UNIVERSITÀ DEGLI STUDI DI MODENA
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Supporto alla gestione di dati distribuiti in applicazioni di workflow mediante tecnologia DDBMS: progetto e realizzazione

Relatore:

Chiar.mo Prof. Sonia Bergamaschi

Correlatore:

Dott. Ing. Maurizio Vincini

Tesi di Laurea di:

Nicola Fiorenzi

Anno Accademico 1997 - 98

Parole chiave:

Workflow
Database distribuiti
Replica di dati
Dynamic Ownership
Distributed Database Management System commerciale

Ai miei genitori

RINGRAZIAMENTI

A conclusione di questo lavoro desidero ringraziare le persone che, direttamente o indirettamente, lo hanno reso possibile.
Prima di tutto vorrei ringraziare la Professoressa Sonia Bergamaschi per l'aiuto fornito nella realizzazione della presente tesi.
Un ringraziamento particolare va inoltre all'Ing. Maurizio Vincini per la preziosa collaborazione e la costante disponibilità.
Un grazie, infine, va anche alla Dott.ssa Alessandra Garuti per l'introduzione al Visual C++ e al signor Livio Laurora per la disponibilità e l'aiuto nella configurazione delle postazioni di SQL Server.

Vorrei poi spendere qualche parola di ringraziamento per le persone che mi hanno accompagnato in questo lungo viaggio che mi ha portato alla laurea. Prima di tutto, ringrazio la mia famiglia per avermi sempre sostenuto in questi anni e avermi permesso di portare a termine gli studi nel migliore dei modi. Ringrazio poi i compagni dell'equipaggio per esserci sempre stati, ogni volta che c'era da fare pausa, uscire a cena, fare baracca. In particolare, grazie al capitano T. J. Kirk Antonio Antani; al sottoufficiale Pippo Sulu, con cui ho avuto un imponente rapporto epistolare (elettronico); al dottor Carullo Max McCoy Bavaglino; alla nemica Jessy Chessy Klingon; a Katia Ura; a Massi Cerbiattone-Sorriso-Tintinnante, il più bell'uomo che l'ingegneria ricordi e nostro maestro di vita; a Scotty Devil "Toxic" Denis; al M.I.B Tiziano; al mitico help on line S.H.L e ai "DOH" che mi hanno aiutato ad andare avanti nei momenti difficili. Un grosso grazie all'intero ufficio dottorandi per la simpatia e l'ospitalità; ad Alberta, compagna di tesi, mouse contro mouse, per quasi sei mesi; a Sparc20 che ha sempre funzionato egregiamente. Ringrazio inoltre, la Scossa Shock Band per le serate di musica ed il Leader per avermi riportato a suonare il R&R; la squadra di volley per le partite e le serate in amicizia al Goblin.
Ringrazio infine Fra per essermi stata accanto, sempre e comunque, in tutti questi anni.
Se qualcuno manca, è solo perchè non ho memoria sufficiente per ricordare tutti.

Indice

Elenco delle figure

Introduzione

Un'azienda che voglia affrontare competitivamente il mercato, non può ignorare le spinte propulsive delle nuove tecnologie, soprattutto quelle dell'informatica. Questa, infatti, propone soluzioni nuove a vecchi problemi, aumentando le capacità competitive delle aziende e rimodellandole nella struttura. È difficile dire dove finiscano i bisogni reali e inizino invece quelli fittizi creati dal mercato, certo è che le aziende si devono confrontare criticamente con un mondo che evolve sempre più verso soluzioni che tentano di sfruttare la tecnologia informatica in ogni settore.

L'informatica è entrata e penetrerà con sempre più forza in azienda, non solo nell'area della produzione, dove già da tempo lavorano macchinari guidati da computer, ma un po' in tutti i settori. In ambito gestionale, ad oggi, l'informatica è spesso ridotta ai sistemi di videoscrittura, ai fogli di calcolo, ai database per gestire archivi di vario tipo. Manca quindi, rispetto alla funzione di produzione, un'automazione spinta dei processi gestionali, che faciliti il lavoro di gruppo, sfruttando al meglio le potenzialità di computer, di reti locali sempre più economiche, ottimizzando procedure e risparmiando lavoro, tempo, carta. Le più recenti proposte in questo campo vengono dal settore dei *sistemi di workflow*, che si presentano come un'ottima opportunità per un salto di qualità nella gestione dei processi aziendali.

In ambito aziendale sono molte le attività che richiedono, per essere portate a termine, il succedersi di più passi e il coinvolgimento di più attori con compiti specifici: idealmente un processo aziendale dovrebbe fluire dall'inizio alla fine, nel minor tempo possibile, col minimo sforzo, passando da un agente al successivo fino al completamento. I sistemi di workflow propongono l'automazione dei flussi di lavoro implicati da questi processi, sfruttando i supporti informatici. I vantaggi che un sistema di workflow può portare sono principalmente tre:

- aumento dell'*efficienza*, che porta a minori costi o ad una più alta capacità di carico di lavoro;

- aumento del *controllo*, risultante dalla standardizzazione delle procedure;
- aumento della capacità di *gestione* dei processi: i problemi di performance vengono evidenziati meglio e compresi, e c'è in generale una riorganizzazione dei processi stessi.

La promessa del workflow, quindi, è di far risparmiare alle aziende lavoro, carta, ridurre i tempi morti nell'esecuzione dei processi e migliorare la qualità dei processi stessi attraverso la standardizzazione delle procedure (molto importante in aziende che vogliono avviarsi alla certificazione dei propri processi).

D'altra parte, però, i sistemi reali non mantengono appieno ciò che promettono, soprattutto a causa della relativa novità di questo settore, ancora troppo "giovane" e privo di standard fissati ed accettati. I prodotti commerciali presentano difetti in vari ambiti, ma le carenze più importanti riguardano la gestione dei dati e il supporto per ambienti distribuiti. In particolare, nella gran parte dei sistemi di workflow, non è chiara la distinzione fra dati di controllo del flusso e dati dell'applicazione implementata, non si lascia il giusto spazio alla definizione di dati dell'applicazione, mancano adeguati supporti di modellazione dei flussi di dati e di gestione degli stessi. Inoltre, a parte rare eccezioni, i prodotti di workflow sono basati sul paradigma client-server e non offrono quindi le funzionalità, oggi così richieste, di gestione di processi distribuiti. Quest'ultima mancanza è grave e limitante per questo tipo di sistemi, soprattutto in una realtà dinamica come quella odierna, in cui sempre più le aziende tendono a decentralizzare determinate funzioni e a favorire l'elaborazione locale dei dati. Si distinguono in questo ambito i sistemi di gestione di basi di dati distribuite (DDBMS), che tramite la replica dei dati riescono a fornire il supporto informativo necessario ad aziende che sono divise in dipartimenti ubicati in luoghi diversi.

Da queste osservazioni trae spunto il presente lavoro di tesi, che si ripropone di integrare le carenze dei sistemi di workflow tradizionali utilizzando una tecnologia già affermata qual'è quella dei DDBMS. Si propone infatti il progetto e la realizzazione di un sistema di workflow che supporti l'elaborazione distribuita e sia adatto ad applicazioni data-intensive, in cui cioè sono i dati utilizzati ad essere il cuore del processo aziendale. Il progetto e la realizzazione del sistema sono basati su due sistemi commerciali, uno di workflow e un DDBMS, per dare maggiore concretezza all'intero lavoro. La proposta, che si inserisce in un ampio filone di ricerca, pone quindi particolare attenzione al trattamento dei dati in applicazioni di workflow distribuito, utilizzando un modello di replica, chiamato Dynamic Ownership, sviluppato all'Università di Modena e presentato in un precedente lavoro di tesi. Questo

modello, infatti, presenta caratteristiche che lo avvicinano al workflow, di cui affronta proprio l'aspetto relativo ai dati e al loro flusso.

Il contenuto della tesi si articola nel modo seguente. Il primo capitolo è dedicato ad una panoramica sul mondo del workflow: in esso si forniscono i concetti chiave dell'approccio, si descrivono architetture e funzionalità dei sistemi attuali, si evidenziano i limiti degli stessi, si offre qualche notizia sul mercato e sulla sua possibile evoluzione. Il secondo capitolo si propone di affrontare il problema della modellazione dei processi di workflow, facendo un'analisi comparativa fra diversi approcci ad essa (gli standard, la ricerca, il mercato). Il terzo capitolo presenta il modello Dynamic Ownership, descrivendo le caratteristiche salienti, proponendo alcune modifiche evolutive e mettendolo a confronto con i sistemi di workflow. Nel quarto capitolo si offre una breve descrizione dei sistemi utilizzati nella tesi (ActionWorkflow della Action Technology Inc. e Microsoft SQL Server), cui si farà riferimento nel seguito del lavoro. Nel quinto capitolo si presenta il progetto del supporto alla gestione di dati distribuiti per applicazioni di workflow, la cui implementazione è invece descritta nel sesto capitolo.

Capitolo 1

Introduzione al Workflow

Il Workflow Management consiste nell'automazione di procedure o flussi di lavoro in cui vengono condivisi fra più partecipanti, in accordo a regole ben definite, documenti, compiti da svolgere (*task*), informazioni. I prodotti di workflow, come altre tecnologie software, si sono evoluti da diverse origini: da sistemi di gestione di immagini o di documenti, da DBMS relazionali o ad oggetti, da sistemi di posta elettronica. Le aziende che hanno lanciato i primi prodotti di workflow, hanno ideato e proposto una terminologia e delle interfacce ad hoc, spesso svolgendo un lavoro di adattamento dei loro prodotti di tipo gestionale preesistenti. Il nuovo mercato che è nato (e che è in continua e rapida evoluzione), ha proposto una notevole varietà di prodotti di workflow, tanto che è nata presto la necessità di avere una base comune a cui fare riferimento sia per i produttori di software, sia per i possibili utenti. Il tentativo più importante di colmare questo vuoto è cominciato nel 1993 con la nascita della Workflow Management Coalition (W*f*MC), ma sono in corso anche altri tentativi di standardizzazione da parte di gruppi di aziende diverse. In questo capitolo tenteremo di entrare nel mondo del workflow per spiegare che cosa è, come è fatto, come funziona, quali limiti ha e in che modo si sta sviluppando il suo mercato. Le informazioni e le notizie raccolte fanno in gran parte riferimento a [16] e a [7].

