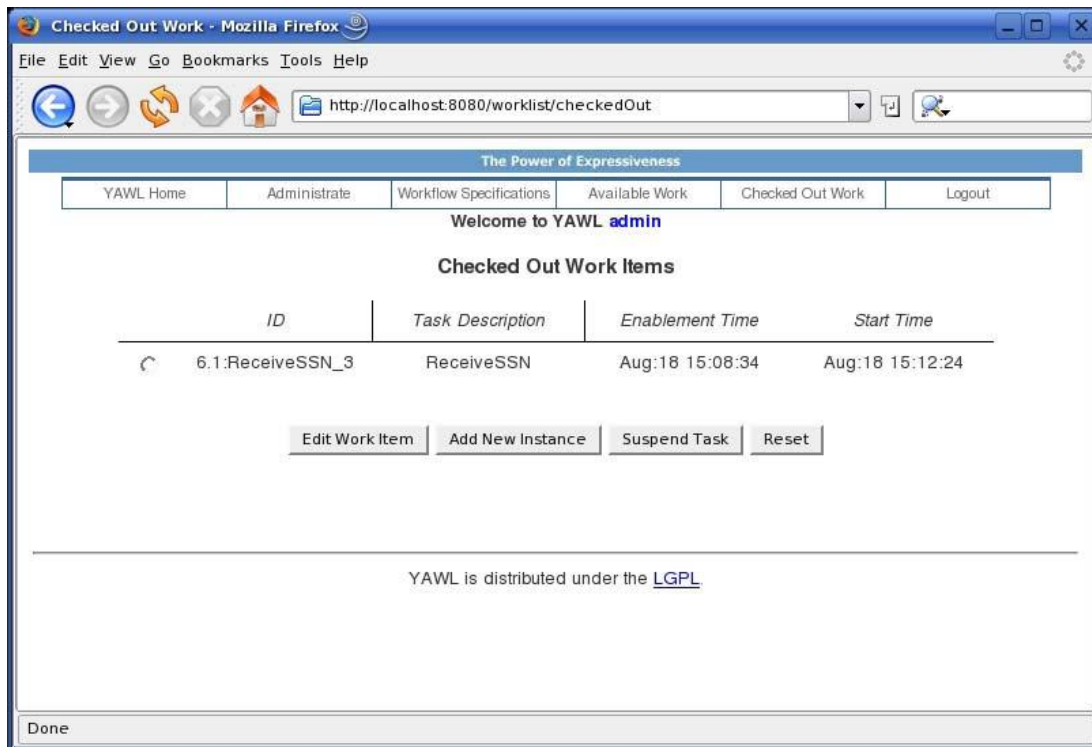


Figura 3.24 Esempio task in attesa

- **Logout:** esce dal sistema.

YAWL costruisce automaticamente un'interfaccia grafica per facilitare l'immissione dei dati richiesti dal task così come questo è stato implementato nella specifica. In caso di fallimento di questa operazione, causato dalla complessità dei tipi di dato costruiti dall'utente o dal verificarsi di un errore, il sistema permette comunque l'inserimento dei dati direttamente in formato XML. Nell'area riservata all'amministratore si possono svolgere parecchie funzioni:

- Caricamento ed eliminazione specifiche nell'engine
- Gestione utenti e ruoli e possibilità di statistiche tramite grafici o tabelle

Le risorse sono le entità che interagiscono con l'engine. Possono essere degli utenti (human) o altre componenti del sistema informativo (machine) in cui è integrato il WfMS (Workflow Management System).E' possibile

assegnare uno o più ruoli ad una risorsa selezionandoli tra quelli presenti nell'engine. E' possibile anche raggruppare le risorse in ruoli.

Details for a human resource

Select Human ResourceID:

Resource ID:

Description:

Is of Type: Human Non-Human

If the resource is of type 'Human' fill in the fields below:

Given Name:

Surname:

Has access to: Worklist Administration Tool

Initial Password:

Confirm Password:

Figura 3.25 Esempio tabella di un task

Grafici e tabelle permettono all'amministratore di monitorare l'esecuzione di casi e task. Attraverso l'interfaccia grafica è possibile costruire le query necessarie ad estrarre i dati dal database e aggregarli al fine di produrre reportistica utile all'amministrazione. Possono essere costruite differenti viste dei dati per confrontare l'efficienza delle risorse, dei casi e delle attività. Per la creazione di tabelle o grafici sono necessarie tre fasi essenziali:

- **Selezione dati:** E' possibile scegliere se si intende visualizzare i dati relativi ai casi (quanto tempo viene impiegato per eseguirli, quanti ne sono stati eseguiti,...) oppure quelli relativi ai singoli task. Inoltre è necessario scegliere la forma di visualizzazione che si desidera

ottenere: tabella riepilogativa o grafico (il tipo di grafico verrà stabilito successivamente).

The screenshot shows the 'YAWL Administration and Monitoring Tool' interface. At the top, there is a navigation bar with tabs for 'Resources', 'Roles', 'Charts', 'Worklist', and 'Logout'. Below this, the 'Overview' section contains the text 'Please select which chart you would like to view below.' The main area is labeled 'STEP 1:' and includes 'Select Element' (set to 'Cases'), 'Select View' (set to 'Table'), and a 'Start New Query' button.

Figura 3.26 Selezione dati

- **Filtri dei dati:**

The screenshot shows the 'STEP 2:' configuration for data filters. It includes dropdown menus for 'Specification', 'Case', and 'Resource', all set to '-All-'. The 'Status' section has two dropdown menus for '-Select Option-' and a checkbox. The 'Interval Filter' section has a dropdown for '-Select Option-', a 'More' dropdown, and a 'Seconds' dropdown. A callout box explains that data can be filtered by specification, case, or resource. Another callout explains that the status filter selects cases exclusively in a specific state (e.g., in execution, completed, etc.) within a time interval. A third callout explains that the interval filter selects only cases/work items that complete an execution phase within a certain time interval.

Figura 3.27 Filtri dei dati

Per i casi la fase di esecuzione è sempre del tipo “startedcompleted” mentre per i work item si può scegliere tra 3 fasi:

1. Enabled-started: da quando sono disponibili per l'esecuzione (available) a quando viene effettuato il check out.

2. Started-completed: da quando viene effettuato il check out a quando viene completato.
3. Enabled-started: corrisponde all'esecuzione completa di un task.

Possono essere inseriti più filtri, in quanto il sistema effettua l'unione dei risultati filtrati, ma è obbligatorio averne almeno uno. Un filtro di default causa la cancellazione di tutti gli altri filtri.

- **Scelta tipo di grafico o tabella:** Permette di personalizzare la creazione del grafico scegliendo il tipo, il raggruppamento degli elementi e i valori da visualizzare. Oppure permette di personalizzare la creazione della tabella scegliendo il tipo di raggruppamento dei dati desiderato.

Per finire ricordiamo due operazioni fondamentali:

- Collegarsi al PostgreSQL

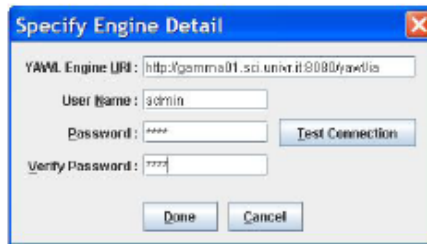


Parametri da impostare:

```
DB Server URI: jdbc:postgresql://gamma01.sci.univr.it:5432/  
DB Name: yawl  
DB User-id: postgres  
DB Password: yawl
```

Figura 3.28 Collegamento al database

- Collegarsi allo YAWL Engine



Parametri da impostare:

YAWL Engine URI: `http://gamma01.sci.univr.it:8080/yawl/ia`

User Name: `admin`

Password: `YAWL`

Figura 3.29 Collegamento allo YAWL Engine

CAPITOLO 4

Estensione fuzzy di YAWL

4.1 Introduzione

Sulla base delle precedenti osservazioni bisogna prima di ogni altra cosa decidere quali sono i vincoli sui quali è possibile applicare un qualsiasi processo di fuzzificazione. La decisione è stata presa sulla base di due considerazioni:

1. Valutare se il vincolo, per definizione, supporta una qualche forma di fuzzificazione;
2. Valutare attentamente se ha senso in termini di guadagno espressivo e di correttezza sintattica introdurre il concetto di incertezza;

Per le precedenti considerazioni è stato deciso di escludere alcuni dei vincoli temporali, primi tra tutti gli eventi esterni che, per definizione, sono delle attività istantanee non prevedibili a priori e in quanto tali durante la fase di progettazione di un workflow non è in alcun modo possibile acquisire informazioni utili ad un processo del genere. Il concetto di incertezza è già implicito in un evento esterno dato che esso può manifestarsi in un qualsiasi momento; se ciò non fosse, si parlerebbe di una semplice attività. Per quanto

riguarda invece i vincoli periodici, essi in base alla definizione fornita rientrerebbero nella categoria dei vincoli ai quali associare un grado di incertezza. Si è deciso, nonostante tale considerazione, di escludere anch'essi dalla trattazione in quanto è stato valutato come minimo, se non addirittura negativo, il guadagno apportato dall'introduzione della logica fuzzy. Essi rappresentano gli unici vincoli sui quali non è possibile speculare, infatti non hanno senso asserzioni del tipo “più o meno il secondo giorno della settimana”. È più facile trovarsi in situazioni in cui non si hanno informazioni sufficienti per affermare se un'attività può essere eseguita all'interno di un intervallo temporale assoluto (vincoli assoluti) oppure per definire la durata precisa di tale attività (vincoli di durata), o ancora per definire i rapporti tra attività consecutive (vincoli di durata sui flussi) e non consecutive (vincoli relativi e ritardi richiesti). Durante la fase di definizione si è deciso di lasciare la massima libertà all'utente che utilizzerà il sistema permettendogli di esprimere l'incertezza in termini percentuali. Il rapporto è il seguente: maggiore è la percentuale, minore è il livello di conoscenza dell'informazione che si sta esprimendo. Una percentuale pari a 0 indica che si possiede la massima consapevolezza del vincolo che si sta introducendo; al contrario una percentuale pari a 100 indica che l'incertezza è elevata. Il valore 0 è stato introdotto per fare in modo che il sistema originale venga preservato nel caso in cui non si voglia considerare la componente di incertezza nella definizione del workflow mantenendo rettangolare la funzione di appartenenza. Più la percentuale si abbassa e più il confine si assottiglia verso il nucleo del fuzzy set. Per calcolare il trapezio si utilizzeranno le seguenti formule:

Per gli istanti:

- $a = b - f(i)$, $b = l(i)g(i)$, $c = u(i)g(i)$, $d = c + f(i)$

con $f(i) = [(c - b)/k(i)] * (x/100)$ e “ $k(i)$ ” = 12 e “ x ” = incertezza inserita dall'utente

Per le durate:

- $a = b - (b/k(i))*(x/100)$, $b = l(i2g(i))$, $c = u(i2g(i))$, $d = c + (c/k(i))*(x/100)$
con “k(i)” = 5 e “x” = incertezza scelta dall'utente

Una volta che in fase di progettazione tutti i vincoli temporali sono stati tradotti in istanti o durate lecite attraverso la costruzione dei trapezi si passa alla fase di esecuzione. Durante questa fase le attività del workflow vengono svolte e l'engine di YAWL si occupa di verificare se e in che modo i vincoli definiti sono stati rispettati. A seconda dei vincoli la fase di controllo può avvenire al momento di iniziare l'esecuzione dell'attività, al momento di terminare l'esecuzione dell'attività oppure in entrambe le occasioni. Se un vincolo non viene rispettato viene lanciata un'eccezione e l'istanza di workflow viene dichiarata fallita. In questo caso occorre ricominciare da capo l'esecuzione. Non è possibile effettuare alcuna operazione di predizione sui cammini migliori in termini di ottimalità poichè il grado di ottimalità dipende esclusivamente dal momento in cui il nodo verrà attraversato, informazione che non si può conoscere a priori.

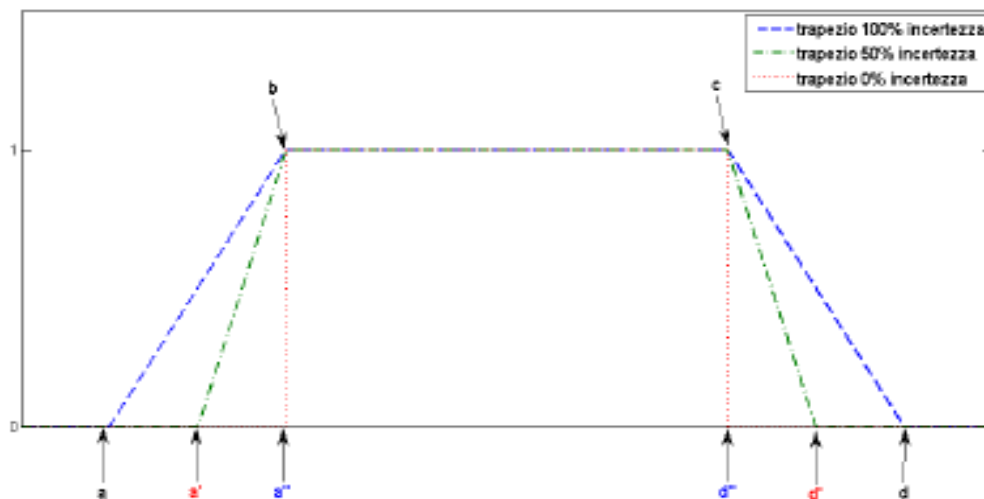


Figura 4.1 Esempio trapezio

Per l'implementazione della fuzzy logic in YAWL sono state necessarie 3 fasi principali:

1. La prima fase è consistita nella definizione dei vincoli temporali fuzzy durante la progettazione (editor) costruendo opportune interfacce utente per l'inserimento, e attraverso semplici algoritmi, la verifica della consistenza dei vincoli stessi.
2. Nella seconda fase è stata adattata la sintassi XML per tener conto dei nuovi costrutti in modo che fosse interpretabile dall'engine di YAWL.
3. La terza fase è consistita, infine, nella messa a punto di procedure per la verifica, durante il processo di esecuzione, del soddisfacimento dei vincoli definiti al momento della progettazione.