1.1 Che cos'è il workflow?

In ambito aziendale sono molte le attività che richiedono, per essere portate a termine, il succedersi di più passi e il coinvolgimento di più attori con compiti specifici. Idealmente un processo aziendale dovrebbe fluire (*to flow*), dall'inizio alla fine, nel minor tempo possibile, col minimo sforzo, passando da un agente al successivo fino al completamento. Per rendere possibile

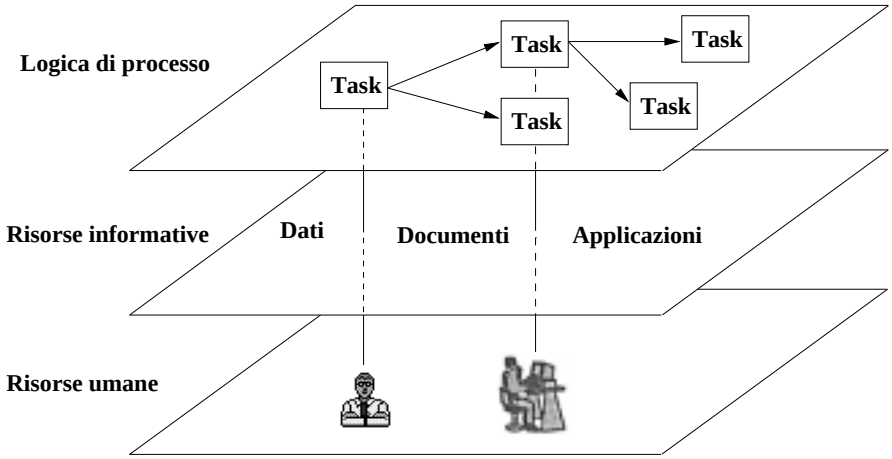


Figura 1.1: Il workflow rafforza la logica dei processi e trasporta le risorse per i task

questo flusso è necessario identificare le attività (o *task*), che compongono il processo aziendale, assegnarle ai corretti esecutori, e ricostruire poi il processo stesso come flusso di lavoro coordinato. Collegando quest'idea con le tecnologie informatiche sono nati i sistemi di gestione di workflow (WfMS), i quali forniscono un supporto software per automatizzare, ove possibile, le procedure e i processi aziendali opportunamente "rivisitati" mediante quello che è noto come *Business Process Reengineering*, (BPR).

1.1.1 La promessa del workflow

Il workflow promette di fornire una soluzione all'annoso problema della gestione e del supporto all'esecuzione dei processi aziendali. Le aziende sono motivate dalla prospettiva di risparmiare: lavoro, carta, ridurre i tempi morti nell'esecuzione dei processi e migliorare la qualità dei processi stessi attraverso la standardizzazione delle procedure. Il workflow può aiutare le aziende in un'esecuzione più efficiente e consistente dei processi.
La novità della proposta sta nel modo in cui le tecnologie informatiche vengono sfruttate per supportare il lavoro strutturato. Ciò che il workflow offre è un nuovo paradigma per la divisione del lavoro fra persone e computer. Un sistema di workflow, come mostra la Fig. 1.1:

- documenta e rafforza la logica che governa le transizioni fra i vari task all'interno di un processo;
- raccoglie e mette insieme le risorse umane e informative necessarie a portare a termine ciascun task.

I sistemi di workflow forniscono una base di controllo dei processi che permette l'unione "just-in-time" fra i task da eseguire e le persone disponibili a fare il lavoro; forniscono inoltre supporto alla progettazione (modellazione) e all'esecuzione dei processi.
Il semplice controllo dei processi non è però sufficiente: costringere i lavoratori a seguire il processo sbagliato va a danno della produttività, piuttosto che incrementarla. Le funzionalità di controllo sui processi offerte dai prodotti di workflow dovrebbero essere idealmente bilanciate da più ampie capacità di gestione dei processi stessi. Un elemento essenziale del "process management" è l'abilità di rimodellare i processi, sia per sfruttare al meglio le risorse disponibili, sia per rispondere adeguatamente a nuovi bisogni. Molti, anche se non tutti, i prodotti di workflow, offrono supporto diretto per questo importante aspetto.

1.1.2 Chi utilizza il workflow e perchè

I sistemi di workflow vengono utilizzati da molte orgnizzazioni diverse e in diversi modi. Per esempio:

- le compagnie di assicurazione utilizzano il workflow per velocizzare la gestione dei reclami mantenendone però il controllo;
- alcuni dipartimenti governativi inglesi utilizzano il workflow per aumentare l'efficienza nel prendere le decisioni riguardo i pagamenti di indennità della previdenza sociale;
- organizzazioni di tutti i tipi stanno utilizzando il workflow per aumentare l'efficacia delle loro operazioni di servizio clienti e per la gestione degli ordini;
- il workflow viene utilizzato per supportare procedure amministrative standard, come il reporting sul personale o la gestione dei reclami sulle spese;
- processi molto complessi, come i progetti di sviluppo di prodotti software imponenti, sono ad oggi supportati da sistemi di workflow.

La maggior parte delle organizzazioni sono motivate da tre fattori:

- aumento dell'*efficienza*, che porta a minori costi o ad una più alta capacità di carico di lavoro;
- aumento del *controllo*, risultante dalla standardizzazione delle procedure;

- aumento della capacità di *gestione* dei processi: i problemi di performance vengono evidenziati meglio e compresi, e c'è in generale una riorganizzazione dei processi stessi.

La domanda per prodotti di workflow negli ultimi anni sta letteralmente esplodendo. Tuttavia il workflow non è necessariamente la giusta soluzione per tutti i tipi di processi e gli stessi prodotti differiscono molto fra loro per funzionalità, condividendo invece parecchi limiti.

1.1.3 L'evoluzione del Workflow.

Sono molti i prodotti che negli anni hanno supportato funzioni ora proprie dei sistemi di workflow, ma solo oggi si riconosce l'importanza di questo mercato di IT (Information Technology), nell'evoluzione del workflow come tecnologia a se stante. Vediamo alcune delle aree interessate da prodotti di questo tipo.

- **Image Processing.** Il workflow è stato associato molto strettamente ai sistemi di Image Processing, e d'altra parte, molti di essi possono fornire funzioni tipiche dei prodotti di workflow. Molti processi aziendali coinvolgono documenti su supporto cartaceo che devono essere digitalizzati (tramite l'uso di uno *scanner*, ad esempio), per poter essere utilizzati dai computer. Una volta resi disponibili in forma elettronica (come immagini), questi vengono spesso passati fra più partecipanti e possono interagire con altre applicazioni di IT, creando così una prima forma di workflow.

- **Document Management.** La tecnologia di Document Management è incentrata sulla gestione del ciclo di vita dei documenti elettronici. Sempre più, oggigiorno, ciò significa poter offrire funzionalità di gestione di archivi di documenti distribuiti fra vari siti su cui è dislocata un'impresa. Si crea così una risorsa condivisa con tutta una serie di *utilities* per l'invio di documenti, per l'accesso ad essi in lettura o modifica a seconda dei ruoli occupati dagli utenti all'interno dell'organizzazione. Se poi il documento entra a far parte di un processo aziendale che richiede l'accesso ad esso da parte di diversi utenti che intraprendono diverse attività in accordo a particolari regole procedurali, si può a ra gione parlare di una forma di workflow *document-oriented*.

- **Electronic Mail & Directories.** I programmi di posta elettronica rappresentano un sistema potente di distribuzione di informazioni fra individui diversi all'interno di un'organizzazione o fra organizzazioni

diverse. L'uso del meccanismo dei direttori, inoltre, fornisce la possibilità di identificare gli individui all'interno di un dominio di posta elettronica e di registrare per ognuno di essi attributi come il ruolo organizzativo o altri legati a particolari processi aziendali. I sistemi di posta elettronica si sono così evoluti verso funzionalità tipiche dei sistemi di workflow, grazie all'aggiunta di meccanismi di *routing* per definire una sequenza di destinatari per particolari tipi di oggetti di posta elettronica in risposta a determinate procedure aziendali.

- **Groupware Applications.** L'industria di sistemi "groupware" ha introdotto una vasta gamma di applicazioni software per supportare e migliorare le interazioni fra gruppi di individui. Inizialmente queste applicazioni non prevedevano una formalizzazione dei processi, requisito che è però divenuto sempre più importante con l'evoluzione e l'ampliamento degli scopi di questi sistemi. Il workflow può essere una risposta a questo tipo di bisogni.

- **Transaction-based Applications.** Per molti anni determinate procedure aziendali (le transazioni), sono state sviluppate sfruttando le funzionalità offerte dai TP monitors o dai DBMS. Sempre più queste applicazioni sono divenute distribuite, per rispondere alle nuove necessità che il mercato imponeva alle imprese. Tipicamente, però, queste applicazioni pur fornendo importanti caratteristiche di robustezza e supporto per transazoni atomiche, non hanno una netta separazione fra la logica della procedura aziendale e l'invocazione di diverse applicazioni che possono essere necessarie per svolgere appieno le attività costituenti il processo. Questa considerazione sta portando verso lo sviluppo di sistemi di workflow che da una parte sfruttano consolidate applicazioni *transaction-based* per determinate parti del processo aziendale, mentre dall'altra possono invocare diverse applicazioni di IT (gestione documenti, e così via), per altre necessità del processo.

- **Project Support Software.** Il software per gestire lo sviluppo di complesse applicazioni di IT, ha spesso fornito funzionalità tipiche del workflow all'interno dell'ambiente del progetto. Prova ne è la capacità di trasferire specifici compiti dello sviluppo fra i progettisti e con essi le informazioni per supportarli. In alcuni casi questo tipo di software è stato generalizzato offrendo funzionalità di workflow più evolute e tali da permettere la gestione di progetti più ampi e complessi e da offrire una visione più generale del processo aziendale.

- **BPR and Structured System Design Tool.** Questo tipo di applicazioni comprende tutti quei prodotti software di supporto alle attività di analisi, modellazione e ridefinizione (per questo si parla di *Re-Engineering*), dei processi aziendali e alla valutazione dell'impatto sull'azienda di tali cambiamenti. Ciò significa analisi della struttura dei processi e dei flussi di informazioni necessari a supportarli, nonchè dei ruoli organizzativi che prendono parte ai processi stessi. Una naturale estensione di questi prodotti è il fornire la funzionalità di implementazione del processo aziendale con tutto il supporto IT per controllare i flussi di attività e dati all'interno dello stesso.

Il mercato di prodotti legati al workflow è, come abbiamo visto, ampio e animato da spinte provenienti da diverse aree di interesse, con una vasta gamma di software dedicato alle singole aree. Per ottenere tutte le funzionalità richieste da un sistema di workflow, quindi, è necessario che diversi prodotti software di aziende diverse possano cooperare fra loro in un'implementazione il più possibile rispondente alle singole esigenze degli utenti. Questa possibilità così ampia di scelta, perciò, si scontra con la necessità di garantire la compatibilità fra prodotti diversi, ed aumenta il bisogno di standard che regolino questo mondo così variegato.

1.2 Concetti chiave

I sistemi di workflow sono basati su quattro concetti chiave:

- logica di processo;
- risorse umane per i task;
- risorse informative per i task;
- gestione dei processi.

1.2.1 Logica di processo

La logica di processo definisce le regole che un processo aziendale deve seguire. Per esempio:

- la logica di un processo d'ordine può implicare che non vengano effettuati controlli sulle credenziali di clienti di vecchia data che facciano piccoli ordini, ma che per nuovi clienti e grossi ordini sia attivata una procedura complessa che coinvolge anche agenti esterni;

- la logica di un processo di controllo e assicurazione della qualità può implicare che il ciclo di controllo e rilavorazione siano effettuati fino a che non venga raggiunto un detrminato risultato.

Un sistema di workflow fornisce il *supporto informatico* per la logica di processo. Utilizzando un sistema di workflow, un'organizzazione può quindi rafforzare e documentare le regole che governano i propri processi. In particolare, mediano il flusso di responsabilità da persona a persona e da task a task per mezzo di alcune operazioni:

- rappresentano la definizione di ogni processo;
- tengono traccia dello stato di ogni istanza di processo durante il suo passaggio attraverso i vari step definiti;
- "spingono" il processo al task successivo da eseguire, in accordo alla logica di processo stabilita.

Il progettista del workflow definisce, utilizzando gli strumenti forniti dal sistema di workflow, quali azioni devono essere intraprese, a chi assegnarle, sotto quali condizioni eseguirle. Il sistema poi fornisce il supporto software run-time per far passare il flusso di controllo da un task all'altro, e da un lavoratore all'altro, in accordo a quanto stabilito in fase di progettazione.

Design e istanze Utilizzando un sistema di workflow, un singolo modello (*design*), di processo, può diventare la base per più *istanze* di processo. Ciò significa che a run-time possono coesistere parecchie istanze dello stesso processo (basate sullo stesso modello), in modo del tutto indipendente le une dalle altre. Le transizioni fra i task all'interno delle varie istanze sono gestite dal sistema di workflow; questo è anche in grado di gestire diversi processi, ossia di far eseguire istanze contemporanee basate su diversi modelli.

Definizione di processo: semplice o sofisticata? Il tipo di supporto che un sistema di workflow fornisce per definire i processi può essere molto semplice o piuttosto complesso e articolato. Nella sua forma più semplice, questo supporto consiste nel controllare il passaggio di consegne fra gli individui responsabili dei task in una semplice sequenza lineare, senza una logica di processo vera e propria. Per esempio, un sistema di workflow può far passare un documento elettronico fra più persone in sequenza. Il semplice passaggio di un task a qualcuno non garantisce che questi lo porterà a termine; tuttavia, per processi semplici, tale supporto può essere sufficiente ed utile. Un supporto semplice alle definizioni di processo può:

- prevenire che i processi cadano in un "buco nero" fra i passi di un task;
- ridurre considerevolmente l'ammontare di carta che viene passato di ufficio in ufficio;
- ridurre e persino eliminare la necessità di ricodificare le informazioni fra i vari passi di un processo.

I sistemi di workflow più sofisticati supportano una varietà di processi più ampi e complessi, con topologie più articolate come task paralleli, o condizionati, task composti di sottotask e persino processi che richiamano e utilizzano altri processi. Quando queste capacità sono accoppiate con l'abilità di un sistema di workflow di riconoscere il superamento di scadenze e intraprendere le azioni appropriate, le potenzialità dei sistemi di workflow cominciano ad emergere.
Ciò che un sistema di workflow può fare quando si accorge che una scadenza è stata superata, è un buon indicatore delle differenze fra i vari sistemi. Alcuni permettono al progettista di notificare ad un supervisore quando qualche istanza ha dei problemi; altri possono allocare dinamicamente le risorse per superare il collo di bottiglia che si è creato nel processo e persino modificare la definizione dello stesso per aumentare la velocità di esecuzione e la produttività.

Tipi di processi I sistemi di workflow non sono adatti per automatizzare un qualsiasi processo che un'organizzazione può avere al suo interno. Sono infatti specificatamente costruiti per processi che:

- hanno un'inizio ben definito, scatenato da un evento o da un ciclo temporale;
- hanno uno o più stadi finali;
- possono essere suddivisi in un numero definito di passi o task;
- hanno transizioni fra i task governate da regole.

Task La maggior parte dei sistemi di workflow identificano i passi necessari a completare un processo con task "atomici", cioè quell'insieme di azioni che, dal punto di vista del sistema, sono viste come un'unica attività che non può essere decomposta in sotto attività. Un task è dunque un uno step all'interno di un processo, visto ad un certo livello di granularità. Tuttavia, questa definizione è ambigua e lascia spazio a molte domande: non è infatti

Figura 1.2: La tipica struttura di un task di workflow

chiaro quale sia il confine fra un task e il successivo, o quale sia il livello di granularità più adatto per un determinato modello di processo.
La maggior parte dei sistemi di workflow hanno una idea implicita che vede il task come definito dalla congiunzione di tre criteri, come mostrato in Fig. 1.2:

- l'uso di una particolare applicazione;
- lo sforzo di una singola persona;
- un singolo e contiguo intervallo di tempo.

Quest'approccio è parte di ciò che conferisce al workflow il suo potere, ma pone anche dei limiti sul tipo di processi che possono essere supportati.
Non tutti i task necessitano di persone per essere portati a termine. Inviare una lettera, effettuare un calcolo complesso, spedire beni in formato elettronico sono tutti esempi di task effettuabili automaticamente. Alcuni sistemi permettono di creare task di questo tipo all'interno della modellazione di un processo.

1.2.2 Risorse umane per i task

I sistemi di workflow hanno la responsabilità di assicurare che i task da eseguire siano collegati alle risorse necessarie al loro completamento. Quando un task richiede l'intervento di una persona, è il sistema ad occuparsi di effettuare il collegamento lavoratore-task.
I sistemi di workflow differiscono fra loro nell'approccio al problema del collegamento fra persone e task. Nella maggior parte dei casi il sistema collega le persone con qualsiasi task pronto per l'esecuzione, perchè i task che lo precedono logicamente sono già stati completati. Il protocollo con cui viene

effettuato il collegamento varia grandemente. Alcuni prodotti di workflow utilizzano un protocollo in cui i task vengono attribuiti a individui precisi (per nome), quando arriva il loro turno di fare qualcosa che porti avanti il processo. Questo è di solito il più semplice protocollo seguito quando si utilizzano pacchetti di instradamento (*routing*), di form.
Altri tipi di sistemi offrono un protocollo più complesso che permette il collegamento *just-in-time*, (chiamato anche *late binding*), fra persone e task. In questo scenario, il motore di workflow offre un task da eseguire ad un gruppo di persone, ognuna delle quali ha le abilità ed i permessi necessari per farlo. La persona che può occuparsi del task, lo accetta e automaticamente il task viene eliminato dalle liste dei lavori (*worklist*), degli altri potenziali esecutori. Si può definire quest'approccio come *system-offer protocol*. La possibilità e la capacità di utilizzare tale protocollo risulta molto importante, soprattutto quando esso è accoppiato al sistema dei *ruoli*.

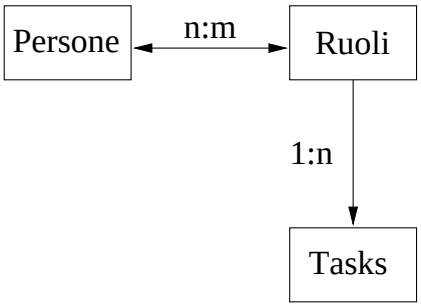


Figura 1.3: Persone, task e ruoli

Ruoli Alcuni sistemi di workflow utilizzano i ruoli per mediare il collegamento fra persone e task. I ruoli sono un costrutto di modello molto potente che introduce un grado di indirezione nel collegamento suddetto, come mostrato in Fig. 1.3.
Piuttosto che rendere fissa la relazione fra persone specifiche e i task, durante la modellazione il progettista specifica solo il ruolo della persona che dovrà eseguire un certo task. A run-time, il sistema di workflow si preoccupa di cercare le persone designate ad occupare quel ruolo.
L'utilizzo dei ruoli è diverso dall'inviare il task ad un gruppo di persone. In quest'ultimo caso, infatti, bisogna fronteggiare il problema di cosa accade dopo che il task è stato inoltrato. I riferimenti basati sui ruoli, invece, implicano un meccanismo per trasportare il riferimento al ruolo in un collegamento ben definito fra il task e l'individuo che è responsabile della sua esecuzione.
La risoluzione di questo riferimento può avvenire in due modi:

- il sistema stesso decide a chi assegnare il task, per esempio andando a rotazione fra i possibili esecutori;
- il sistema usa un protocollo di offerta per abilitare il collegamento *just-in-time* fra persona e task.

Utilizzare i ruoli per collegare le persone ai task significa che l'informazione che determina chi farà cosa è centralizzata e controllata, piuttosto che distribuita fra le scrivanie delle persone. L'uso dei ruoli, però, richiede al sistema di workflow funzionalità avanzate di reporting e gestione: cosa accadrebbe infatti se nessuno si prendesse la responsabilità di un task e questo rimanesse in attesa indefinitamente?
Un passo ulteriore, fornito da alcuni sistemi, è la possibilità di specificare, oltre al ruolo, anche altri attributi dell'insieme di persone che possono eseguire un certo task. Questa capacità è utile quando risulta importante poter assegnare il lavoro a persone in particolari luoghi, o per bilanciare il carico di lavoro fra i gruppi (redistribuendo a run-time il lavoro).

1.2.3 Risorse informative per i task

I task possono necessitare di risorse umane o informative. Quando queste ultime sono in formato elettronico, il sistema di workflow può occuparsi di assicurare ai task le giuste informazioni al momento giusto per permetterne il completamento. Ci sono veri tipi di risorse informative, ma la distinzione principale è fra risorse *applicative per i task* e risorse di *contenuto*.

Risorse applicative per i task Questo tipo di risorsa è utilizzato nei task dei processi di workflow e serve a mostrare, creare e modificare le risorse di contenuto. Queste applicazioni includono:

- applicazioni di produttività personale, come i word-processor;
- applicazioni gestionali, come i sistemi di supporto alle vendite o agli ordini;
- moduli elettronici.

La maggior parte dei sistemi di workflow scelgono fra due possibili approcci per collegare queste risorse al sistema stesso:

- forniscono un metodo per costruire semplici applicazioni utilizzando i tool del sistema di workflow;
- forniscono i mezzi per l'integrazione fra applicazioni esterne e il processo controllato dal sistema.

Risorse di contenuto Le risorse di contenuto contengono le informazioni che sono utilizzate, modificate e persino create dalle risorse applicative per i task all'interno dei processi di workflow. Esse includono:

- dati;
- documenti.

Esistono diversi approcci al collegamento delle risorse di contenuto con i task: si passa da sistemi che trascurano il problema, ad altri che fanno di questo collegamento parte integrante della progettazione/definizione dei task stessi.

1.2.4 Process management

Il process management è uno dei concetti base dei sistemi di workflow. È infatti molto più importante la capacità di gestire i processi, che quella di implementarli poi fisicamente.
Il semplice controllo sul processo, senza la capacità di miglioramenti dinamici, fornisce un guadagno di breve durata, e spesso non serve a molto. Il process management implica la capacità di modificare i processi quando necessario, sia per fare miglior uso delle risorse disponibili, sia per rispondere ai cambiamenti esterni. Per esempio:

- per risolvere un "collo di bottiglia" nella struttura del processo, può essere necessario allocare più risorse (ad esempio persone) a specifici task;
- cambiamenti nelle leggi o eventi interni (come l'introduzione di nuovi prodotti e servizi), possono comportare che le definizioni dei processi vadano rapidamente adeguate per rispondere alle nuove esigenze.