4.2 Inserimento vincoli temporali nell'editor

Il primo passo nell'implementazione dell'estensione fuzzy [2] è stato quello di modificare l'interfaccia utente in modo da permettere l'inserimento delle informazioni necessarie per modellare in modo fuzzy ciascuno dei vincoli temporali ritenuti idonei ad un'operazione del genere. Il package principale sul quale sono state effettuate pressochè tutte le modifiche è *.swing* nel quale si estende il framework Java Swing per la comunicazione con l'utente. All'interno della cartella *element* sono presenti le classi che definiscono le interfacce di comunicazione grafica con l'utente. Il primo vincolo trattato è quello assoluto. L'inserimento di un vincolo assoluto è possibile grazie alla classe *AbsoluteConstraintDialog* all'interno della quale sono presenti tutte le istruzioni necessarie per la creazione dell'interfaccia utente. Grazie all'estensione della classe *JComboBox* sono stati realizzati i campi per l'inserimento delle informazioni riguardanti la granularità (*ExtendedGranularityComboBox*), la tipologia (*TypeConstraintComboBox*) e l'incertezza (*PercentageComboBox*) di ognuno dei due istanti. Per quanto riguarda la definizione dell'istante vero e proprio si sono utilizzati oggetti di tipo *JCalendarInstance* per la creazione del Calendario Gregoriano più tre

ulteriori oggetti JComboBox per individuare l'ora, il minuto e il secondo desiderati. Tra tutti questi oggetti esistono delle dipendenze che concernono la possibilità di restringere l'insieme dei valori leciti di un campo grazie all'utilizzo di meccanismi di reazione basati sulla classe ActionListener :

- Se la granularità scelta coincide con i secondi, allora i campi con etichetta Type e Uncertainty assumono un valore rispettivamente pari a "Exactly On" e "0" non modificabile finchè la granularità non viene incrementata;
- Se la granularità selezionata G è meno fine rispetto ai secondi allora i campi per l'inserimento dei granuli appartenenti alle granularità più fini di G vengono disabilitati.

In seguito all'inserimento di tutti i dati necessari, al momento di cliccare il tasto Done viene chiamato il metodo getDateAsString() che si occupa di memorizzare il vincolo, questo metodo è stato modificato per renderlo idoneo all'estensione fuzzy.

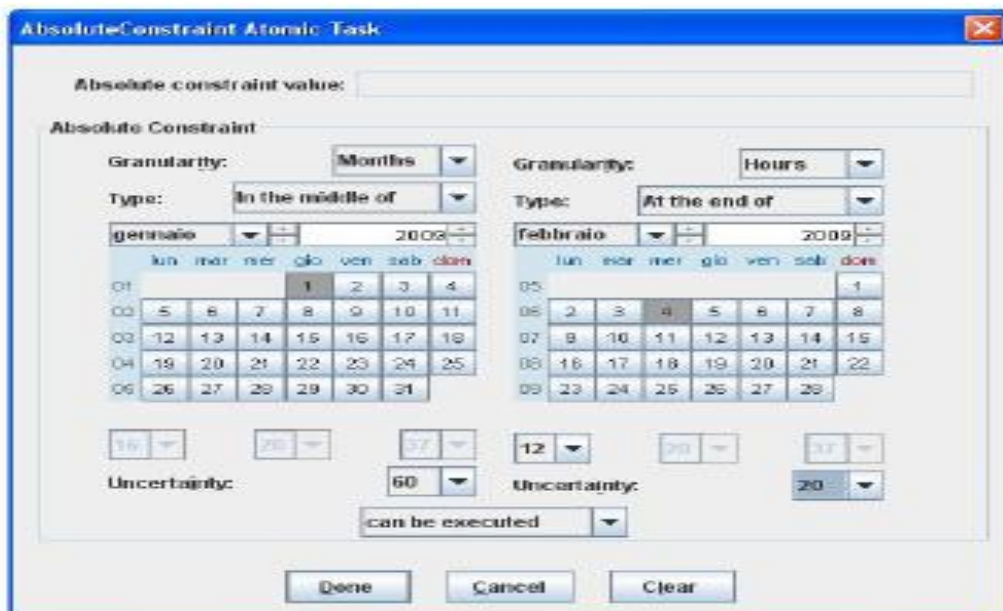


Figura 4.2 Inserimento di un vincolo assoluto

L'inserimento di un vincolo di durata dipende dalla sua tipologia. Se si tratta della durata di un'attività la classe corrispondente è *DurationElementDialog*, se si tratta invece della durata di un flusso la classe di riferimento è *DurationFlowElementDialog*. Le due classi differiscono tra loro semplicemente per le modalità con le quali il vincolo viene memorizzato mentre la sintassi del vincolo è identica. Le operazioni di modifica hanno riguardato sostanzialmente l'introduzione di due campi aggiuntivi di tipo *PercentageComboBox* attraverso i quali è possibile indicare le percentuali, distinte, di incertezza relative al lower bound e all'upper bound sulla durata.

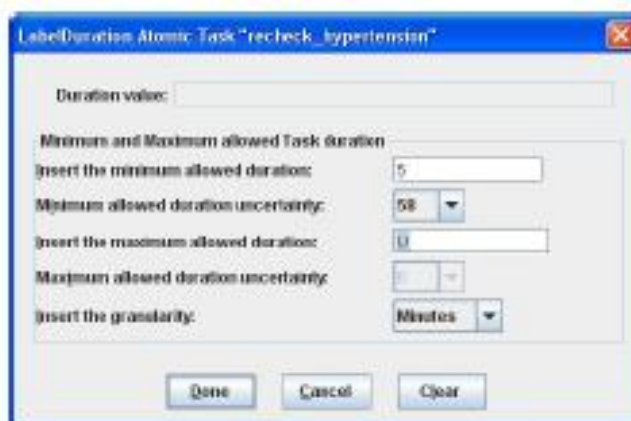


Figura 4.3 Inserimento di un vincolo durata

L'inserimento di un vincolo relativo fa riferimento alla classe *RelativeConstraintDialog* che è stata modificata per introdurre l'incertezza. In questo caso è stato ritenuto sufficiente introdurre un unico valore comune di percentuale attraverso un oggetto di tipo *PercentageComboBox* il cui campo di inserimento viene disabilitato nel caso di una granularità impostata pari ai secondi.

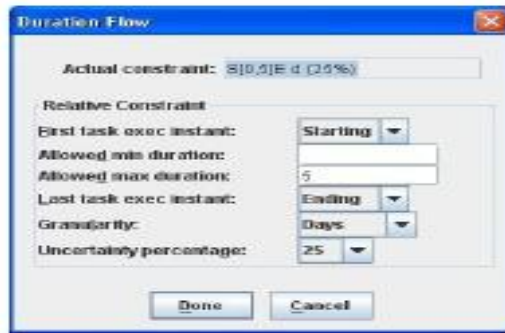


Figura 4.4 Inserimento di un vincolo relativo

I ritardi richiesti vengono modellati seguendo le istruzioni presenti nella classe *DurationFlowElementDialogT*. Come per i vincoli relativi è possibile inserire un unico valore che rispecchia la percentuale di incertezza relativa al vincolo che viene azzerato se la granularità coincide con i secondi.

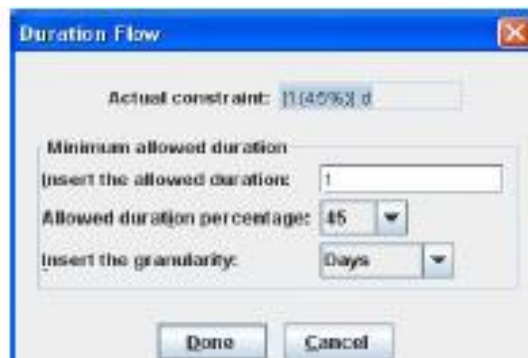


Figura 4.5 Inserimento di un vincolo di flusso

4.3 Passaggio tra editor ed engine

Una volta che tutti i vincoli necessari sono stati introdotti grazie alle interfacce modificate dell'editor, si è preso in considerazione il problema di modificare le classi relative all'esportazione e all'importazione della specifica in entrambi i formati YWL e XML. È la classe *Java EngineSpecificationExporter* che si

occupa di richiamare il metodo *marshal* (metodo per trasformare le specifiche dei vincoli in XML) della classe *YMarshal dell'engine*, il quale a sua volta, data la specifica passata dall'editor richiama il metodo *toXML* della classe *YExternalNetElement*. Ad ognuno degli oggetti creati in fase di progettazione è stato associato un metodo per tradurre il vincolo in una stringa. Ad ogni vincolo temporale è associata una classe engine presente nel package *.elements* che effettua l'operazione richiesta. Il cambiamento è proporzionale alla quantità di informazione aggiuntiva introdotta grazie alla nuova definizione del vincolo:

- *YAbsoluteConstraint*
- *YrequiredDelay*
- *YRelativeConstraint*
- *YminMaxDuration*

Capitolo 5

Consistenza del workflow temporale

5.1 Introduzione

In generale per verificare la consistenza di un workflow si deve risolvere un *Problema di Soddisfacimento di Vincoli* (**CSP** Constraint Satisfaction Problem) che si può definire come un insieme di *vincoli* su un insieme di *variabili*, dove ciascuna variabile ha un insieme di valori che può assumere detto *dominio*. Ogni vincolo specifica gli assegnamenti validi per un sottoinsieme di variabili. Un *Problema di Soddisfacimento di Vincoli Temporal* (**TCSP** Temporal Constraint Satisfaction Problem) è un particolare CSP in cui le variabili rappresentano istanti e i vincoli rappresentano gli insiemi di relazioni temporali permesse tra essi. Esistono svariate classi di TCSP che si differenziano per il tipo di entità temporali che le variabili possono rappresentare, istanti assoluti, intervalli, durate, e per la classe di vincoli utilizzata, vincoli di tipo metrico, qualitativo o entrambi.

5.2 TCSP (Temporal Constraint Satisfaction Problem)

Come già accennato esistono svariate tipologie di TCSP; esistono tuttavia caratteristiche comuni ai vari formalismi per i TCSP. Le *variabili* rappresentano istanti, intervalli o durate temporali. I *domini* sono definiti su un singolo insieme la cui struttura è isomorfa o all'insieme degli "interi" o a quello dei "razionali". Questo insieme è detto *struttura temporale*. Il dominio degli istanti e delle durate è la struttura stessa, mentre il dominio delle variabili che rappresentano gli intervalli sarà l'insieme prodotto cartesiano della struttura per sé stessa. Le diverse classi di vincoli sono supportati da un insieme di relazioni temporali di base (**BTR** Basic Temporal Relations). Tutti i BTR soddisfano le seguenti 2 condizioni:

1. I loro elementi si escludono mutuamente
2. L'unione degli elementi è il vincolo universale

Tutti i vincoli di un TCSP sono *binari* e della forma $C_{ij} = \{r_1, \dots, r_k\}$ dove X_i, X_j sono variabili, $k > 0$, e $r_1, \dots, r_k \in \text{BTR}$. La loro interpretazione è:

- $(X_i \ r_1 X_j) \cup \dots \cup (X_i \ r_k X_j)$

Si possono avere vincoli metrici sui punti *unari* ma essi di solito vengono ridotti a vincoli binari. A tal fine viene introdotta una variabile speciale X_0 il cui dominio contiene un solo elemento: $D_0 = \{0\}$. Questo permette di trasformare il vincolo unario $C_i = \{r_1, \dots, r_k\}$ nel vincolo binario $C_{0i} = \{r_1, \dots, r_k\}$. La forma dei vincoli temporali è una caratteristica fondamentali dei TCSP, infatti i vincoli temporali sono descritti in termini di elementi di BTR. Questo significa che si possono trovare soluzioni manipolando sottoinsiemi di BTR e non specifiche estensioni. Un vincolo esprimibile da un'unica disgiunzione viene chiamato *singoletto*. Una *etichettatura dei singoletti* di un TCSP assegna ad ogni paio di variabili X_i, X_j una relazione temporale di base r tale che r sia contenuto in C_{ij} . Le soluzioni di un TCSP sono le sue etichettature con singoletti

consistenti. La consistenza di una etichettatura con singoletti è definita secondo la semantica di ciascuna classe di TCSP. Poiché le soluzioni non sono assegnamenti a variabili, la nozione di valore ammissibile viene sostituita da quella di *relazione ammissibile*. Una relazione $r \in BTR$ è ammissibile per la coppia (X_i, X_j) se e solo se esiste una soluzione nella quale r è assegnata alla coppia.