I sistemi di workflow mantengono la promessa di operare sui processi in modo completo, dal semplice controllo alla gestione vera e propria. Questo viene fatto attraverso due principali meccanismi:

- far diventare la logica di processo uno strato esplicito della rappresentazione del modello di processo, in modo da poterlo pensare ed eventualmente modificare (entro certi limiti), in modo indipendente dal resto;
- permettere ai progettisti di creare, raccogliere e valutare misure relative ai costi e alla qualità nella esecuzione di un processo e dei suoi task costituenti, per avere una base su cui operare intelligentemente dei miglioramenti al modello di processo e al sistema di risorse del processo stesso.

Alcuni sistemi permettono anche al progettista di specificare regole per l'ottimizzazione dinamica dell'ambiente di esecuzione del processo. Nella sua forma più complessa, un sistema di workflow può effettuare un bilanciamento dei carichi di lavoro di diversi gruppi di persone all'interno di un'organizzazione, e correggere dinamicamente l'allocazione di risorse per una migliore produttività.

1.3 Come funziona il workflow

1.3.1 Architettura dei prodotti

La maggior parte dei prodotti di workflow *general-purpose* attualmente disponibili sul mercato, utilizza una delle due seguenti architetture.

- Prodotti *form-based*: moduli (*form*), elettronici forniscono l'interfaccia per i task; queste form sono trasportate attraverso i vari step utilizzando meccanismi di *messaging* e la posta elettronica.

- Prodotti *engine-based*: applicazioni esterne forniscono l'interfaccia per i task. L'ambiente di esecuzione dei processi è controllato da un "motore" di workflow (*workflow engine*), che tiene traccia del progresso di ogni istanza del processo.

Ognuna delle due architetture ha dei vantaggi: i prodotti engine-based permettono una gestione dei processi più articolata, mentre i prodotti form-based forniscono il supporto per costruire (piuttosto che integrare), le applicazioni che verranno usate per compiere il lavoro. Tuttavia, la distinzione tenderà sempre più ad assottigliarsi fino a non esistere più con l'evoluzione delle due categorie di prodotti verso un comune denominatore che raccolga i vantaggi di entrambe. Nella tabella 1.1 sono evidenziati più in dettaglio i vantaggi dei due tipi di architettura.

Prodotti form-based Il tipico prodotto di workflow form-based funziona attraverso una combinazione dei seguenti processi:

- un *servizio di messaging* è utilizzato per instradare un'istanza di processo sul percorso formato dai suoi task;

- la *logica di processo* non è eseguita in modo centralizzato, ma è gestita al terminale del lavoratore tramite l'interazione fra il software di gestione delle form e il sistema di e-mail dell'utente stesso;

Prodotti form-based	**Prodotti engine-based**
Basso costo per utente	Progettati per un'alta integrazione con applicazioni esterne
Le applicazioni di supporto ai task possono essere costruite utilizzando gli strumenti del prodotto	Potenti funzionalità di tracking e gestione dei processi
Rapida prototipazione di applicazioni per i task	Buon supporto visuale alla progettazione per topologie complesse (ad esempio processi paralleli)
Molti utilizzano l'infrastruttura di messaging esistente	Utilizzo del collegamento fra persone e task basato sui ruoli

Tabella 1.1: Vantaggi dei due tipi di architettura dei sistemi di workflow

- le persone vengono collegate ai task all'inizio dell'istanza di processo e secondo la sequenza di esecuzione predefinita si vedono recapitare i task stessi al momento opportuno;
- l'interfaccia utente per ogni task è costituita da una o più form legate fra loro;
- le form possono essere collegate ad un database che memorizza informazioni mostrate nei campi delle stesse.

Tipicamente la logica del processo è collegata strettamente alle form che vengono passate fra gli utenti. La definizione della logica dei processi e dei task, spesso non ben distinte nei prodotti attuali, viene fatta tramite un insieme di *scripts*.

Prodotti engine-based Per contrasto col precedente, il tipico prodotto di workflow engine-based funziona attraverso la combinazione di:

- un *database* che memorizza le definizioni di processo;
- un *database* che memorizza lo stato di tutte le istanze di processo del sistema;
- un *servizio di esecuzione* server-based, detto workflow engine;

- un pacchetto software "client" al terminale dell'utente che comunica col server.

Tipicamente il workflow engine è responsabile per:

- l'implementazione delle regole che governano le transizioni fra i task;
- la modifica dello stato di ogni istanza di processo non appena i task sono completati e le regole valutate;
- l'offerta o la distribuzione dei task da eseguire agli utenti preposti, tramite comunicazione sul loro terminale;
- la esecuzione di azioni sul superamento di scadenze per task o processi (ad esempio segnalazione agli utenti).

Il componente "client" del sistema, posto nei terminali degli utenti, ha il ruolo di gestire localmente i task. Di solito:

- riceve notifica dei task da eseguire dal workflow engine;
- mostra i task all'utente;
- invoca le applicazioni associate ad un particolare task, quando l'utente lo seleziona;
- aggiorna il workflow engine (server), quando un task viene selezionato da un utente per l'esecuzione, cosicchè possa essere cancellato dalla lista centralizzata dei task da eseguire;
- aggiorna il workflow engine (server), quando un task viene completato, in modo che il server possa aggiornare di conseguenza lo stato dell'istanza di processo relativa.

Nei sistemi attuali la comunicazione fra il server e i client viene effettuata tramite una forma di RPC (remote procedure call), ma si sta già pensando di risolvere il problema per mezzo di un "middleware" orientato al messaging.

1.3.2 Modello di implementazione dei prodotti di workflow

Nonostante la gran varietà di prodotti sul mercato e il duplice approccio mostrato sopra, è possibile costruire un modello generale di implementazione per i sistemi di workflow che può essere rispettato dalla maggioranza di essi, fornendo una base comune per l'interoperabilità fra sistemi diversi. Questo

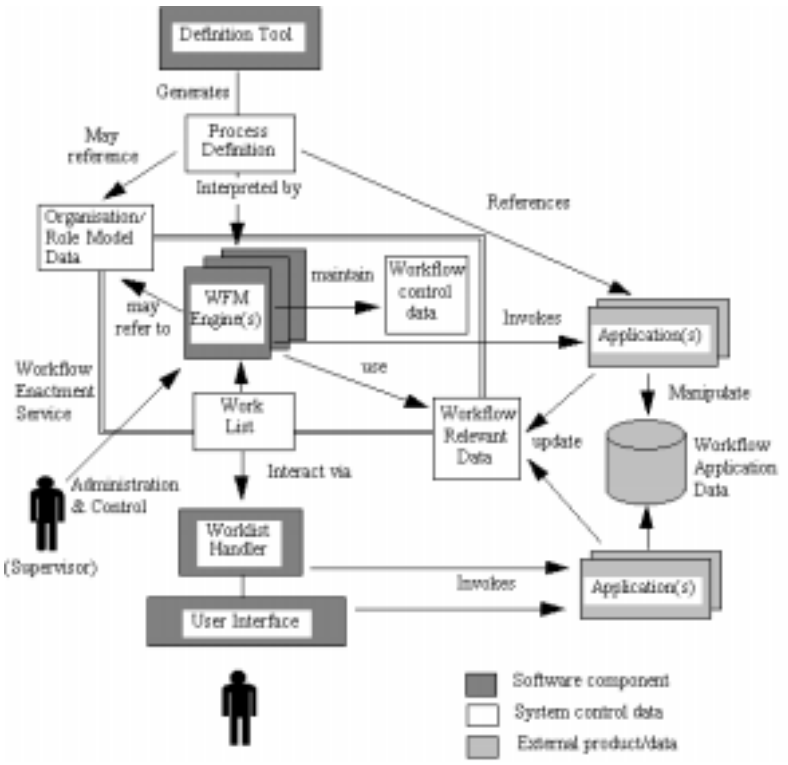


Figura 1.4: La struttura di un generico prodotto di Workflow

modello, frutto dell'analisi su prodotti esistenti, è più vicino all'approccio *engine-based* che non all'altro, ma in realtà cerca di cogliere i benefici di entrambi.
La definizione del processo (*Process Definition*), può riferirsi ad un Modello Organizzativo (o Modello dei Ruoli), che contiene informazioni riguardo l'organizzazione e la sua struttura. Ciò permette di specificare la definizione di processo in termini di entità e funzioni organizzative, piuttosto che in termini di utenti specifici. Il servizio di esecuzione del workflow, *Workflow Enactment Service*, poi, ha la responsabilità di collegare gli utenti con i ruoli organizzativi all'interno dell'ambiente di esecuzione del processo di workflow.
I principali componenti funzionali di un generico sistema di workflow sono illustrati in Fig. 1.4. Come si può vedere dalla grafica si identificano tre tipi di componenti:

1. *Componenti Software* che forniscono il supporto per varie funzioni all'interno del sistema (colorati di scuro);

2. vari tipi di *Definizioni di Sistema* e *Dati di Controllo*, (non colorati), che vengono utilizzati da uno o più componenti software;

3. *Applicazioni* e *Database*, (colorati in grigio chiaro), che non fanno parte del prodotto di workflow, ma sono invocati da esso e rientrano nel più generale sistema di workflow.

Qui di seguito vengono illustrati più in dettaglio i componenti funzionali elencati sopra.

Process Definition Tool. Il PDT è usato per creare una descrizione del processo in forma comprensibile e utilizzabile dal computer. Questa può basarsi su un linguaggio formale di definizione di processi, su un modello di relazioni fra oggetti, o, nei sistemi più semplici, su *scripts* o insiemi di comandi di instradamento per trasferire informazioni fra diversi partecipanti. Questo tool può essere fornito come parte del prodotto di workflow o anche all'interno di un prodotto di analisi di processi aziendali. In questo secondo caso è necessario avere però un formato compatibile per poter trasferire la definizione del processo da/verso il sistema di workflow.

Process Definition. La definizione del processo contiene tutte le informazioni riguardanti il processo necessarie a renderlo eseguibile dal software di esecuzione di workflow. In essa sono incluse informazioni sulle condizioni di inizio e fine del processo, sulle attività che lo costituiscono e sulle regole per navigare fra esse, sui compiti da assegnare agli utenti, riferimenti ad applicazioni da invocare e ad ogni tipo di dato rilevante per il sistema di workflow, e così via. La definizione del processo, unitamente ai dati rilevanti del workflow, è utilizzata per controllare la sequenza delle attività, fornendo informazioni sui criteri di ingresso/uscita da esse, su eventuali esecuzioni parallele, sui compiti svolti da utenti (può essere necessario accedere ai dati del Modello Organizzativo), o da applicazioni IT.

Workflow Enactment Service. Il software di esecuzione del workflow (WfES), ha il compito di interpretare la definizione del processo e di controllare l'esecuzione dei processi e la sequenza delle attività, aggiungendo alle liste dei lavori degli utenti, (*Worklist*), i nuovi compiti (*Work Item*), da svolgere e invocando altre applicazioni software quando necessario. Tutto ciò viene fatto attraverso uno o più "motori di workflow", (nella figura i *Workflow Management Engines*), che cooperano fra loro gestendo l'esecuzione delle istanze dei vari processi attivi. Il WfES mantiene al suo interno dati di controllo, sia centralizzati che distribuiti fra i vari motori. Questi dati interni di controllo includono le informazioni sullo stato di esecuzione dei processi e delle attività; in più spesso contengono anche informazioni di

sincronizzazione e di backup rispettivamente per coordinare le attività e recuperare lo stato di esecuzione in caso di problemi. Il WfES utilizza inoltre i *Relevant Data* del workflow, mentre sui dati delle applicazioni (*Application Data*), opera solo indirettamente.
Più in particolare, il WfES può essere considerato una macchina a stati, in cui istanze di processo o attività cambiano stato in risposta ad eventi esterni (come il completamento di un'attività), o a specifiche decisioni di controllo prese da un motore di workflow (come il passaggio allo step successivo all'interno di un processo). È possibile dare una illustrazione dello schema base del diagramma stati-transizioni per un'istanza di processo come mostrato in Fig. 1.5.

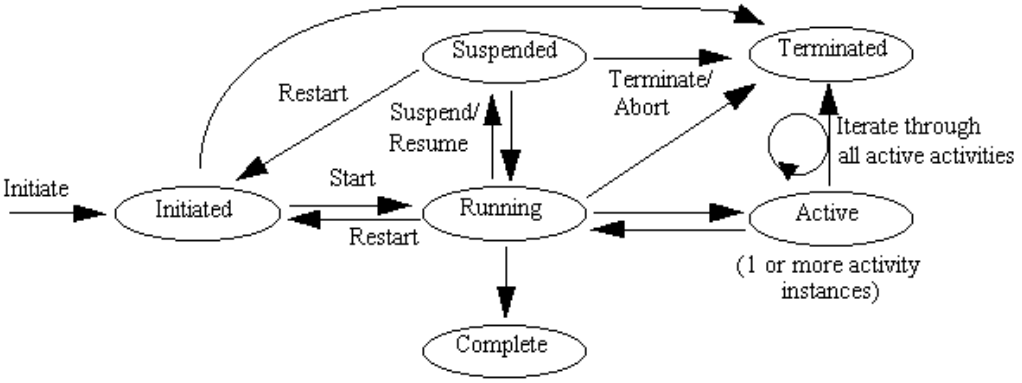


Figura 1.5: Diagramma stati-transizioni per l'istanza di un processo.

Nel diagramma sopra le transizioni fra gli stati (rappresentate da frecce), avvengono in risposta a particolari eventi, o in seguito al raggiungimento di condizioni elencate nella definizione del processo. Gli stati rappresentati (che costituiscono un insieme minimo per le diverse implementazioni), sono:

- *initiated*-l'istanza del processo è stata creata ma non è ancora in esecuzione;
- *running*-l'istanza ha iniziato l'esecuzione e una o più attività possono essere fatte partire;
- *active*-una o più attività sono iniziate ed esistono già le loro istanze;
- *suspended*-l'istanza del processo è in attesa; nessuna ulteriore attività viene iniziata fino ad un "risveglio" del'istanza;
- *complete*-l'istanza del processo ha raggiunto le condizioni di terminazione e sono in corso attività posteriori come il log dei dati;

- *terminated*-l'esecuzione del processo è stata fermata in modo anomalo per errori o per una richiesta dell'utente;
- *archived*-l'istanza del processo è stata posta in uno stato indefinito di archiviazione, ma può essere ripresa se si presenta la necessità (non è rappresentato nel diagramma).

Un analogo diagramma può essere usato per rappresentare gli stati di un'istanza di attività, con lo stesso significato di quello sopra presentato (per completezza questo diagramma sarà riportato nell'appendice A)

Workflow Relevant Data and Application Data. I *dati rilevanti* (detti anche *Case Data*) sono un insieme di informazioni generate o modificate dai programmi di applicazione, su cui si basano le scelte del percorso da seguire nell'esecuzione del processo o il controllo della stessa effettuati dal motore di workflow. Quando dunque si pone un problema di scelta fra attività alternative o semplicemente si deve monitorare lo stato di un processo, il motore di workflow accede a questi dati (che sono l'unico tipo di dati delle applicazioni a cui può accedere). I *dati delle applicazioni* vengono modificati direttamente e in modo esclusivo dalle applicazioni chiamate, sebbene il motore di workflow possa essre responsabile del trasferimento di questi dati fra applicazioni diverse quando più d'una è invocata per portare a termine il processo.

Worklist. Quando durante l'esecuzione del processo è necessario interagire con utenti umani, il motore di workflow si occupa di riempire un'elenco dei compiti da affidare ad essi (la *Worklist*, appunto), che è gestito, così come le interazioni con gli utenti, da un'apposito componente detto *Worklist Handler*. Questo processo può essere invisibile ai partecipanti al workflow, ai quali viene presentato solamente il prossimo compito da svolgere; oppure può essere più palese, con gli utenti che hanno davanti l'intera worklist da cui scegliere le attività da svolgere (secondo criteri prefissati) e che mostrerà il completamento di queste.

Worklist Handler & User Interface. Il gestore della worklist è un componente software che ha il compito di gestire le interazioni fra gli utenti e il WfES. È responsabile per tutti quei lavori che richiedono l'attenzione dell'utente e interagisce tramite la worklist con il WfES. Può essere un sistema molto semplice, che tiene traccia delle attività che attendono l'intervento dell'utente per essere svolte, come un sistema più sofisticato capace di controllare i carichi di lavoro ed eventualmente di riassegnare i compiti. Oltre a

ciò, in genere un gestore di worklist fornisce una vasta gamma di interazioni con applicazioni diverse, come l'aggiunta o l' eliminazione di partecipanti, la richiesta di far iniziare un'istanza di un determinato processo, l'accodamento di certe attività alle worklist di partecipanti scelti secondo qualche criterio, e così via.
Nella figura 1.4 l' Interfaccia Utente (*User Interface*), viene presentata come un componente software separato, responsabile del contatto con l'utente. In certi sistemi questa può essere integrata nel gestore di lista a formare un'unica entità.
L'invocazione di applicazioni diverse può rendersi necessario per supportare l'utente nello svolgimento di alcuni dei suoi compiti. Questo servizio può essere fornito dal gestore di lista automaticamente alla presentazione delle attività da svolgere, o può essere lasciato alla responsabilità dell'utente che utilizza per la chiamata l'interfaccia a lui dedicata.

Supervisory Operations. Le operazioni di supervisione vengono normalmente fornite in un sistema di workflow in base a privilegi di particolari utenti e/o workstation. Queste funzioni possono permettere ai supervisori di alterare l'allocazione delle attività fra gli utenti, di controllare l'evoluzione del processo, di valutare il funzionamento del sistema tramite statistiche, messaggi di errore, e così via.

Scenari alternativi. Mentre la maggior parte dei prodotti di workflow posseggono i componenti appena elencati, non è altrettanto vero che questo sia l'unico modello implementativo possibile (e utilizzato nella realtà). Citiamo per completezza che si può presentare la scelta fra WfES centralizzati o distribuiti, e quella fra diversi meccanismi di implementazione dei Worklist Handlers.

1.3.3 Modelli di workflow

In generale i prodotti disponibili sul mercato differiscono non solo nell'architettura, ma anche nell'approccio al problema (la "filosofia" del prodotto). Gli aspetti più importanti nella filosofia di un prodotto possono essere evidenziati ponendosi alcune domande:

- che cosa fluisce - informazioni o controllo?
- chi lo fa fluire - la persona che lo accetta (secondo un protocollo di offerta), o la persona che ha l'incarico di farlo (secondo un protocollo di distribuzione)?

- come lo fa fluire - attraverso definizioni statiche o attraverso interazioni dinamiche?

Cosa fluisce? Il termine workflow implica che qualcosa, e la parola "lavoro" non spiega molto, fluisce fra i task che compongono un processo. Tuttavia, i sistemi di workflow hanno concetti diversi riguardo il cosa fluisca. Queste idee condizionano sia il progettista, sia il modo in cui il sistema effettivamente lavorerà. Ci sono due modelli di base riguardo ciò che fluisce in un sistema di workflow:

- flusso *concreto*;
- flusso di *controllo*.

Flusso concreto I sistemi che seguono questo modello vedono il workflow come il trasferimento di form, dati o documenti fra i task. La maggior parte dei task condividono le stesse risorse di contenuto, che cambiano stato avanzando fra i task nel processo. Le informazioni sono condivise fra task successivi attraverso una catena in cui l'output di un task diviene l'input del seguente. In questi sistemi risulta quindi quasi impossibile progettare processi in cui diversi task utilizzano diverse risorse di contenuto.

Flusso di controllo È un modello più astratto del precedente, che lascia più libertà ai progettisti di associare diversi task a diverse risorse di contenuto. La logica del flusso di controllo fra i task è concepita indipendentemente dal flusso di dati fra i task, cosicchè questi possono condividere risorse informative, ma non sono costretti a farlo.
Il flusso di controllo è un modo orientato al sistema di descrivere il modello; in una visione orientata all'utente le transizioni fra i task potrebbero essere descritte come flussi di *responsabilità*; in una orientata agli oggetti si potrebbe usare la frase "passaggio di messaggi"; in una orientata alle regole, infine, si potrebbe vedere come mediazione fra le condizioni di uscita di un task e quelle di ingresso di un altro.

Confronto fra flusso di informazioni e flusso di controllo Se il trasferimento e la trasformazione di "cose" fra i task non è concepito in modo separato dalla nozione di flusso di controllo, questo limita le possibilità di modellazione. I prodotti con un modello di flusso concreto limitano la generalità e la flessibilità del mapping fra attività e risorse: attività diverse non possono usare facilmente risorse diverse (all'interno dello stesso processo). Questo tipo di modello funziona al meglio per i processi in cui il flusso di

controllo si mappa nel flusso di informazioni. Questa caratteristica è spesso verificata, tuttavia, questo fatto, per quanto utile, risulta limitante: è un errore assumere a priori che determinate attività, solo perchè fanno parte dello stesso processo, debbano condividere le stesse risorse:

- esistono spesso modi diversi di fare le stessa cosa: quale sia il metodo migliore può essere determinato proprio dalla disponibilità delle risorse;
- può essere desiderabile sviluppare processi con attività parallele che sfruttino risorse separate.

I prodotti che seguono un modello di flusso concreto sono spesso poco adatti allo sviluppo di processi che prevedono il parallelismo, proprio per la difficoltà di separare le risorse fra task contemporanei.
I modelli di flusso di controllo, invece, offrono una maggiore possibilità di modellazione. Con la libertà, però, nascono maggiori responsabilità: maggiore è la richiesta di impegno ai progettisti e di supporto al prodotto per quanto rigurda sempre la fase di progettazione. Il progettista dovrebbe essere in grado di lavorare ad un visione del processo che renda esplicito il legame fra le risorse di contenuto utilizzate in diversi task.

Verso chi fluisce? I sistemi reali di workflow tendono a favorire uno dei due meccanismi di collegamento fra task e persone: modelli di *system-offer* o di *system-deliver*.