I due problemi principali che ci si pone quando si trattano TCSP sono:

1. Deciderne la consistenza
2. Trovare la rappresentazione minima.

In questo contesto solo il primo punto ci interessa poiché studieremo la consistenza di un workflow temporale.

Tutte le tecniche che elaborano vincoli temporali utilizzano gli operatori di intersezione e composizione dei vincoli. Tra queste ricordiamo:

Imporre la consistenza locale

L'idea è di imporre un certo grado di consistenza localmente (k -consistenza $k < n$) per eliminare le etichettature non realizzabili dai vincoli del problema. Esiste la consistenza sugli archi (*arc-consistency*) e la consistenza sui cammini (*path-consistency*). La Path-consistency ha un carattere meno locale dell'altra, in quanto coinvolge tutti i cammini tra due variabili. Tuttavia è noto come sia sufficiente la 3-consistency per garantire la consistenza sui cammini. Path-consistency è molto efficace e per alcune classi di TCSP è sufficiente per decidere la consistenza. La complessità dell'algoritmo classico è $O(v^3)$ dove v è il numero delle variabili.

Metodi con ricerca

Poiché in generale i TCSP non sono trattabili, algoritmi di utilità pratica devono ricorrere a qualche tipo di ricerca. Per fare ciò si esegue un algoritmo

di ricerca con backtracking nel quale ciascun vincolo viene etichettato con una delle BTR fino a quando l'etichettatura non sia consistente. Ogni volta che si trova un'inconsistenza l'algoritmo applica il backtracking, cioè continua ad esplorare altri rami dello spazio di ricerca in maniera ricorsiva. Tuttavia per la maggior parte delle classi trattabili l'algoritmo di path-consistency è sufficiente per trovare una soluzione senza il backtracking.

5.3 Classi di TCSP

Esistono tre classi principali di TCSP:

Vincoli Qualitativi sui Punti

Le variabili rappresentano istanti e la BTR = {< , = , >}. Per questa classe sono state studiate due algebre:

1. *Algebra sui punti di Base* in cui le relazioni sono < , = , > , ?
2. *Algebra sui punti Convessa* in cui le relazioni sono < , = , > , ? , ≤ , ≥ , ∅

Vincoli Qualitativi sugli Intervalli

Le variabili rappresentano intervalli temporali e l'insieme delle relazioni temporali di base è così definito :

BTR = { precedes, after, meets, met by, overlaps, overlapped by, starts, started by, during, contains, finishes, finished by }

Vincoli puntuali metrici

I TCSP di questa classe sono costituiti da:

- Un insieme di **variabili** X_1, \dots, X_n , a dominio continuo, ciascuna delle quali rappresenta un istante, cioè un punto sull'asse temporale.
- Un insieme di **vincoli** sulle variabili.

Ogni vincolo è rappresentato da un insieme di intervalli:

- $\{ I_1, \dots, I_n \} = \{ [a_1, b_1], \dots, [a_n, b_n] \}$.

Un **vincolo unario** T_i restringe il dominio della variabile $X - i$ all'insieme di intervalli che lo compongono. Formalmente rappresenta la disgiunzione:

- $(a_1 \leq X_i \leq b_1) \text{OR} \dots \text{OR} (a_n \leq X_i \leq b_n)$

Un **vincolo binario** T_{ij} indica i valori ammissibili per la distanza $X_j - X_i$ e quindi rappresenta la disgiunzione:

- $(a_1 \leq X_j - X_i \leq b_1) \text{OR} \dots \text{OR} (a_n \leq X_j - X_i \leq b_n)$

Una rete di vincoli binari (TCSP binario) consiste in un insieme di variabili X_1, \dots, X_n e in un insieme di vincoli unari e binari. Tale rete può essere rappresentata da un *grafo orientato di vincoli*, nel quale i nodi rappresentano le variabili e un arco da "i" a "j" indica la specifica del vincolo T_{ij} , l'arco viene etichettato con l'insieme degli intervalli. Una variabile speciale X_0 viene introdotta per rappresentare l'istante in cui si fa partire il tempo al quale, da ora in poi, faremo riferimento come "inizio del mondo". Tutti gli istanti diventano così relativi a X_0 , e quindi ogni vincolo unario diventa un vincolo binario T_{0j} avente lo stesso insieme di intervalli.

Vengono definiti ora tre operatori binari sui vincoli:

Siano $T = \{ I_1, \dots, I_L \}$ e $S = \{ J_1, \dots, J_m \}$ due vincoli, cioè insiemi di intervalli di una variabile reale t :

1. **UNIONE**, $T \cup S$, ammette solo i valori che sono ammessi da almeno uno dei due vincoli:

$$T \cup S = \{ I_1, \dots, I_L, J_1, \dots, J_m \}$$

2. **INTERSEZIONE**, $T \cap S$, ammette solo i valori che sono ammessi da entrambi:

$$T \cap S = \{ K_1, \dots, K_n \}$$

Con $K_k = I_i \cap J_j$ per qualche i e j e $n < l+m$.

3. **COMPOSIZIONE**, $T \times S$, ammette solo valori r per cui esiste $t \in T$ e $s \in S$ tali che $t = s = r$:

$$T \times S = \{ K_1, \dots, K_n \}$$

dove $K_k = [a = c, b + d]$ per qualche $I_i = [a, b]$ e $J_j = [c, d]$ e $n \leq l * m$

Per alcune di queste operazioni gli intervalli risultanti non sono in forma canonica, e dunque sarà necessaria una ulteriore riduzione. Le operazioni definite seguono le rispettive sui vincoli classici. In particolare se T e S sono vincoli binari rispettivamente sulle distanze $X_j - X_i$ e $X_k - X_j$ allora $T \times S$ ammette solo coppie di valori (x_i, x_j) per le quali esiste un valore x_j tale che (x_i, x_j) è ammessa da T e (x_j, x_k) è ammessa da S.

Un vincolo binario T è più *stretto* di S, $T \subseteq S$, se ogni coppia di valori ammissibile per T lo è anche per S. Il vincolo più stretto è l'insieme vuoto. Una rete T è più *stretta* di una rete S, $T \subseteq S$, se l'ordinamento parziale \subseteq è soddisfatto da tutti i vincoli che si corrispondono, quindi per tutte le coppie i,j. Due reti sono *equivalenti* se rappresentano lo stesso insieme di soluzioni.

5.4 Consistenza di una rete

Data una rete di vincoli il primo problema che ci si può porre è quello di determinarne la consistenza. Esistono vari teoremi che riguardano la consistenza. Ad esempio Davis ha dimostrato che determinare la consistenza di un TCSP è in generale un problema NP-hard.

Teorema di Davis

- Decidere la consistenza di un TCSP è NP-hard.
- Decidere la consistenza di un TCSP con al più due intervalli per vincolo è NP-hard.

Una sottoclasse di TCSP che sono più semplici da trattare sono gli STP.

5.4.1 STP (Simple Temporal Problems)

Un TCSP in cui tutti i vincoli sono costituiti da un unico intervallo è detto *Problema a Vincoli Temporalis Semplici* (STP). Nelle reti corrispondenti a tali problemi ogni arco è etichettato con un singolo intervallo $[a_{ij}, b_{ij}]$ il quale rappresenta il vincolo:

$$a_{ij} \leq X_j - X_i \leq b_{ij}$$

In alternativa i vincoli possono essere espressi come una coppia di disuguaglianze

$$\begin{aligned} X_j - X_i &\leq b_{ij} \\ X_j - X_i &\geq a_{ij} \end{aligned}$$

e quindi risolvere un STP è equivalente a risolvere un insieme di disequazioni lineari nelle X_i .

5.4.2 Grafo delle distanze

Ad ogni STP viene associato un grafo orientato e pesato $G_d = (V, E_d)$ detto **grafo delle distanze**, da distinguersi dal grafo dei vincoli G . Esso ha gli stessi nodi di G , e ogni arco " i, j " è etichettato con un peso a_{ij} . Ogni cammino da i a j in G_d , $i_0 = i, i_1, \dots, i_k = j$ induce i seguenti vincoli sulla distanza $X_j - X_i$:

$$X_j - X_i \leq \sum_{l=1}^k a_{i_{l-1}, i_l}$$

Se c'è più di un cammino da i a j allora è facile verificare che l'intersezione di tutti i vincoli indotti dai vari cammini è

$$X_j - X_i \leq d_{ij}$$

dove d_{ij} è la lunghezza del cammino minimo tra i e j .

Sulla base di queste osservazioni sono state stabilite le seguenti condizioni di consistenza per un STP:

- Dato un STP T , esso è consistente se e solo se il suo grafo delle distanze, G_d , non ha cicli negativi. (Dal teorema di Liao e Wong)
- Sia G_d il grafo delle distanze di un STP consistente. Allora i seguenti assegnamenti sono due soluzioni:

$$S_1 = (d_{01}, \dots, d_{0n})$$

$$S_2 = (-d_{01}, \dots, -d_{0n})$$

I due assegnamenti corrispondono rispettivamente al verificarsi degli eventi il più presto possibile e il più tardi possibile.

Quindi una rappresentazione efficace di un STP si ottiene con un grafo orientato e completo, detto **d-grafo**, dove ogni arco “ da i a j ” viene etichettato con la lunghezza del cammino minimo d_{ij} in G_d .

5.4.3 Algoritmo Floyd-Warshall

Il d-grafo di un STP può essere costruito applicando l'algoritmo Floyd-Warshall al grafo delle distanze, trovando così il cammino minimo tra tutte le coppie. La complessità di tale algoritmo è $O(n^3)$, ed esso è inoltre in grado di scoprire eventuali cicli negativi semplicemente guardando gli elementi diagonali d_{ii} . Floyd-Warshall, dunque, costituisce un algoritmo polinomiale che determina la consistenza di un STP e trova i domini minimi e la rete minima. In definitiva trovare una soluzione di un STP nel caso peggiore richiede un tempo di $O(n^3)$ dovuta all'algoritmo di Floyd-Warshall.

Se abbiamo a che fare con un TCSP un modo diretto per risolverlo è quello di decomporlo in vari STP, di risolvere tali STP e di ricombinare poi le soluzioni ottenute. Dato un TCSP binario, T , definiamo una *etichettatura* di T come una scelta di un intervallo per ciascun vincolo. Ogni etichettatura

definisce un STP i cui archi sono etichettati con gli intervalli selezionati. Possiamo dunque risolvere qualunque TCSP considerando tutti gli STP che da esso si possono ottenere. In particolare un TCSP sarà consistente se almeno un'etichettatura darà origine ad un STP consistente.

Viene ora riportato lo pseudo-codice dell'algorithm di Floyd-Warshall:

```
function fw(int[0..n,0..n] graph) {
  // Initialization
  var int[0..n,0..n] dist := graph
  var int[0..n,0..n] pred
  for i from 0 to n
    for j from 0 to n
      if dist[i,j] > 0
        pred[i,j] := i
  // Main loop of the algorithm
  for k from 0 to n
    for i from 0 to n
      for j from 0 to n
        if dist[i,j] > dist[i,k] + dist[k,j]
          dist[i,j] = dist[i,k] + dist[k,j]
          pred[i,j] = pred[k,j]
  return dist
}
```

5.4.4 Algoritmi di Path-Consistency

Ai vincoli temporali, come a quelli classici, è possibile applicare la tecnica risolutiva con backtracking. Essa consiste nel determinare la soddisfacibilità di problemi di vincoli scegliendo un sottoinsieme di variabili, e determinando per ogni combinazione di valori delle variabili in tale sottoinsieme la soddisfacibilità parziale della rete di vincoli risultante, Il fatto di imporre la consistenza locale, in questo caso intesa come consistenza sui cammini, può essere una utile pre-elaborazione al fine di diminuire i passi di backtracking. L'algorithm Floyd-Warshall può essere visto come un algorithm che procede per rilassamenti successivi, in quanto ad ogni passo l'etichetta di un arco viene aggiornata secondo le etichette degli archi adiacenti. Montanari è stato il primo ad utilizzare questo algorithm ma l'approfondimento va attribuito a Mackworth che ha anche dato la seguente definizione:

- Un cammino attraverso i nodi i_0, \dots, i_m è consistente se e solo se per qualsiasi coppia di valori v_0 e v_m , tali che $v_m - v_0 \in T_{0m}$, esiste una sequenza di valori v_1, \dots, v_{m-1} tale che $v_1 - v_0 \in T_{01}$, $v_2 - v_1 \in T_{12}, \dots$, $v_m - v_{m-1} \in T_{m-1,m}$. Una rete è path-consistent se ogni cammino è path consistent.

Pseudo-codice dell'algoritmo di PC-1

```
repeat
S := T
for k:=1 to n do
  for i,j:=1 to n do
     $T_{ij} = T_{ij} \cap T_{ik} \times T_{kj}$  and
    if  $T_{ij} = \emptyset$  then exit (inconsistenza);
until S = T;
```

Questo algoritmo propaga la consistenza locale tramite l'intersezione e la composizione di vincoli. Esso considera i triangoli di vincoli e rilassa i vincoli fino a quando non si raggiunge una determinata soglia o un vincolo diventa inconsistente. La rete ottenuta con PC-1 è equivalente a quella di partenza, cioè mantiene le stesse soluzioni. Montanari ha dimostrato che nel caso di domini discreti l'algoritmo termina e la rete finale è effettivamente consistente sui cammini. Montanari inoltre asserisce che:

- Applicare PC-1 alla rete STP coincide esattamente con applicare Floyd-Warshall al grafo delle distanze (Teorema di Montanari).

Per TCSP interi, cioè i cui intervalli hanno estremi interi, PC-1 termina. Questo è dovuto al fatto che gli interi costituiscono un insieme di valori discreti. Si può concludere quindi dicendo che l'algoritmo PC-1 produce una rete consistente sui cammini.

Un'algoritmo più efficiente è il PC-2 elaborato sempre da Mackworth che sfrutta la nozione di rete minima. Infatti Montanari ha mostrato che se i vincoli soddisfano la proprietà distributiva (cioè la composizione distribuisce sull'intersezione) ogni rete path-consistent è al tempo stesso minima e

decomponibile. In tal caso è sufficiente un'unica iterazione del ciclo principale. Se consideriamo STP allora la proprietà distributiva vale e la rete elaborata è minima.

Pseudo-codice dell'algoritmo PC-2

```

Q := {(i,j,k) | (i < j) and (k != i,j)};
while Q != ∅ do
  selezionare e cancellare un cammino (i,j,k) da Q
  if  $T_{ij} \neq T_{ik} \times T_{kj}$  then
     $T_{ij} := T_{ij} \cap (T_{ik} \times T_{kj})$ 
    if  $T_{ij} = \emptyset$  then exit (inconsistenza)
     $Q := Q \cup \text{RELATED\_PATHS}((i,j,k))$ 
  end-if
end-while

```

La funzione RELATED_PATHS((i , j , k)) restituisce tutti I cammini di lunghezza 2 che devono essere riconsiderati dopo l'aggiornamento di T_{ij} .

Un terzo algoritmo, PC-3, sviluppato da Mohr-Henderson, si basa sulla nozione di supporto inteso come strutture dati in grado di memorizzare informazioni di supporto rilevanti. Viene utilizzato un contatore per ogni etichetta di un arco, "[i,j],b", che sta ad indicare il numero di etichette al nodo j che sono consistenti con l'etichetta b al nodo i. Inoltre per ogni etichetta c al nodo j, l'elemento (i,b) dell'insieme S_{jc} è l'etichetta b al nodo i che è supportata dall'etichetta c al nodo j. Infine viene utilizzata una tabella, M, per tener traccia di quali etichette sono state eliminate e da quali nodi, e una lista, List, per controllare la propagazione dei vincoli. Questo algoritmo però non sempre funziona, infatti la coppia Han-Lee [6], che ne ha scoperto la non generalità, ne ha proposto un altro, PC-4, che è la modifica dell'algoritmo di Mohr-Henderson. Nel PC-4 M è una tabella con indici [i,b,j,c], e l'insieme S_{ibjc} contiene elementi nella forma (k, d), dove le relazioni binarie R_{ik} e R_{ki} sono supportate dalla relazione $R_{ij}(b, c)$. Vengono utilizzati anche contatori con indici nella forma [(i, b, j, c), k], dove Contatore[(i, b, j, c), k] è il numero di coppie ammissibili (i, b) – (k, d) che supportano la relazione binaria $R_{ij}(b, c)$, dove d è un'etichetta ammissibile per il nodo k. Si può verificare che Contatore[(i, b, j, c), k] è essenzialmente sempre uguale a Contatore[(j, c, i, b), k]. Se entrambi sono uguali a 0 allora non ci sono valori

ammissibili per il nodo k che supportino la relazione binaria $R_{ij}(b, c)$, quindi questa relazione verrà settata a 0 (cioè falsa).

Pseudo-codice PC-4

Step 1

```

M:=0; Sibjc:=Empty_set; List:=Empty; Counter:=0;
for (I, j) ∈ E do
for k = 1, n do
  for b ∈ Ai do
    for c ∈ Aj do such that Rij(b, c) = 1 do
      begin
        Total:=0;
        for d ∈ Ak do
          if Rik(b, d) = 1 and Rkj(d, c) = 1 then
            begin
              Total:=Total + 1;
              Append(Sibkd, (j, c));
              Append(Sjckd, (I, b));
            end
          if Total:=0 then
            begin
              M[I, b, j, c] := 1; M[j, c, I, b] := 1;
              Rij(b, c) := 0; Rji(c, b) := 0;
            end
          else
            begin
              Counter[(I, b, j, c), k] := Total;
              Counter[(j, c, I, b), k] := Total;
            end
          end
        end
      initialize List with {(I, b, j, c) | M[I, b, j, c] = M[j, c, I, b] = 1 and i < j}

```

Step 2

```

while List not Empty do
  begin
    choose(k, d, l, e) from List and remove it from List;
    for (j, c) ∈ Skdle do
      begin
        Counter[(k, d, j, c), l] := Counter[(k, d, j, c), l] - 1;
        Counter[(j, c, k, d), l] := Counter[(j, c, k, d), l] - 1;

```



```

Remove (j, c) from Skdle;
Remove (k, d) from Sjcle;
if Counter[(k, d, j, c), l]=0 then
  if M[k, d, j, c]=0 then
    begin
      M[k, d, j, c]:=1; M[j, c, k, d]:=1;
      Append(List, (k, d, j, c));
      Rkj(d, c):=0; Rjk(c, d):=0;
    end
  end
end
for (j, c) ∈ Slekd do
  begin
    Counter[(l, e, j, c), k]:=Counter[(l, e, j, c), k]-1;
    Counter[(j, c, l, e), k]:=Counter[(j, c, l, e), k]-1;
    remove(j, c) from Slekd;
    remove(l, e) from Sjckd;
    if Counter[(l, e, j, c), k]=0 then
      if M[l, e, j, c]=0 then
        begin
          M[l, e, j, c]:=1; M[j, c, l, e]:=1;
          Append(List, (l, e, j, c));
          Rij(e, c):=0; Rji(c, e):=0;
        end
      end
    end
  end
end
end

```

La complessità temporale di questo algoritmo è pari a $O(n^3 a^3)$ dove $a \geq |A_i| = |A_j| = |A_k|$, con A_i = insieme di valori che può assumere la variabile i .

Analizzando il PC-4 si può affermare che non si possono cancellare i valori dall'insieme ammissibile dei nodi se ci sono alcuni cammini inconsistenti. Tuttavia si possono rimuovere le relazioni inconsistenti impostandole a zero. L'errore del PC-3 era appunto quello di eliminare valori dall'insieme di un nodo appena si trovava un cammino inconsistente, questo problema è stato risolto nel PC-4 con la rimozione delle relazioni inconsistenti [6].

Un ulteriore algoritmo proposto da Bliet e Sam-Haroud chiamato PPC (partial path consistency) [12] migliora gli algoritmi di path consistency visti fino ad ora. Questo algoritmo sfrutta la triangolazione di una rete di vincoli,

infatti prima il grafo viene triangolarizzato e poi si applica la PPC. Con questa tecnica gli archi che ogni volta devono essere modificati nella PPC sono molto meno numerosi rispetto agli archi modificati da una delle quattro PC.

Pseudo-codice PPC

```

Dato un grafo  $G(V, E)$ 
Metto in  $Q$  tutti gli elementi di  $E$ 
Until  $Q$  is empty do
   $q = \text{Deque}(Q)$ 
  for  $(v_i, v_j, v_k) \in \text{Related-triplets}(q)$  do
     $C_{v_i, v_j} = C_{v_i, v_j} \cap (C_{v_i, v_k} \times C_{v_k, v_j})$ 
    If  $C_{v_i, v_j}$  has changed then
      Enqueue  $((v_i, v_j), Q)$ 
    End
  End
End
End
End

```

Un ulteriore miglioramento è stato introdotto dalla coppia Xu e Choueiry nel 2003 con l'algoritmo Δ -PPC [12]. Con questo approccio si migliora il PPC poichè i lati del triangolo vengono considerati e aggiornati tutti allo stesso tempo a differenza del precedente dove veniva preso in considerazione un arco alla volta. Inoltre nel PPC la propagazione dei vincoli è fatta su tutti i triangoli in cui uno dei lati del triangolo originale partecipa. Questo è un lavoro inutile, dal momento che solo i triangoli in cui i lati vengono aggiornati devono essere considerati. Per questo motivo il Δ -PPC verifica molti meno vincoli e impiega meno tempo del PPC.

Pseudo-codice di Δ -PPC:

```

Begin
Consistency = True
 $G = \text{Triangulate the graph of } P$ 
 $Q_T = \text{all triangles in } G$ 
While  $Q_T$  and consistency Do
   $Q_E = \text{empty list}$ 
   $\langle i, j, k \rangle = \text{First}(Q_T)$ 

```

```

I'_{ij} = I'_{ij} \cap (I'_{ik} \times I'_{kj})
When I'_{ij} \neq I_{ij} Then I_{ij} = I'_{ij} and Enqueue (e_{ij}, Q_E)
I'_{ik} = I'_{ik} \cap (I'_{ij} \times I'_{jk})
When I'_{ik} \neq I_{ik} Then I_{ik} = I'_{ik} and Enqueue (e_{ik}, Q_E)
I'_{jk} = I'_{ij} \cap (I'_{ii} \times I'_{ik})
When I'_{jk} \neq I_{jk} Then I_{jk} = I'_{jk} and Enqueue (e_{jk}, Q_E)
When I_{ij}, I_{ij}, I_{ij} is empty Then consistency = False

When consistency
For e_{mn} \in Q_E Do
  T_{mn} = all triangles containing e_{mn}
  For <r, t, s> \in T_{mn} Do
    Unless <r, t, s> \in Q_T Then Enqueue (<r, t, s>, Q)
  QT = Remove (<i, j, k>, Q_T)
Return consistency
End

```

Un ultimo miglioramento è stato apportato nell'algoritmo chiamato P³C ideato da Planken e de Weerdts appartenenti alla Delft University of Technology in Olanda [4]. Questi autori hanno dimostrato che l'algoritmo Δ -PPC, in alcuni casi, ha una complessità quadratica nel numero di triangoli, invece con le modifiche apportate nel P³C la complessità sarà sempre lineare nel numero dei triangoli. Il P³C consiste essenzialmente in uno sweep avanti e indietro su tutti i vincoli ordinati del workflow effettuando ogni volta operazione di aggiornamento. Lo sweep in avanti è effettuato tramite l'algoritmo DPC (Direct Path Consistency). Con questo algoritmo vengono anche introdotti due teoremi:

- Teorema 1: L'algoritmo P³C realizza PPC su una STN (Simple Temporal Network) cordale.
- Teorema 2: P³C ha una complessità temporale pari a $\Theta(t)$ dove t è il numero di triangoli nel STN cordale. Se lo schema è inconsistente viene scoperto in $O(t)$.

Pseudo-codice P³C:

Input: Un STN cordale $S=(V,E)$ con ordine di eliminazione $d=(v_n, \dots, v_1)$

Output: Rete PPC di S o inconsistenza

```
call DPC(S,d)
return INCONSISTENT if DPC did
for k = 1 to n do
  forall i,j < k such that (i,k), (j,k) ∈ E do
     $w_{ik} = \min(w_{ik}, w_{ij} + w_{jk})$ 
     $w_{kj} = \min(w_{kj}, w_{ki} + w_{ij})$ 
  end
end
return S
```

Pseudo-codice DPC

Input: STN $S=(V,E)$ e un ordinamento $d = (v_n, \dots, v_1)$

Output: Consistenza o inconsistenza

```
for k = n to 1 do
  forall i,j < k such that (i,k), (j,k) ∈ E do
     $w_{ij} = \min(w_{ij}, w_{ik} + w_{kj})$ 
    if  $w_{ij} + w_{ji} < 0$  then
      return INCONSISTENT
    end
  end
end
return CONSISTENT
```

Ora vengono riportati alcuni grafici che mostrano il confronto tra alcuni algoritmi proposti in alcune situazioni limite [4].