- Con i modelli di system-deliver, il sistema di workflow si occupa di distribuire (to deliver), i task agli utenti. Il sistema può o meno avere consapevolezza se la persona che riceve il task sia nella posizione di poterlo eseguire.
- I modelli di system-offer, invece, offrono i task agli utenti, i quali poi sono liberi di decidere se accettarli o meno. Questo funziona poichè i modelli di system-offer utilizzano spesso i ruoli per fornire il collegamento just-in-time fra utenti e task: i task sono definiti come relativi a ruoli piutosto che ai singoli individui. Quando un task deve essere eseguito, viene offerto a tutti coloro che soddisfano i requisiti richiesti per il ruolo cui questo è legato.

Vediamo più in dettaglio i due modelli sopra accennati.

Modelli di system-offer Un sistema viene classificato all'interno di questa categoria se gli utenti hanno la responsabilità dei task in accordo ad un meccanismo di accettazione dell'offerta degli stessi.

Sistema di workflow: Ecco un task (o più task).
Utente: Lo eseguirò.

Non si dice nulla sul fatto che l'utente debba esplicitamente rifiutare un offerta o se basta che semplicemente la ignori per non accettarla; ciò che è importante è che l'utente deve esplicitamente accettarla per diventare responsabile dell'esecuzione del task offertogli.
I modelli di system-offer sono frequentemente associati ad ambienti di esecuzione di processi caratterizzati da:

- molti gruppi di lavoratori funzionalmente equivalenti;
- alto volume di lavoro;
- indirezione tramite ruoli;
- responsabilità dell'utente immediata.

In questi casi, questo modello è utilizzato in combinazione con i riferimenti basati sui ruoli per permettere il collegamento just-in-time fra gli utenti e i task, permettendo agli utenti stessi di accettare subito quei task che necessitano di un'esecuzione immediata.

Modelli di system-deliver Con questi modelli, i task vengono assegnati direttamente agli utenti dal sistema di workflow. Questo può accadere o perchè il sistema assegna i task a rotazione (o secondo qualche criterio predefinito), o perchè gli utenti sono stati esplicitamente nominati nella definizione del processo. L'essenza del protocollo fra il sistema e l'utente è come segue:

Sistema di workflow: Ecco un task per te.
Utente: [molteplici comportamenti possibili].

I modelli di system-deliver frequentemente lavorano tramite e-mail, sebbene questa non sia una caratteristica determinante. In questa classificazione non si pongono costrizioni sulla forma o sulla funzione della risposta dell'utente alla notifica del task, ma si assume che la notifica sia l'equivalente del trasferimento di responsabilità. Tuttavia questo trasferimento non deve

necessariamente essere la "fine della storia": i modelli di system-deliver possono fornire meccanismi per il rifiuto, la delega o un altro trasferimento della responsabilità di cui gli utenti sono stati notificati.

Confronto fra system-deliver e system-offer I modelli di system-offer sono più potenti di quelli di system-deliver. Usati in accoppiamento coi ruoli, essi semplificano l'amministrazione dell'ambiente di esecuzione centralizzando l'informazione per mezzo della quale gli utenti sono legati ai ruoli. Gli ambienti di system-offer forniscono anche un semplice ed automatico meccanismo di bilanciamento del carico di lavoro: gli utenti accettano i task in ragione della loro disponibilità e capacità di eseguirli. In realtà, però, per funzionare adeguatamente, i modelli di system-offer necessitano di un certo grado di supporto di gestione, o da parte del sistema di workflow o da parte di una persona preposta: cosa accadrebbe infatti se nessuno degli utenti avvisati accettasse il task? Un buon sistema di workflow dovrebbe guidare gli utenti in questa scelta tramite l'accostamento di criteri (priorità crescente con il trascorrere del tempo, etc.), alle liste dei task da svolgere presentate loro. Un'altra caratteristica che determina il buon funzionamento di un modello di system-offer è l'avere un efficace meccanismo di comunicazione bidirezionale fra le liste dei task offerti agli utenti, e il server centrale che gestisce i task in offerta. Questo può essere difficile da creare dal nulla con un sistema di e-mail, perchè il problema è quello di rimuovere il task dalle liste degli altri utenti dopo che uno di essi l'ha accettato.

Come fluisce? Molti sistemi di workflow presenti oggi sul mercato utilizzano un modello del flusso di processo detto *static-chained*. In esso i processi sono concatenati (chained), cioè tutti i task si originano a seguito di un singolo evento iniziale. L'avanzamento del processo è determinato dalle diramazioni della mappa che rappresenta il processo, in accordo con la logica che governa le transizioni fra i vari task. Gli eventi originati da applicazioni esterne o da altri processi di workflow, sono potenzialmente in grado di influenzare il corso dell'istanza, ma non tutti i prodotti permettono a livello di progettazione di inserire questi eventi esterni nella logica del processo.
La maggior parte delle definizioni di processo, oltre ad essere "concatenate", sono anche *statiche*: i task, le risorse usate e le regole di transizione devono essere totalmente e una volta per tutte specificate durante la fase di definizione (progettazione), del processo. Se anche questa metodologia è la più utilizzata, esiste un'alternativa, è cioè possibile avere definizioni *dinamiche*. In quest'ottica, le definizioni di un task o di un sotto-processo hanno effetto non a tempo di progettazione, ma a run-time, quando cioè il task o il sotto

processo viene invocato. Questa capacità di legame dinamico risulta molto utile se accoppiata a meccanismi che permettono di modificare "al volo" e in ogni momento, le definizioni. Nel caso di controllo dinamico del processo, non si può parlare di un "percorso di processo" come comunemente inteso: tutte le transizioni fra i task, infatti, sono affidate a e determinate dalla combinazione delle condizioni di uscita di un task e quelle di ingresso di altri possibili task, oppure dalle decisioni di un gestore di eventi preposto. In realtà sono molto rari i prodotti che offrono questo grado di libertà agli utenti.

Confronto fra controllo "chained" e "delegato" I modelli di controllo statico sul processo sono rigidi e rendono impossibile il supporto a processi per cui è noto a tempo di modellazione l'obiettivo di alcuni task ma non il metodo per raggiungerlo. Possono inoltre dare problemi in caso di condizioni globali che possono interessare un'istanza di processo, di necessità di gestione di eventi esterni, di necessità di roll-back o gestione di errori.
Il controllo delegato permette al progettista di disegnare sistemi potenti e flessibili. In particolare, rende più semplice la progettazione di un sistema che reagisca in modo appropriato agli eventi che si presentano dopo che il processo è iniziato. Queste funzionalità vengono però pagate in termini di difficoltà di implementazione, e richiedono un lavoro notevole da parte dei progettisti.

1.4 Limiti dei prodotti attuali

Molti dei prodotti di workflow presenti sul mercato sono nuovi e quindi ancora ad uno stadio poco più che prototipale che non garantisce, su carichi di lavoro consistenti, disponibilità costante ed affidabilità elevata. Essi presentano inoltre alcuni importanti limiti di tipo concettuale:

- nozione restrittiva del concetto di *task*;
- funzionalità relative ai ruoli non molto avanzate;
- capacità ancora povere in ambito di gestione dei processi;
- limitata capacità nella gestione dei dati.

1.4.1 Limiti del modello di workflow basato sui task

La maggior parte dei sistemi di workflow considerano il *task* come qualcosa che è eseguito da un'unica persona alla volta utilizzando le risorse di un'applicazione. Questo esclude il lavoro basato sui gruppi ed i task che sono definiti

dall'obiettivo piuttosto che dal metodo. Questa nozione di task, inoltre, non fornisce supporto nè per il "meta-lavoro" che le persone possono dover fare per gestire i loro reali incarichi, nè per task che hanno una struttura interna (in particolare quando la struttura è gerarchica o deve essere dinamica).
Ci sono varie alternative alla nozione "una persona per un task in un momento preciso":

- più di una persona lavora su un task in un momento (task multi-utente);
- una persona lavora su un task in diversi intervalli di tempo (task distribuiti nel tempo);
- una persona lavora su un task utilizzando una molteplicità di risorse applicative (task multi-risorsa).

Vediamo quanto supporto gli attuali sistemi forniscono per queste soluzioni alternative.

Una persona, un task Nessun prodotto di workflow attualmente disponobile supporta i task multi-utente in cui il lavoro è il frutto della collaborazione fra due o più persone. Le cause di questo limite sono di tipo ideologico e tecnico:

- il tipo di lavoro per cui si impiegano i sistemi di workflow è organizzato secondo una visione "tayloristica" (a catena di montaggio), piuttosto che secondo una visione basata sui gruppi di lavoro, che sarebbe forse più indicata per i processi aziendali diversi dalla produzione;
- utilizzando le risorse informatiche, risulta difficile farle condividere in tempo reale agli utenti. Tuttavia, l'avvento della video conferenza e l'utilizzo di ambienti condivisi, può facilitare notevolmente questo compito.

Un task, un intervallo di tempo Molti prodotti di workflow permettono agli utenti di iniziare e fermare il lavoro su di un task quando questo è avviato. Tuttavia, solo pochi supportano attivamente questo stile di lavoro. È spesso una buona idea permettere alle persone di sospendere il lavoro su di un task:

- le persone necessitano comunque di pause, e sarebbe utile poter distinguere il tempo totale speso per il completamento di un task, da quello in cui il task è stato effettivamente attivo;

- un lavoro ad alta priorità potrebbe interromperne uno con priorità inferiore;
- alcuni task potrebbero richiedere input esterni non immediatamente disponobili, per essere completati.

Se è possibile per un lavoratore operare su più fronti contemporaneamente, risulta necessario il supporto al meta-lavoro che accompagna questo spazio di scelta: gestire cosa fare e quando farlo. Tuttavia, solo pochi prodotti forniscono funzionalità in questa direzione, comprendendo che organizzare il lavoro già avviato è altrettanto importante che selezionarne di nuovo da far partire.

Un task, un'applicazione I task multi-risorsa sono un altro limite dei sistemi attuali. Ciò deriva dal fatto che la progettazione implica la creazione di un riferimento ad una (spesso singola), applicazione collegata ad uno specifico task. Ci sono prodotti che lasciano un certo grado di libertà, permettendo un "late-binding" a run-time, ma in generale già a livello di modellazione bisogna fissare i riferimenti. Inoltre, anche i sistemi più "liberali", non offrono poi un supporto adeguato per permettere all'utente di cambiare l'applicazione associata al task una volta che questo gli è stato notificato. In generale, l'idea comune a tutti i sistemi è che i task sono definiti dal metodo piuttosto che dallo scopo; per contrasto le risorse umane sono considerate di solito come suscettibili di sostituzione.

Modellazione gerarchica La maggior parte dei sistemi di workflow sono pensati per progetti "piatti", privi quasi totalmente di una struttura gerarchica, ossia della possibilità di avere sotto-processi come componenti di un processo di livello superiore. Un tipo di modellazione così fatta avrebbe benefici per i processi complessi, ma anche per processi con attività parallele. In questo caso, infatti, un processo "piatto" lascia l'utente che attende la convergenza di due task paralleli per iniziare il proprio, in uno stato di impotenza (non può ad esempio far affrettare uno dei due utenti se l'altro ha già terminato, e così via). Un processo strutturato gerarchicamente, invece, permetterebbe al terzo utente di vedere l'andamento dei due task paralleli (visti come sotto-task del suo), e, data la sua responsabilità superiore, di intervenire attivamente per favorirne l'esecuzione.