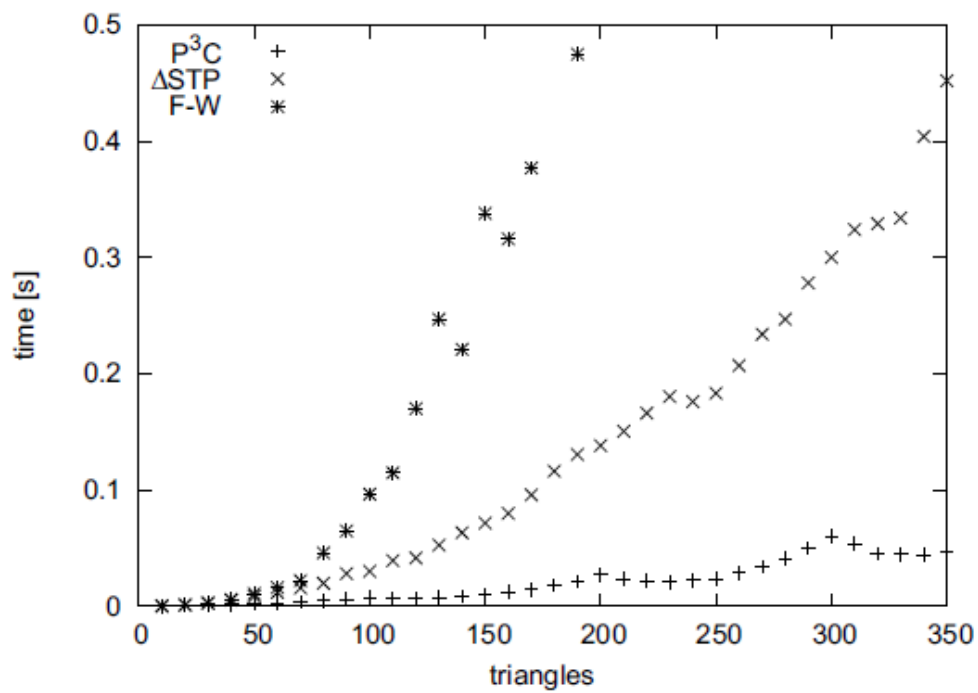


Figura 5.1 Performance degli algoritmi Floyd-Warshall, Δ -PPC e P^3C in casi patologici

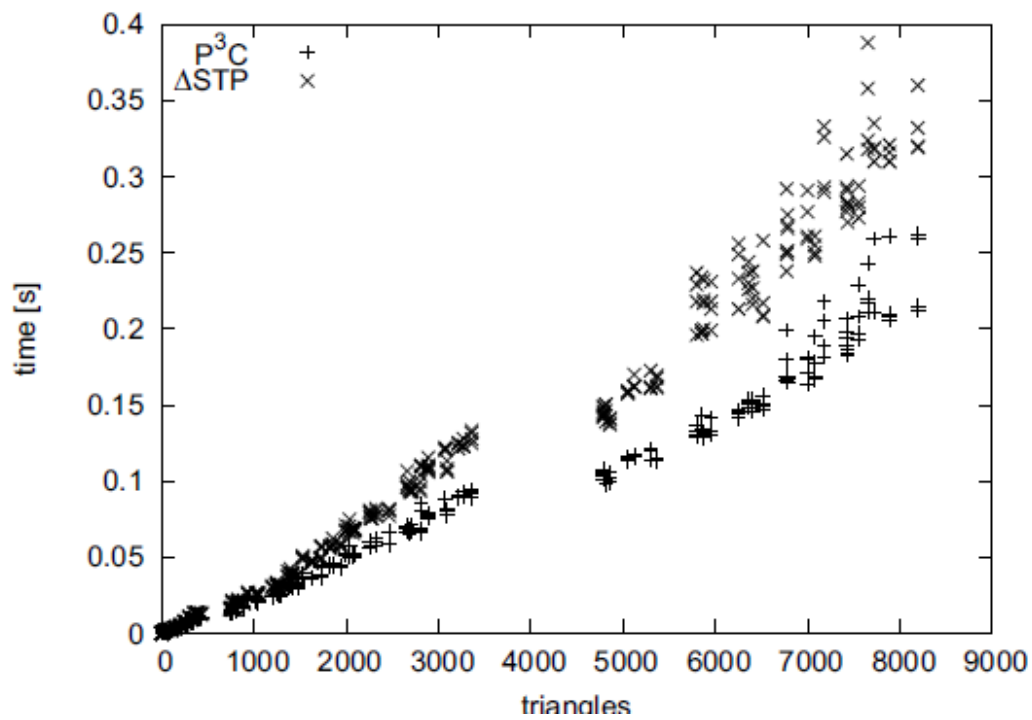


Figura 5.2 Performance di Δ -PPC e P^3C in esempi casuali

5.5 Implementazione del controllo di consistenza in YAWL

In questo paragrafo verrà descritto il lavoro implementativo in Java per permettere il controllo di consistenza a tempo di progettazione e non a tempo di esecuzione come era possibile in precedenza con YAWL. L'inserimento di questa opzione è importante per non perdere tempo in inutili esecuzioni di workflow temporali che non porterebbero l'esecuzione a termine. Ho scelto di permettere il controllo di consistenza in ogni momento della progettazione e non solo a workflow temporale finito, questo per permettere, in caso di inconsistenza, di scoprire e correggere subito la causa dell' inconsistenza senza dover aspettare la conclusione della progettazione. Ho scelto di verificare la consistenza con due diversi algoritmi descritti in precedenza, Floyd-Warshall, uno dei primi nati per questo scopo, e l'algoritmo P³C che è l'ultimo in ordine di tempo. Per il controllo di consistenza, oltre alla modifica di alcune classi già presenti in YAWL, sono state aggiunte tre nuovi file: Consistenza.java, VettoreConsistenza.java, Sequenza.java. Il file Consistenza.java è il cuore del controllo di consistenza, in questo file vengono effettuate tutte le operazioni necessarie al soddisfacimento del problema da me affrontato, VettoreConsistenza.java è una classe che permette l' utilizzo dei *singleton* , che verranno spiegati successivamente, sulla classe Consistenza, infine il file Sequenza.java è stato creato per poter dare un identificativo univoco ad ogni task visto che in YAWL non era molto chiaro come venissero distinti i vari task.

5.5.1 I singleton in Java

L'utilizzo di questo "strumento" di Java è stato reso obbligatorio dalla necessità di creare un'istanza della classe Consistenza che fosse visibile e

utilizzabile da tutte le classi di YAWL, l'unico modo per fare ciò in Java è l'utilizzo dei singleton.

Il Singleton è un pattern fondamentale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza. L'implementazione più semplice di questo pattern prevede che la classe *singleton* abbia un unico costruttore privato, in modo da impedire l'istanziamento diretta della classe. La classe fornisce inoltre un metodo "*getIstance()*" statico che ritorna una istanza della classe (sempre la stessa), creandola preventivamente o alla prima chiamata del metodo, e memorizzandone il riferimento in un attributo privato anch'esso statico. Un ulteriore approccio si può classificare come basato sul principio della *lazy initialization* (letteralmente "inizializzazione pigra") in quanto la creazione dell'istanza della classe viene rimandata nel tempo e messa in atto solo quando ciò diventa strettamente necessario (al primo tentativo di uso). Ciò che è stato fatto è creare una classe *VettoreConsistenza*, che implementa *YAWLBaseAction*, composta da un oggetto di tipo *Consistenza*, che conterrà tutti i task e i vincoli del workflow temporale progettato, un costruttore privato, un metodo *getIstance()* per tornare l'istanza, e nel caso di primo accesso per crearla, un metodo *getVettore()* che ritorna l'oggetto di tipo *consistenza*, e infine il metodo *actionPerformed(actionEvent e)*, appartenente alla classe *YAWLBaseAction*, utile per richiamare i metodi di controllo consistenza quando viene richiesto il controllo tramite il menù a tendina di YAWL dove è stato inserito un nuovo campo *Consistenza*. L'*actionEvent e* è il comando per catturare il click con il mouse sull'opzione *Consistenza*, nell' *actionPerformed* vengono richiamati i metodi di controllo consistenza con Floyd-Warshall e P³C sull'oggetto consistenza.

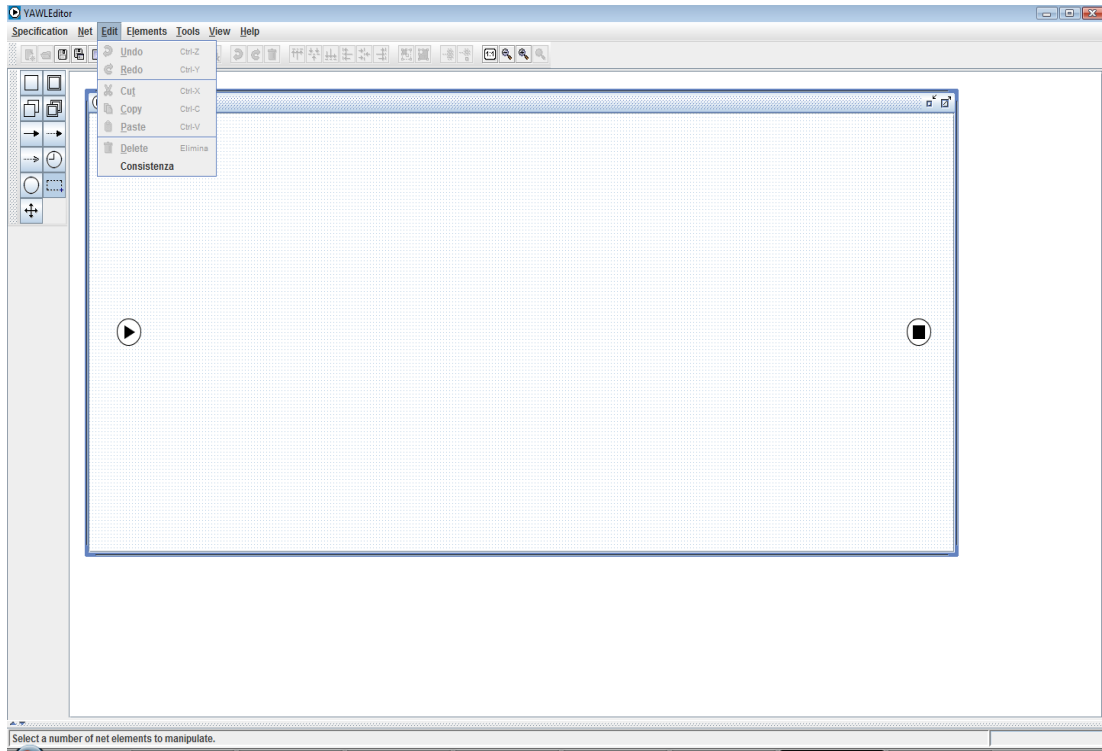


Figura 5.3 Editor di YAWL con l'inserimento nel menù a tendina di Consistenza

5.5.2 La classe Sequenza

La creazione e l'utilizzo di questa classe, contenuta in Sequenza.java, si è reso necessario per poter identificare univocamente un task, cosa che in YAWL non si riusciva facilmente a fare. Anche qui si è dovuto ricorrere all'utilizzo del singleton poichè ogni task deve avere un identificativo univoco e quindi nel corso della progettazione si deve creare un'unica istanza globale di tipo sequenza visibile a tutte le classi. Sequenza contiene, oltre al metodo *getIstance()*, un contatore che serve per assegnare un identificativo numerico ad ogni task, un vettore per contenere eventuali identificativi di task eliminati che nel caso verranno assegnati a nuovi task, e il metodo *RitornoPosizione()* che è il metodo principale che assegna al task il suo numero identificativo. Questo metodo prima controlla se ci sono numeri disponibili tra i task eliminati altrimenti ne assegna uno nuovo andando ad incrementare la variabile contatore. Per utilizzare questo identificativo è stata aggiunta alla

classe YAWLVertex una nuova variabile privata *identificatore* che viene inizializzata nel costruttore ogni qualvolta viene creato un nuovo task, inoltre è stato aggiunto anche un metodo *RitornoPosizione()* che restituisce l'identificativo del task selezionato.

```
if(i!=0){
    cont = taskEliminati[i-1];
    taskEliminati[i-1]=0;
    i--;
    return cont;
}
contatore++;
return contatore;
```

Figura 5.4 Codice del metodo RitornoPosizione della classe Sequenza

5.5.3 La classe Consistenza

Questa classe è il cuore di tutto il lavoro effettuato per permettere il controllo della consistenza, al suo interno sono contenuti tutti i metodi necessari per andare a verificare la consistenza di un workflow temporale. Di questa classe verrà creata un'unica istanza globale per ogni workflow temporale grazie all'utilizzo della classe VettoreConsistenza. Al suo interno sono contenuti anche i due algoritmi, Floyd-Warshall e il P³C, per andare a controllare la consistenza. Qui di seguito viene riportato l' interfaccia della classe Consistenza:

```
public class Consistenza {
    private class Evento { }
    private Evento eventi [];
    private int contatore;
    private FileOutputStream file;
```

```

private    PrintStream Output;
private    int maxIdTask;
private    int taskInseriti[]= new int[150];
public Consistenza ();
public Evento[] getEvento();
public int getContatore();
private    int Ricerca( String iden);
private    boolean RicercaCorda(String corde[][],String num1,String
num2);
public long [] Fuzzy(int ti,int tf,String g,int in1,int in2);
public void InserimentoFittizio(String idI,String idF);
public void InserimentoVD(int id,String vincolo);
public void InserimentoVR(int idI,int idF, String vincolo);
public void InserimentoRR(int idI,int idF,String ritardo);
public void InserimentoVF(int idIniz,int idFine,String vincolo);
public void InserimentoVA(int id,long [] dataIni,long [] dataFine);
public void EliminazioneTask(int posizione);
public void EliminazioneVincolo(String idI,String idF);
public long[] Incrocio(long a,long b,long c,long d);
public long [] MinimoTrapezi(long[] iJ,long[] iK,long[] kJ);
public void ControlloConsistenza();
public String [][] GrafoCordale();
public void ControlloConsistenzaP3C();
public void ScriviFile(String b);

```

Figura 5.5 Interfaccia della classe Consistenza

Classe Evento

La prima parte del file Consistenza.java riguarda la classe privata *Evento*, un'istanza di questa classe consiste nella descrizione di un task con tutti i vincoli a lui associati, ogni task del workflow temporale viene diviso in due eventi, uno che riguarda l'inizio del task e l'altro che prende in considerazione la fine del task selezionato. E' stata fatta questa scelta poichè i vincoli possono iniziare o finire sia all'inizio che alla fine di un task, e il vincolo di durata unisce l'inizio e la fine di un task, questo permette di considerare un task come due eventi distinti tra loro. L'identificativo dell'evento d'inizio è il

numero identificativo del task originale con l'aggiunta di "I", mentre l'evento finale sarà identificato dallo stesso numero ma con una "F" aggiunta. Per ogni istanza della classe Evento viene creata una matrice *vincoli* dove vengono memorizzati tutti i vincoli dell'evento con gli altri eventi del workflow temporale, e un vettore *eventiVincolati* dove vengono inseriti gli identificatori dei vari eventi collegati all'evento selezionato. Volendo si poteva andare a creare un vettore tridimensionale e unire il vettore e la matrice ma per una questione di utilizzo nel prosieguo del programma ho deciso di tenerli divisi. Il costruttore della classe che riceve in ingresso una stringa, che sarà l'identificatore dell'evento, va a settare l'identificatore e va ad inizializzare tutte le variabili di istanza della classe. Il metodo principale di questa classe è *InserimentoVincolo* che permette di inserire il vincolo associato nella matrice *vincoli* e va ad inserire l'identificatore nel vettore *eventiVincolati*. Tutte le tipologie di vincolo possibili verranno ricondotte ad un unico vincolo ma questo verrà fatto tramite i metodi della classe Consistenza.

```
private class Evento {
    private String id;
    private long tempo [];
    private long vincoli[][];
    private String eventiVincolati[];
    private int i;
    private Evento ( String a );
    public String getId();
    public String [] getEventiVincolati();
    public int getI();
    private void InserimentoVincolo(long [] vincolo,String eventovinc);
    private void InserimentoTempo(long []vincolo);
    public long[][] getVincoli();
```

Figura 5.6 Interfaccia della classe Evento

I Vincoli

Il secondo passo importante dopo la creazione della classe Evento è stato quello di catturare i vincoli che l'utente inserisce tramite l'editor, per fare ciò si è reso necessario modificare le classi di YAWL che gestivano i vari inserimenti dei vincoli. I file, e quindi le classi interessate dalla modifica sono state:

- DurationElementDialog per i vincoli di durata
- RelativeConstraintDialog per i vincoli relativi
- DurationFlowElementDialogT per i ritardi richiesti
- DurationFlowElementDialog per i vincoli di flusso
- AbsoluteConstraintAction per i vincoli assoluti

In tutte e cinque le classi la modifica è avvenuta nel metodo *CommitValues* e di seguito viene riportato un frammento del codice per la cattura del vincolo di durata di un task. Le modifiche per gli altri vincoli sono simili alla sottostante.

```
String val = value;
YAWLTask task = (YAWLTask) getVertex();
int id = task.RitornoPosizione();
VettoreConsistenza.getInstance().getVettore().InserimentoVD(id, val);
```

Figura 5.7 Frammento di codice per la cattura del vincolo di durata

Per ogni tipologia di vincolo è stato creato un metodo ad hoc nella classe Consistenza, in ogni metodo vengono passati gli identificatori dei task interessati e il valore del vincolo tramite stringa di caratteri, in questa stringa oltre al valore verrà specificata la granularità e l'incertezza del vincolo stesso. Poiché la fuzzificazione dei vincoli veniva fatta a tempo di esecuzione si è reso necessario creare un metodo *Fuzzy* che ricevuto in ingresso tutte le informazioni necessarie restituisce un vettore di quattro elementi che contiene i valori dei vertici del trapezio corrispondente al vincolo fuzzificato. Per la fuzzificazione si è utilizzata la formula descritta nei capitoli precedenti.

```

if(g.equals(""+'d')){
    valore[3] = (((tf-ti)*24*60*60)/5)*in2/100)+(tf*24*60*60);
    valore[1] = (ti*24*60*60);
    valore[2] = tf*24*60*60;
    valore[0] = (ti*24*60*60)-(((tf-ti)*24*60*60)/5)*in1/100);
}

```

Figura 5.8 Frammento di codice del metodo Fuzzy

Eliminazione task o vincoli

Un ulteriore problematica riguarda l'eliminazione di task o vincoli da parte dell'utente durante la progettazione tramite il pulsante Delete. Quando viene utilizzata questa funzione il programma deve risalire ai task interessati e deve eliminare nel vettore Consistenza ciò che l'utente ha cancellato. Per prima cosa è stato completamente modificato il metodo *actionPerformed* della classe *DeleteAction* andando a controllare che cosa effettivamente viene eliminato, se un task o un vincolo, e nel caso di vincolo a quale tipologia appartiene. Nella classe Consistenza sono stati creati due metodi: *EliminazioneTask* e *EliminazioneVincolo* per togliere dal vettore Consistenza ciò che è stato eliminato. Nel caso di eliminazione di un task oltre ad eliminare i due eventi a lui associati e tutti i vincoli propri bisogna andare ad eliminare anche i vincoli appartenenti ad altri task in cui il task eliminato è presente. Ora vengono mostrate alcune modifiche effettuate al metodo della classe *DeleteAction*, mentre i due metodi della classe Consistenza saranno aggiunti in allegato insieme a tutto il file Consistenza.java. Viene mostrato come riconoscere se l'eliminazione riguarda un task o un vincolo di flusso, per gli altri vincoli basta effettuare piccole modifiche nel controllo dei dati eliminati dall'utente.

```

if (graph != null) {
    try{
        container=(VertexContainer) graph.getSelectionCells()[0];

```

```

        task = container.getVertex();
        Sequenza.getIstance().TaskEliminato(task.RitornoPosizione());
    }catch (Exception exc) {

try{
if(graph.getSelectionCells()[0].getClass().getName().contains("YAWLFlowRelation"))
{
flow=(YAWLFlowRelation)graph.getSelectionCells()[0];
tempoI=flow.getSourceVertex().RitornoPosizione();
tempoF=flow.getTargetVertex().RitornoPosizione();
VettoreConsistenza.getIstance().getVettore().EliminazioneVincolo(""+tempoI+"I", ""+tempoF+"I");
}
....

```

Figura 5.9 Frammento di codice del metodo *actionPerformed* di *DelectAction*

5.5.4 Implementazione dell'algoritmo Floyd-Warshall

Come già descritto in precedenza, Floyd-Warshall viene applicato al grafo delle distanze, trovando così il cammino minimo tra tutte le coppie. La complessità di tale algoritmo è $O(n^3)$, ed esso è inoltre in grado di scoprire eventuali cicli negativi semplicemente guardando gli elementi diagonali d_{ii} . Solitamente questo algoritmo viene poco utilizzato con intervalli temporali e meno ancora con workflow temporali fuzzificati, per questo ho dovuto riadattare l'algoritmo al mio caso specifico per renderlo funzionale. Per l'implementazione di questo algoritmo ho creato il metodo *ControlloConsistenza()* nella classe *Consistenza*. La prima operazione è quella di creare una matrice delle distanze e andarla a riempire in modo conforme ai vincoli inseriti dall'utente nel progetto del workflow. La matrice viene completata andando ad inserire infinito dove non esiste alcun vincolo. Il secondo e fondamentale passo è quello di calcolare tutte le distanze possibili tramite l'utilizzo di tre cicli *for*, questo porta l'algoritmo ad avere una complessità cubica. Per calcolare le distanze ho creato un metodo

MinimoTrapezi() che prende in esame tutti i casi possibili che si verificano per l'intersezione tra due trapezi fuzzificati. Il terzo e ultimo passo è controllare la diagonale della matrice calcolata verificando se ci sono cicli negativi che riconoscono quindi un'inconsistenza del workflow creato. Di seguito viene riportata la parte principale del codice per l'implementazione di Floyd-Warshall, mentre il metodo *MinimoTrapezi()* viene aggiunto in allegato vista la sua lunghezza.

```

...
for(k=0;k < contatore ; k++)//Calcolo di tutte le distanze
    for (i=0; i < contatore ;i++)
        for (j=0; j < contatore; j++){
            if((distanza[i][j][3]==-2) || (distanza[i][k][3]==2) ||
                (distanza[k][j][3]==-2)){
                for(w=0;w<4;w++)
                    distanza[i][j][w] = -2;
            }
            else {
                risultato = MinimoTrapezi(distanza[i][j],
                    distanza[i][k] , distanza[k][j]);
                mantieni=distanza[i][j][3];
                for(w=0;w<4;w++)
                    distanza[i][j][w] = risultato[w];
            }
            if(distanza[i][j][3]==mantieni)
                predecessore[i][j] = predecessore[k][j];
        }
for(i=0;i<contatore;i++){
    if((distanza[i][i][3] < 0)&&(distanza[i][i][3]!=-900000000)){
        consistente = false;
    }
}...

```

Figura 5.10 Frammento di codice per l'implementazione di FW

Viene anche riportato una schermata dell'editor di YAWL con l'utilizzo del controllo di consistenza.

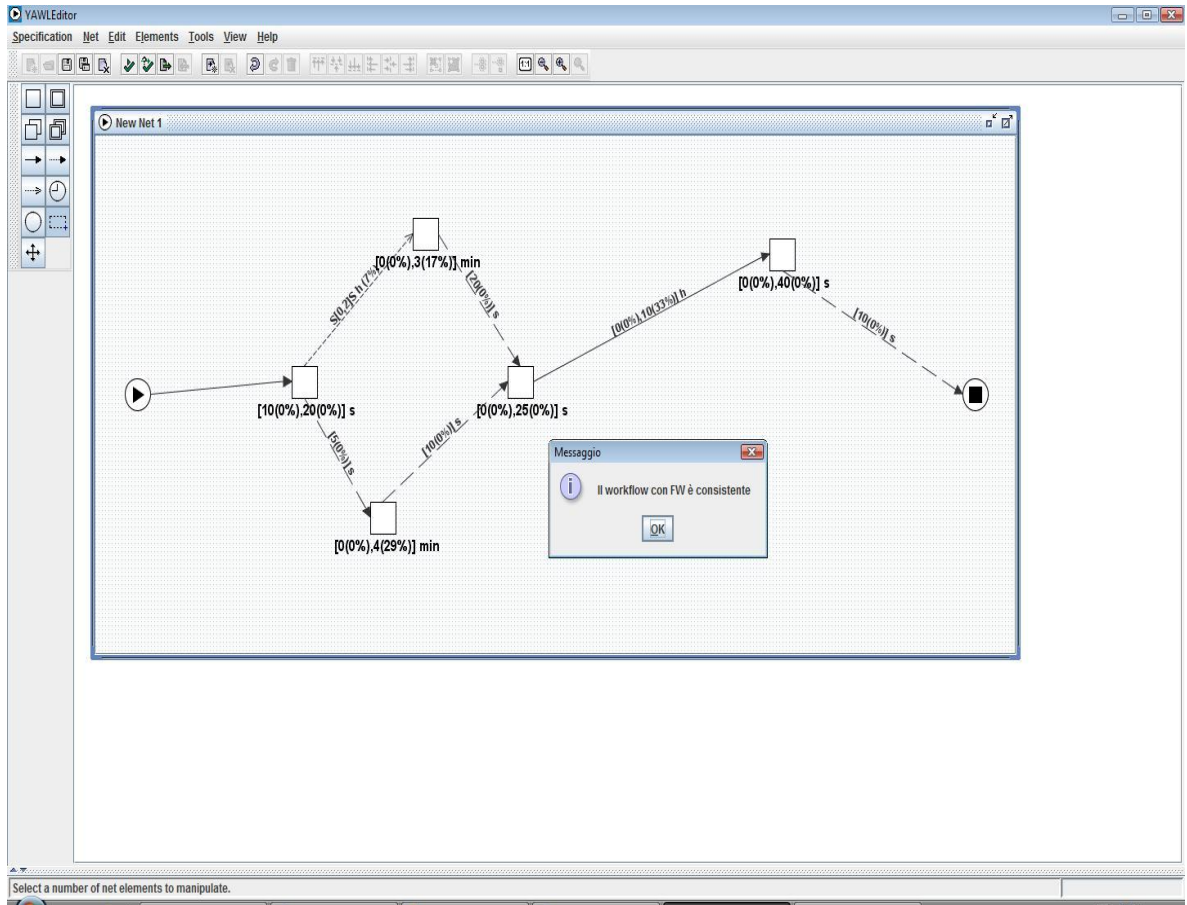


Figura 5.11 Esempio di un controllo di consistenza effettuato con Floyd-Warshall

5.5.5 Implementazione dell'algoritmo P³C

La scelta di implementazione è caduta su questo algoritmo poichè è l'ultimo e il più efficiente. La prima cosa che questo algoritmo richiede è di inserire un grafo cordale, quindi la prima operazione da compiere è quella di trasformare il workflow temporale in un workflow temporale cordale.

Cordalità

Un grafo si dice cordale se ogni ciclo di lunghezza maggiore o uguale a quattro ha una corda, che è un arco fra due nodi non adiacenti nel ciclo. Quindi per forzare la cordalità, quando c'è un ciclo di lunghezza maggiore o uguale a quattro, si aggiunge un nuovo vincolo che deve essere sempre soddisfatto. Un modo per rendere cordale un grafo è quello di generare il grafo indotto, si sceglie un ordine fra le variabili, per ogni variabile, si unisce ogni coppia di variabili che la precedono nell'ordine e che sono collegate ad essa, quello che si ottiene è un grafo cordale perchè i cicli "minimi" hanno tutti lunghezza tre (un nodo più due che lo precedono). Per rendere il workflow cordale ho creato nella classe Consistenza un metodo *GrafoCordale()* che ritorna una matrice di stringhe $n \times 2$ in cui sono contenute le corde da aggiungere al grafo originale per renderlo cordale. La prima operazione che viene effettuata dal metodo è quella di creare una matrice che contenga tutti i collegamenti dati dai vincoli del workflow, con l'aggiunta di tutti i vincoli duali. Completata la matrice si verifica se qualche evento è collegato a due eventi che lo precedono in ordine numerico, se ciò avviene bisogna aggiungere una corda, tra i due eventi trovati, al workflow temporale. Dopo l'aggiunta di questa corda si ricomincia il controllo dal maggiore dei due eventi poichè a lui è stato aggiunto un evento minore. La complessità temporale di questo algoritmo è nel caso peggiore n^2 cioè è un $O(n^2)$. Il codice per forzare la cordalità è aggiunto in allegato vista la notevole lunghezza.