Raffinamento dinamico dei task Da un punto di vista più ampio, un task può essere visto come la produzione di uno stato finale definito che soddisfa uno scopo, senza preoccuparsi dei mezzi. Ciò che si intende con

raffinamento dei task è l'abilità dei lavoratori di elaborare la definizione dei task di cui hanno responsabilità all'interno di un'istanza di processo, suddividendoli in sotto-task da assegnare a differenti utenti. Questa idea molto potente è però scarsamente supportata dai sistemi attuali, principalmente per le carenze relative all'utilizzo di processi strutturati gerarchicamente.

Task distribuiti La maggior parte dei sistemi attuali sono ancora basati sul paradigma *client-server* per la gestione dei processi di workflow. Solo pochi prodotti hanno seguito (o lo stanno facendo), la tendenza evolutiva dei sistemi verso il paradigma *distribuito*, per fornire più ampia libertà agli utenti di workflow in una realtà aziendale che va sempre più verso la decentralizzazione dei processi. Lo scenario che si prospetta è quello di un insieme di motori di workflow ognuno ubicato in siti diversi, capaci di scambiarsi dati informazioni e task, senza dipendere troppo dalla struttura di interconessione fra i siti (che può non essere fissa). Di questo argomento parleremo ancora in seguito, nell'illustrare il progetto affrontato in questa tesi.

1.4.2 Limiti dei ruoli

Utilizzare i ruoli per collegare le persone con i task permette al progettista di specificare un ruolo per le attività da svolgere e le persone che possono impersonare quel ruolo. La persona reale che eseguirà il lavoro viene definita run-time, di solito non appena il task è pronto per essere svolto. Questo meccanismo è molto potente, ma ha delle mancanze. Pochi sistemi, infatti, permetto al progettista di andare oltre avvantaggiandosi dei benefici pratici del riuso di precedenti legami ruolo/lavoratore/task. Il progettista quasi mai può specificare un ordine di preferenza all'interno di un insieme di "mappaggi". Questo appare chiaro pensando ad processo in cui un task sia inserito in un ciclo da eseguire finchè determinate condizioni non vengano soddisfatte. È evidente che sarebbe desiderabile che fosse la stessa persona che ha esguito il task la prima volta a rieseguirlo, date le informazioni che ha già acquisito nella prima esecuzione. Se però il task è assegnato ad un ruolo, non c'è modo di impedire che altre persone con le stesse caratteristiche possano assumerne la responsabilità nei cicli seguenti. La rilevanza di queste osservazioni, comunque, si estende a tutti i processi in cui ricorra più volte lo stesso ruolo. Ancora si possono avere cicli, quali quelli di controllo qualità, in cui è desiderabile avere persone diverse nelle varie iterazioni, per avere giudizi più attendibili.
In generale, dunque, quando un progettista non può controllare intelligentemente le relazioni ruolo/persona/task, ne risultano limitate possibiltà di sfruttamento delle conoscenze dei diversi utenti e delle loro relazioni.

1.4.3 Limiti nella gestione dei processi

I prodotti di workflow, per quanto possa sembrare strano, sono maggiormente carenti nell'ambito della gestione di processo (*process management*). Due problemi comuni alla maggioranza dei sistemi sono: la gestione delle *condizioni di sincronizzazione* (quando cioè più flussi di lavoro convergono in uno solo), e l'*ottimizzazione dinamica*. Il problema base è che lo stato dell'ambiente di esecuzione del processo non è di solito un elemento di progetto che possa essere usato per "lanciare" azioni a scelta dei progettisti.

Condizioni di sincronizzazione Spesso l'unico modo per gestire le condizioni di sincronizzazione in processi che presentano flussi di lavoro paralleli è l'utilizzo di "allarmi" basati sul controllo di scadenze temporali. In questo modo il supervisore può accorgersi che qualcosa è in ritardo, ma sono pochi i prodotti che permettono di eseguire delle notifiche basandosi sullo stato dei singoli task che devono sincronizzarsi in un flusso parallelo. È inoltre quasi impossibile modificare adeguatamente la priorità di un task in ritardo al punto di sincronizzazione che frena la continuazione del processo.
Un altro grosso problema è la possibilità di esprimere solamente condizioni di *and* non flessibili per la sincronizzazione di flussi paralleli, condizioni cioè rigide che impediscono al processo di proseguire finchè tutti i flussi non sono arrivati al punto. La soluzione, per ora non supportata che da pochissimi prodotti, è permettere condizioni più flessibili, con cui si accetta che anche solo una parte dei flussi sia terminata per poter portare avanti il processo. Questo comporta però, un'adeguata gestione dei flussi rimanenti che vanno monitorati e cancellati; funzionalità peraltro ancora remote per i prodotti attuali.

Ottimizzazione dinamica La gestione di processo in un sistema di workflow è spesso limitata al meccanismo di avvertimento (*warning*). Risulta difficile per un progettista creare un sistema che corregga automaticamente un ambiente di processo mal funzionante. Quando un sistema va in crisi, ci vuole ben altro che una serie di allarmi per correggerlo, sono necessari passi più seri:

- modificare l'accesso dei task alle risorse, per abilitare più persone a contribuire a risolvere i "colli di bottiglia";
- modificare l'allocazione dei task, per trasferire task pendenti da un utente (o gruppo di utenti), ad un altro;
- modificare la stessa definizione di processo.

Le risorse disponibili possono determinare non solo quali risorse sono più adatte da usare, ma anche quale metodo è più consono per portare a termine un'istanza di processo. Può essere utile assegnare i task "pericolosi" per il processo, a persone esperte, o ancora modificare le priorità di alcuni task durante l'esecuzione di un'istanza. La maggior parte dei sistemi di alto livello propongono una visione del database che contiene tutte le informazioni sulle istanze attive, ma pochi di essi forniscono una comoda interfaccia grafica per intervenire dinamicamente su di esse.

1.4.4 Limiti nella gestione dei dati

Una carenza generalizzata nei prodotti di workflow attualmente presenti sul mercato è riscontrabile nella gestione dei dati, come evidenziato ad esempio in [2]. Molti (se non tutti), i sistemi di workflow hanno la possibilità di dichiarare, al momento della definizione di processo, alcuni dati che saranno utili nella gestione del processo stesso (i *Relevant Data* del workflow). Normalmente, poi, è possibile definire anche dati che verranno utilizzati dalle applicazioni esterne nello svolgimento dei vari task (gli *Application Data*). Un primo problema si presenta già a questo livello, in quanto spesso i due tipi di dati non vengono distinti dal sistema, che li memorizza allo stesso modo. Quando poi il sistema si trova ad operare su processi *data-intensive*, cioè che hanno bisogno di utilizzare una gran mole di dati (ad esempio nella gestione di documenti), queste carenze vengono acuite. In queste situazioni il generico sistema di workflow risulta quindi inadatto, non disponendo delle funzionalità tipiche ad esempio di un DBMS per garantire la correttezza e l'affidabiltà dei dati. Approfondiremo in seguito questo argomento, proponendo anche una soluzione possibile per superare questi limiti.

1.5 La decisione sul workflow

La decisione principale da compiere non è sul prodotto da scegliere, ma sull'utilizzare o meno un qualsiasi sistema di workflow. Ci sono quattro messaggi chiave per i futuri/possibili utilizzatori di sistemi di workflow:

- comprendere i limiti dei prodotti attualmente disponibili;
- considerare se i processi da automatizzare sono veramente adatti ad una soluzione mediante workflow;
- operare una accurata scelta del prodotto;
- pianificare l'implementazione al fine di minimizzare i rischi.

1.5.1 Per cosa è indicato il workflow?

A dispetto dei limiti dei sistemi attuali, si possono elencare cinque condizioni sotto cui può aver senso pensare ad una implementazione di un workflow:

- i processi sono divisi esplicitamente in task;
- esistono regole che determinano la logica delle transizioni fra i task;
- i task utilizzano risorse informative in formato elettronico;
- i task necessitano di essere comunicati ai lavoratori;
- c'è la necessità di avere un controllo di processo.

Quando un processo soddisfa tutti questi criteri, una sua implementazione tramite un sistema di workflow può risultare soddisfacente e vantaggiosa.

1.5.2 Selezione dei prodotti

Una volta che si è deciso di adottare per i propri processi una soluzione mediante un sistema di workflow, bisogna scegliere fra tre modi in cui questo può essere acquisito:

- costruirne uno;
- comprarne uno che è inserito in un'applicazione gestionale;
- costruirne uno utilizzando un prodotto di workflow general-purpose.

La prima via è da praticare solo nel caso in cui nessun prodotto esistente possa soddisfare i propri bisogni. La seconda ha senso nella misura in cui l'applicazione supporta i processi da automatizzare; ma una sua eventuale estendibilità e personalizzabilità son caratteristiche da non trascurare. Infine, l'ultima opzione consente di disegnare i propri processi, ma impone di adeguarsi agli standard del sistema scelto e di imparare ad utilizzarne le diverse funzionalità. In ogni caso rimane fondamentale nella scelta, l'attenta valutazione delle proprie necessità: prodotti con funzionalità eccellenti in determinati ambiti possono non essere adatti al nostro caso.

1.5.3 Alcune parole sull'implementazione

Ci sono molte questioni tecniche relative all'implementazione che meritano attenzione nella pianificazione di un progetto di workflow:

- uso delle risorse e performance;
- limitazioni su workflow distribuiti;
- sforzo di integrazione col sistema esistente;
- "gap" culturale fra la progettazione di workflow e i professionisti di IS (Information System).

Il workflow comporta un cambio di prospettiva e di paradigma per i professionisti da sempre impegnati nella gestione della sicurezza e dell'organizzazione dei dati. Tuttavia la questione più importante è l'effetto del workflow sui lavoratori. Molte organizzazioni intraprendono un progetto di workflow nella speranza di poter avere riduzioni di personale (cosa nascosta dietro frasi come "riduzione dei costi" e "miglioramento dei processi"), senza rendersi conto che il workflow non è una cura per tutti i problemi. È quindi importante, per avere poi effettivi benefici, identificare l'origine dei problemi e delle inefficienze dei processi esistenti, prima di cercare di migliorarli attraverso il workflow.

1.6 Gli standard

1.6.1 La Coalition

La Workflow Management Coalition (W*f*MC), è stata fondata nell'Agosto del 1993 come associazione internazionale senza fini di lucro per lo sviluppo e la promozione di standard per i sistemi di workflow e si è in breve affermata come il riferimento principale nel mondo per questo tipo di software. Possono farne parte tutti coloro che sono interessati nella creazione, analisi o impiego di Sistemi per la Gestione di workflow (Workflow Management Systems). Attualmenete ad essa partecipano come membri più di 185 fra aziende produttrici, utilizzatori e analisti di sistemi di workflow.

La missione della Coalition prevede tre punti chiave:

1. incrementare il valore degli investimenti degli utilizzatori tramite la tecnologia del workflow;
2. ridurre al minimo il rischio nell'utilizzo di prodotti di workflow, fornendo garanzie sulle funzionalità fornite e sull'affidabilità;

3. espandere il mercato dei prodotti di workflow attraverso la diffusione di una maggiore conoscenza di questo tipo di tecnologia.