P³C

L'implementazione di questo algoritmo è stata effettuata nel metodo *ControlloConsistenzaP3C* della classe Consistenza. La prima operazione consiste nel richiamare il metodo *GrafoCordale* per rendere il workflow cordale, le corde vengono inserite nel workflow originale tramite il metodo *InserimentoFittizio* che ricevendo in ingresso i due identificativi degli eventi

vertici della corda da aggiungere, inserisce un vincolo al workflow originale. Ovviamente questa operazione verrà eseguita non sul vettore originale ma bensì su una copia visto che alla fine della procedura il vettore dovrà rimanere inalterato. Il vincolo aggiunto va da zero a infinito in modo tale da non andare a modificare il risultato del workflow originale. Aggiunte le corde si passa all'implementazione dell'algoritmo vero e proprio, per fare questo viene utilizzato anche qui il metodo *MinimoTrapezi* che permette l'intersezione normalizzata dei vari vincoli. Se viene riscontrata una inconsistenza viene memorizzato il vincolo che causa l'inconsistenza. Se ce ne sono di più viene memorizzato il primo che viene trovato e viene comunicato all'utente che così sa già dove dovrà modificare il progetto. Per sapere il numero del task basta selezionare il settaggio del label del task desiderato, confermare e apparirà una finestra con il numero del task desiderato.

Viene riportato nella figura seguente la parte più interessante riguardante l'implementazione dell'algoritmo P³C.

```

...
for(k=cordale.contatore-1;k>=0;k--){
    j=0;
    while(cordale.eventi[k].eventiVincolati[j]!=null){
        if(cordale.eventi[k].eventiVincolati[j].contains("I"))
            numero = ((Integer.parseInt(cordale.eventi[k].
                eventiVincolati[j].substring(0,cordale.eventi[k].
                eventiVincolati[j].indexOf('I'))))*2)-2;
        else numero = ((Integer.parseInt(cordale.eventi[k].
            eventiVincolati[j].substring(0,cordale.eventi[k].
            eventiVincolati[j].indexOf('F'))))*2)-1;
        x=j+1;
        if(numero<k){
            while(cordale.eventi[k].eventiVincolati[x]!=null){
                if(cordale.eventi[k].eventiVincolati[x].contains("I"))
                    numero2 = ((Integer.parseInt(cordale.eventi[k].
                        eventiVincolati[x].substring(0,cordale.eventi[k].
                        eventiVincolati[x].indexOf('I'))))*2)-2;
            }
        }
    }
}

```

```

else numero2 = ((Integer.parseInt(cordale.eventi[k].
    eventiVincolati[x].substring(0,cordale.eventi[k].
    eventiVincolati[x].indexOf('F'))))*2)-1;
if(numero2<k){
    if((minimo[numero][numero2][3]==2)||
        (minimo[numero][k][3]==-2)||
        (minimo[k][numero2][3]==-2)){
        for(w=0;w<4;w++){
            minimo[numero][numero2][w]=-2;
        }
        else {
            minimo[numero][numero2] = MinimoTrapezi(
            minimo[numero][numero2],minimo[numero][k],
            minimo[k][numero2]);
            if(minimo[numero][numero2][3]==-2){
                if(numero%2==1)
                    inconsistenza[0]=(numero+2)/2+"F"
                else inconsistenza[0]=(numero+2)/2+"I";
                if(numero2%2==1)
                    inconsistenza[1]=(numero2+2)/2+"F"
                else inconsistenza[1]=(numero2+2)/2+"I";
            }
        }
    }
    x++;
}
j++;
}
}
...

```

Figura 5.12 Frammento di codice dell'implementazione di P3C

Viene riportato anche qui una schermata dello stesso esempio di prima calcolato però con P³C.

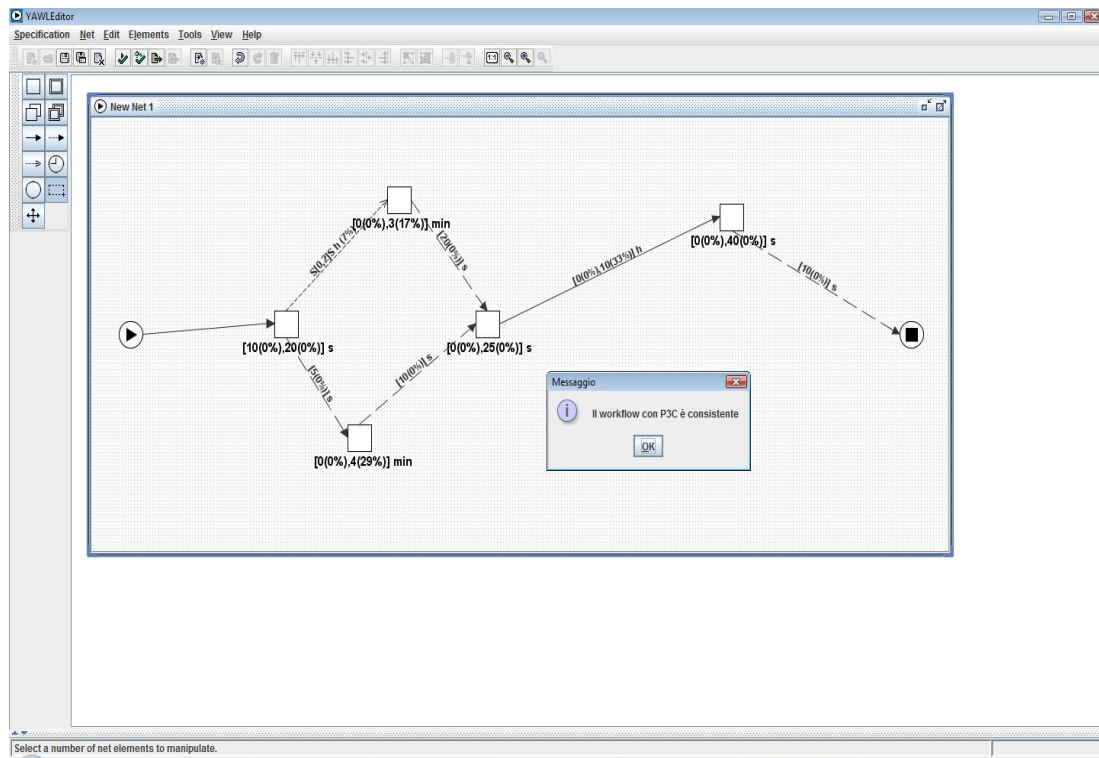


Figura 5.13 Esempio di un controllo di consistenza effettuato con P³C

5.5.6 Esempio di inconsistenza

Viene ora riportato, nelle due figure seguenti, un esempio di un workflow temporale inconsistente controllato prima tramite Floyd-Warshall e poi tramite l'algoritmo P³C. Con l'algoritmo di Floyd-Warshall verrà rilevata solo l'inconsistenza mentre con l'algoritmo P³C, oltre al rilevamento dell'inconsistenza, verrà comunicato all'utente il vincolo che causa l'inconsistenza del workflow temporale progettato.

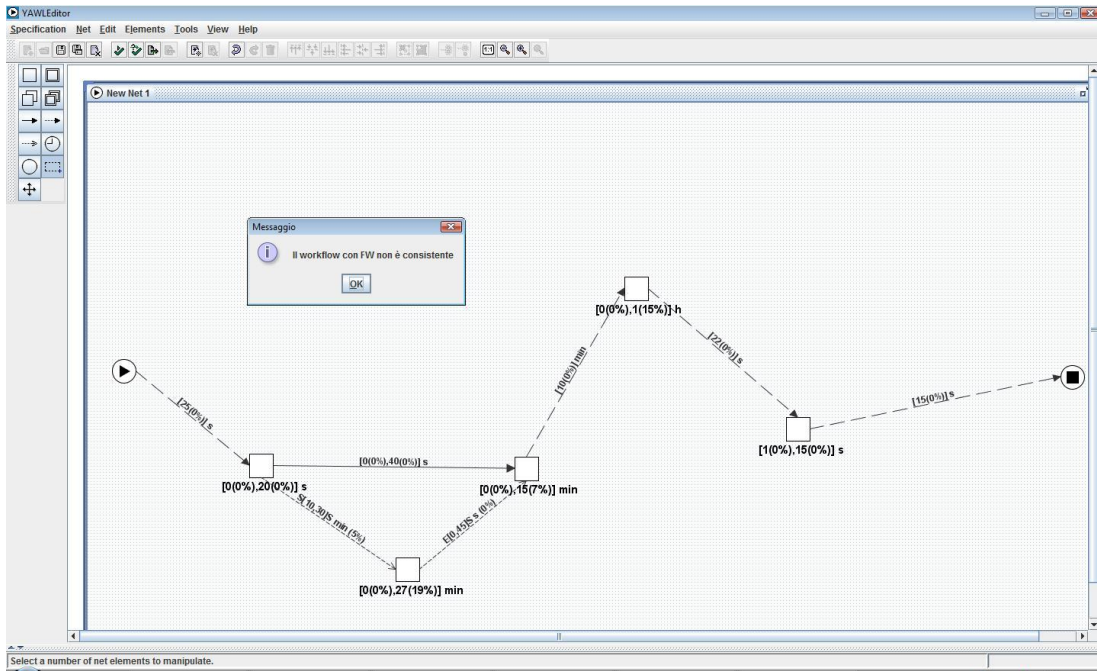


Figura 5.14 Esempio di inconsistanza rilevata tramite l'algorithmo di F-W.

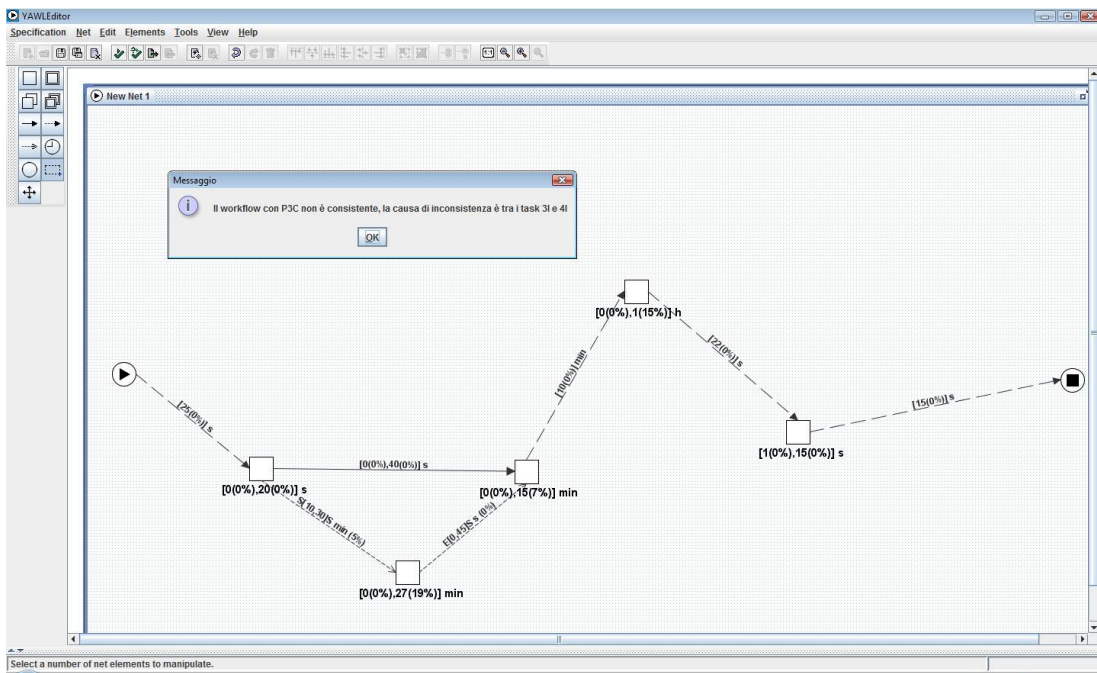


Figura 5.15 Esempio di inconsistanza rilevata tramite l'algorithmo P³C.

L'algorithmo P³C rileva la causa di inconsistanza tra il task "3I" e il task "4I" cioè tra l'inizio del task tre e l'inizio del task quattro. Per sapere il numero identificativo di un task basta selezionare il task, richiedere l'inserimento del label e dare l'ok, apparirà una finestra che comunicherà all'utente il numero

del task selezionato. Nell'esempio preso in esame il vincolo che causa l'inconsistenza è il vincolo di flusso tra il task tre e quattro. L'inconsistenza è dovuta al fatto che il vincolo di flusso ha una durata massima di quaranta secondi, mentre l'altro percorso che va dal task tre al quattro tramite il cinque richiede un tempo più elevato causando così l'inconsistenza. Nel controllo di consistenza con l'algoritmo di Floyd Warshall ho scelto di non implementare la parte riguardante la ricerca del vincolo che causa l'inconsistenza poiché l'inserimento di questo algoritmo è motivato solo dalla possibilità di confrontare i risultati ottenuti dall'algoritmo P³C per avere una conferma in più su l'esattezza del lavoro svolto.

CAPITOLO 6

Conclusioni e sviluppi futuri

In questa tesi è stato proposto e sviluppato un metodo per verificare la consistenza, a tempo di progettazione, di un workflow temporale descritto nel linguaggio YAWL. La verifica della consistenza è importante perchè permette di conoscere in anticipo se un workflow temporale è coerente o meno senza dover obbligatoriamente eseguire il workflow stesso: questo permette di risparmiare tempo e di renderne più efficiente l'esecuzione. Inoltre, la possibilità di utilizzare il controllo di consistenza in qualsiasi momento, permette di identificare subito le cause di inconsistenza senza dover necessariamente completare il progetto, permettendo così un lavoro più facile al progettista. La scelta di implementare due algoritmi che eseguono lo stesso controllo è stata fatta per verificare la bontà del lavoro svolto confrontando i risultati che i due algoritmi, Floyd-Warshall e P^3C , comunicano all'utente. L'algoritmo principale è il P^3C , la scelta è caduta su questo algoritmo per la sua miglior efficienza rispetto agli altri algoritmi dedicati al controllo di consistenza. La versione di YAWL su cui è stato sviluppato il progetto è la 1.4.4 comprensiva delle modifiche per permettere la costruzione di workflow temporali descritte nel capitolo 4. Questa versione non è l'ultima ma è l'unica sui cui sono state apportate le modifiche necessarie a

rappresentare i workflow temporali affetti da incertezza. La parte che ha creato più difficoltà per lo svolgimento della tesi è stata quella di adattare gli algoritmi al problema da risolvere e di introdurre il controllo di consistenza nel modello di workflow in maniera coerente e senza alterarne le funzionalità preesistenti, ma integrandolo in maniera trasparente anche a livello di interfaccia utente. Inoltre non è stato immediato reperire informazioni su algoritmi che costituiscono lo “stato dell’arte” della ricerca nel controllo della consistenza, perché spesso presentati a conferenze specialistiche anche diverse dall’ambito dei workflow temporale, come ad esempio conferenze su planning e scheduling.

I possibili sviluppi futuri del lavoro svolto potrebbero essere molteplici: innanzitutto si potrebbe cercare di riportare tutte le modifiche effettuate dallo YAWL originale 1.4.4 allo YAWL più recente possibile. Questo lavoro non è per nulla facile poiché le modifiche sono molte e le novità inserite sono parecchie. Un altro possibile sviluppo non ancora considerato riguarda gli aspetti temporali legati agli agenti. Le politiche di assegnamento dei task dovrebbero tener conto degli aspetti temporali, come il tempo di disponibilità degli agenti, il loro carico di lavoro, le deadline per il completamento dei task e gli altri vincoli. Anche in questo caso l'incertezza potrebbe ricoprire un ruolo importante. Un'altra direzione di sviluppo potrebbe riguardare il miglioramento dell'interfaccia grafica e della comunicazione tra editor e utente e quindi il miglioramento delle informazioni fornite riguardanti i task e i vincoli. Altri sviluppi potrebbero interessare la parte dell'engine, che in questa tesi non è stata presa in considerazione, migliorando le interfacce di comunicazione con l'utente durante l'esecuzione di un case in modo da fornire all'utente più informazioni riguardo l'andamento dell'esecuzione e fornendo addirittura dei suggerimenti sui cammini più promettenti.

Allegato

- **Metodo MinimoTrapezi della classe Consistenza.java**

```
public long [] MinimoTrapezi(long[] ij,long[] iK,long[] kJ){
long[] risultato = new long[4];
long[] somma=new long[4];
int w;
if((iK[3]==900000000) || (kJ[3]==900000000))//nel caso non esista un collegamento
    return ij;
if((iK[3]==-900000000) || (kJ[3]==-900000000)){//calcolo della somma ik+kj
    somma[0]=kJ[0]+iK[0];
    somma[1]=kJ[1]+iK[1];
    somma[2]=-900000000;
    somma[3]=-900000000;
}
else {
    somma[0]=kJ[0]+iK[0];
    somma[1]=kJ[1]+iK[1];
    somma[2]=kJ[2]+iK[2];
    somma[3]=kJ[3]+iK[3];
}
if((somma[3]==-900000000)&&(ij[3]==-900000000)){//caso in cui entrambi vanno all'infinito
if(somma[1]<ij[0]){
    risultato[0]=ij[0];
    risultato[1]=ij[1];
    risultato[2]=-900000000;
    risultato[3]=-900000000;
}
else if(somma[0]>ij[1]){
    risultato[0]=somma[0];
    risultato[1]=somma[1];
    risultato[2]=-900000000;
    risultato[3]=-900000000;
}
else if((somma[0]<ij[0])&&(somma[1]>ij[1])){
    risultato[0]=ij[0];
    risultato[1]=somma[1];
    risultato[2]=-900000000;
    risultato[3]=-900000000;
}
else {
    risultato[0]=somma[0];

```

```

        risultato[1]=ij[1];
        risultato[2]=-900000000;
        risultato[3]=-900000000;
    }
    return risultato;
}

if(ij[3]==900000000){
    risultato[0]=somma[0];
    risultato[1]=somma[1];
    risultato[2]=somma[2];
    risultato[3]=somma[3];
    return risultato;
}
if((ij[3]==-900000000)){//caso in cui uno va all'infinito
    if(somma[1]<ij[0]){
        risultato[0]=ij[0];
        risultato[1]=ij[1];
        risultato[2]=somma[2];
        risultato[3]=somma[3];
    }
    else if(somma[0]>ij[1]){
        risultato[0]=somma[0];
        risultato[1]=somma[1];
        risultato[2]=somma[2];
        risultato[3]=somma[3];
    }
    else if((somma[0]<ij[0]&&(somma[1]>ij[1]))){
        risultato[0]=ij[0];
        risultato[1]=somma[1];
        risultato[2]=somma[2];
        risultato[3]=somma[3];
    }
    else {
        risultato[0]=somma[0];
        risultato[1]=somma[1];
        risultato[2]=somma[2];
        risultato[3]=somma[3];
    }
}
return risultato;
}
if((somma[3]==-900000000)){//caso in cui uno va all'infinito
    if(somma[1]<ij[0]){
        risultato[0]=ij[0];
        risultato[1]=ij[1];
        risultato[2]=ij[2];
        risultato[3]=ij[3];
    }
}

```

```

}
else if(somma[0]>iJ[1]){
    risultato[0]=somma[0];
    risultato[1]=somma[1];
    risultato[2]=iJ[2];
    risultato[3]=iJ[3];
}
else if((somma[0]<iJ[0])&&(somma[1]>iJ[1])){
    risultato[0]=iJ[0];
    risultato[1]=somma[1];
    risultato[2]=iJ[2];
    risultato[3]=iJ[3];
}
else {
    risultato[0]=somma[0];
    risultato[1]=somma[1];
    risultato[2]=iJ[2];
    risultato[3]=iJ[3];
}
return risultato;
}
if((somma[3]<iJ[0]) || (iJ[3]<somma[0])){//inconsistente
    risultato[0]=-2;
    risultato[1]=-2;
    risultato[2]=-2;
    risultato[3]=-2;
    return risultato;
}
if((somma[0]<iJ[0])&&(somma[1]<iJ[1])&&(somma[2]<iJ[2])&&(somma[3]<iJ[3])){
    risultato[0]=iJ[0];
    risultato[1]=iJ[1];
    risultato[2]=iJ[2];
    risultato[3]=iJ[3];
    return risultato;
}
if((iJ[0]<somma[0])&&(iJ[1]<somma[1])&&(iJ[2]<somma[2])&&(iJ[3]<somma[3])){
    risultato[0]=somma[0];
    risultato[1]=somma[1];
    risultato[2]=somma[2];
    risultato[3]=somma[3];
    return risultato;
}
if((somma[1]<iJ[0])){
    if((somma[3]<iJ[2])&&(somma[3]>iJ[1])){
        risultato[0]=iJ[0];
        risultato[1]=iJ[1];
    }
}

```

```

        risultato[2]=somma[2];
        risultato[3]=somma[3];
    }
    else risultato = Incrocio(somma[2],somma[3],ij[0],ij[1]);
return risultato;
}
if((ij[1]<somma[0])){
    if(ij[3]<somma[2]&&(ij[3]>somma[1])){
        risultato[0]=somma[0];
        risultato[1]=somma[1];
        risultato[2]=ij[2];
        risultato[3]=ij[3];
    }
    else risultato = Incrocio(ij[0],ij[1],somma[2],somma[3]);
return risultato;
}
if((somma[0]<ij[0])&&(somma[1]>ij[1])){
    risultato[0]=ij[0];
    risultato[1]=somma[1];
    if(ij[3]<somma[2]){
        risultato[2]=ij[2];
        risultato[3]=ij[3];
    }
    else if(somma[3]<ij[2]){
        risultato[2]=somma[2];
        risultato[3]=somma[3];
    }
    else if((somma[3]<ij[3])&&(ij[2]<somma[2])){
        risultato[2]=ij[2];
        risultato[3]=somma[3];
    }
    else{
        risultato[2]=somma[2];
        risultato[3]=ij[3];
    }
}

return risultato;
}
if((ij[0]<somma[0])&&(ij[1]>somma[1])){
    risultato[0]=somma[0];
    risultato[1]=ij[1];
    if(ij[3]<somma[2]){
        risultato[2]=ij[2];
        risultato[3]=ij[3];
    }
    else if(somma[3]<ij[2]){
        risultato[2]=somma[2];

```

```

        risultato[3]=somma[3];
    }
    else if((somma[3]<iJ[3])&&(iJ[2]<somma[2])){
        risultato[2]=iJ[2];
        risultato[3]=somma[3];
    }
    else{
        risultato[2]=somma[2];
        risultato[3]=iJ[3];
    }

return risultato;
}
if((iJ[0]<somma[0])&&(iJ[1]<somma[1])){
    risultato[0]=somma[0];
    risultato[1]=somma[1];
    if(iJ[3]<somma[2]){
        risultato[2]=iJ[2];
        risultato[3]=iJ[3];
    }
    else if(somma[3]<iJ[2]){
        risultato[2]=somma[2];
        risultato[3]=somma[3];
    }
    else if((somma[3]<iJ[3])&&(iJ[2]<somma[2])){
        risultato[2]=iJ[2];
        risultato[3]=somma[3];
    }
    else{
        risultato[2]=somma[2];
        risultato[3]=iJ[3];
    }

return risultato;
}
else {
    risultato[0]=iJ[0];
    risultato[1]=iJ[1];
    if(iJ[3]<somma[2]){
        risultato[2]=iJ[2];
        risultato[3]=iJ[3];
    }
    else if(somma[3]<iJ[2]){
        risultato[2]=somma[2];
        risultato[3]=somma[3];
    }
    else if((somma[3]<iJ[3])&&(iJ[2]<somma[2])){

```

```
        risultato[2]=ij[2];
        risultato[3]=somma[3];
    }
    else{
        risultato[2]=somma[2];
        risultato[3]=ij[3];
    }

    return risultato;
}
}
```

Bibliografia

- [1] <http://www.yawlfoundation.org/software/documentation>
- [2] Giorgio Fersini. *Estensione fuzzy di un sistema di workflow temporale*. Tesi presso l'Università degli studi di Padova 2008.
- [3] Beatrice Lazzarini. *Introduzione agli Insiemi Fuzzy e alla Logica Fuzzy*. Dispensa 2008
- [4] Leon Planken, Mathijs de Weerd, Roman van der Krogt. *P³C: A new Algorithm for the Simple Temporal Problem*. ICAPS 2008.
- [5] Dott. Matteo Gozzi. *YAWL Workflow Management System*. Slide del Master in progettazione e gestione dei sistemi di rete (III edizione) 2008.
- [6] Ching-Chin Han, Chia-Hoang Lee. *Comments on Mohr and Henderson's Path Consistency Algorithm*. Department of Computer Science Purdue University West Lafayette, In 47907. 2008
- [7] Carlo Combi e Giuseppe Pozzi. *Temporal Conceptual Modelling of Workflows*. I.-Y.Song et al.(Eds.): ER2003, LNCS 2813, pp 59-76, 2003.
- [8] Carlo Combi e Giuseppe Pozzi. *Architectures for a Temporal Workflow Management System*. 2004 ACM Symposium on Applied Computing
- [9] Carlo Combi e Giuseppe Pozzi. *Towards Temporal Information in Workflow Systems*. A.Olivè et al.(Eds.): LNCS 2784, pp. 13-25, 2003.
- [10] Carlo Combi. *Representing Absolute Time Expression with Vagueness, Indeterminacy and Different Granularities*. AAI, 2000.
- [11] Carlo Combi, Matteo Gozzi, Jose M. Juarez, Barbara Oliboni e Giuseppe Pozzi. *Conceptual Modeling of Temporal Clinical Workflows*. AP2003-4476
- [12] Nobel Khandaker. *Performance Comparison of Path Consistency Algorithms*. 2008
- [13] Kristen Venable. *Soluzione e apprendimento automatico di vincoli temporali con preferenza*. Tesi presso l'università degli studi di Padova 2000-2001.