La cornice degli standard. La Coalition ha proposto un inquadramento per la definizione degli standard per il workflow. In esso sono incluse cinque categorie di interoperabilità e standard di comunicazione che permetteranno a molti prodotti diversi (forniti da un mercato in evoluzione continua), di cooperare all'interno dell'ambiente di lavoro di un utente.

La struttura della Coalition. La Coalition è divisa in due comitati principali, il *Comitato Tecnico* e il *Comitato Direttivo*. All'interno di ognuno di questi due comitati esistono poi gruppi più piccoli a cui spetta il compito di definire la terminologia, gli standard per l'interoperabilità e la connettività di sistemi diversi, e di comunicare l'evoluzione del lavoro della Coalition alla comunità di utenti di sistemi di workflow. I due comitati principali della Coalition si incontrano quattro volte l'anno per un periodo di tre giorni, in meeting che si tengono alternativamente in nord America e in Europa. Per i gruppi più piccoli oltre questi incontri istituzionali, possono essercene altri, quando necessario, durante il corso dell'anno. Per diventare membri della Coalition è necessario sottoscrivere un documento in cui sono spiegati i regolamenti, i diritti, etc. Membri famosi della Coalition nel campo delle aziende sono IBM, Microsoft, Hp ...

Il lavoro della Coalition. Gli obiettivi della Coalition sono:

- sviluppare una terminologia standard per descrivere i sistemi di workflow e il loro ambiente;
- permettere l'interoperabilità fra sistemi di workflow di produttori diversi;
- aiutare gli utenti nella comprensione dei sistemi di workflow attraverso il Reference Model;
- lavorare assieme a gruppi di industrie collegate per fissare gli standard e comunicare il suo lavoro.

I gruppi di lavoro della Coalition. La Coalition ha stabilito un numero di gruppi di lavoro per ogni area di interesse. Il lavoro è incentrato sullo sviluppo di un modello di riferimento che fornisca un ambiente comune per lo sviluppo di sistemi di workflow. Questo modello identifica le caratteristiche

comuni dei sistemi di workflow e tenta di porre le basi per lo sviluppo di interfacce standard fra essi. Qui di seguito si elencano i gruppi di lavoro attualmente costituiti e le loro aree di interesse.

- **Reference Model-** specificare un ambiente di lavoro per i sistemi di workflow, identificando le loro caratteristiche,funzioni ed interfacce.
- **Glossary-** sviluppare una terminologia standard per il workflow.
- **Process Definition Tool Interface(1)-** definire una interfaccia standard fra le applicazioni per la definizione dei processi e il workflow Engine.
- **Workflow Client Application Interface(2)-** definire standard per il mantenimento, da parte del workflow Engine, dei Work Item che il workflow Client presenta all'utente.
- **Invoked Application Interface(3)-** definire una interfaccia standard per permettere al workflow Engine di richiamare applicazioni diverse.
- **Workflow Interoperability Interface(4)-** definire una varietà di modelli di cooperazione e gli standard applicabili ad essi.
- **Administration & Monitoring Tool Interface(5)-** definire le funzioni di monitoraggio e controllo.

Workflow Reference Model Il Workflow Reference Model è stato sviluppato partendo dalla struttura generale di un prodotto di workflow (presentata nelle pagine precedenti), cercando di identificare all'interno di questa struttura, le *interfacce* per permettere a prodotti diversi di cooperare a vari livelli. Tutti i sistemi di workflow presenti sul mercato hanno un insieme minimo di componenti comuni che interagiscono in modi predefiniti; diversi prodotti, però, mostrano tipicamente diversi livelli di funzionalità per ognuno di questi componenti. È necessario quindi, per raggiungere l'interoperabilità fra prodotti diversi, avere un insieme di interfacce e di formati per lo scambio di dati standard. In questo modo possono essere costruiti vari scenari di interoperabilità, identificando diversi livelli di conformità funzionale, per rispondere alla varietà di prodotti sul mercato.
La figura 1.6 mostra i componenti e le interfacce principali all'interno dell'architettura del workflow. Queste interfacce sono denominate *WAPI*, cioè Workflow APIs e formati di interscambio, e possono essere considerate come costrutti attraverso cui è possibile accedere ai servizi del sistema di workflow

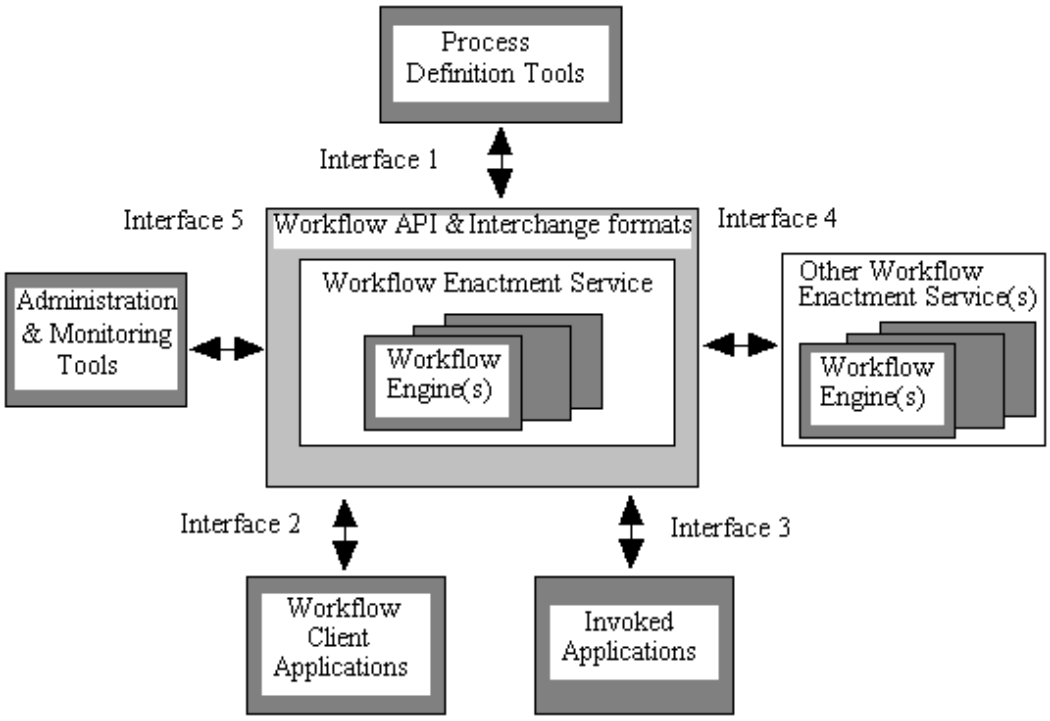


Figura 1.6: Il Modello di riferimento del Workflow: componenti e interfacce.

e che regolano le interazioni fra il software di controllo del workflow e gli altri componenti del sistema. Le interfacce sono cinque, ma poichè forniscono alcuni funzionalità comuni, è meglio considerarle come un servizio di interfaccia unificato che viene utilizzato per supportare le funzioni di gestione del workflow nelle cinque aree funzionali indicate.
Per quanto concerne le descrizioni dei componenti principali si rimanda a quanto detto sopra, qui di seguito illustriamo le caratteristiche salienti delle cinque interfacce, le *Workflow Application Programming Interface & Interchange*, (WAPI).

Workflow Definition Interchange: interfaccia 1. L'interfaccia fra i componenti di modellazione e definizione del processo e quelli di gestione dell'esecuzione (WfES), è detta *interfaccia import/export per la definizione del processo*. La sua natura è un formato di interscambio e delle chiamate API che possono supportare lo scambio di informazioni riguardo la definizione dei processi fra una varietà di media fisici ed elettronici. Lo scambio può essere totale o riguardare anche solo una parte della definizione del processo (ad esempio alcuni cambiamenti ad una precedente definizione, o la descrizione

di nuove attività). L'utilizzo di un formato standard per la definizione del processo comporta diversi benefici.

- Definisce una separazione chiara fra l'ambiente di sviluppo (Build-Time) e quello di esecuzione (Run-Time), permettendo all'utente di scegliere indipendentemente tool per la modellazione e prodotti per la gestione dell'esecuzione dei processi. Basandosi sullo standard, infatti, prodotti diversi possono comunicare correttamente fra loro.

- Offre un supporto per poter esportare la stessa definizione di processo verso parecchi prodotti di workflow che cooperano in un ambiente di esecuzione distribuito.

La Coalition si è concentrata, in quest'area, su due aspetti principali:

1. lo sviluppo di un meta-modello per esprimere gli oggetti, le loro relazioni e attributi all'interno di una definizione di processo, e che possa formare la base per un insieme di formati di interscambio per condividere queste informazioni fra prodotti diversi;

2. chiamate ad APIs fra sistemi di workflow o fra un sistema di workflow e un prodotto per la definizione di processo, per fornire un modo comune di accesso alle definizioni di processo.

Workflow Client Application Interface: interfaccia 2. Quando l'utente finale interagisce col sistema di workflow lo fa attraverso le worklist. Queste possono essere gestite direttamente dal sistema, ma più spesso vengono gestite da applicazioni personalizzate, che rispondono meglio aigli standard grafici e di interfaccia utente della ditta. Nasce in questo contesto la necessità di avere un meccanismo di comunicazione standard fra le applicazioni utilizzate dagli utenti finali per interagire coi sistemi di workflow e i sistemi stessi.
L'approccio della Coalition consiste nel contenere la varietà dei modelli entro un insieme di APIs (le WAPI), da utilizzare per accedere da una applicazione di workflow al Workflow Engine e alle worklist, indipendentemente dalla implementazione pratica del prodotto. Le specifiche delle APIs sono presentate in un apposito documento della Coalition, qui interessa citarne le caratteristiche salienti, raggruppate per aree funzionali.

- *Session Establishment:* conessione/disconnessione di sessioni fra sistemi che partecipano al processo.

- *Workflow Definition Operations:* funzioni di reperimento/ricerca di dati su nomi o attributi di definizioni di processo.

- *Process Control Functions:*

  - creazione/avvio/terminazione di una singola istanza di processo;
  - sospensione/riattivazione di una singola istanza di processo;
  - forzare un cambio di stato in un'istanza di processo o di attività;
  - assegnamento/query di un valore di un attributo (ad esempio la priorità), di un'istanza di processo o di attività.

- *Process Status Functions:*

  - apertura/chiusura di query su istanze di processo o attività;
  - reperimento di informazioni riguardo istanze di processo o attività, secondo specifici criteri di selezione .

- *Worklist Handling Functions:*

  - apertura/chiusura di query su una worklist;
  - reperimento di informazioni riguardo worklist secondo specifici criteri di selezione;
  - notifica della selezione/riassegnamento/completamento di uno specifico workitem;
  - assegnamento/query di un attributo di un workitem.

- *Process Supervisory Functions:* tutte le funzioni di gestione delle istanze di processo (cambio di stato, riassegnamento, terminazione delle istanze, etc.), che dovrebbero essere assegnate ad un utente particolare (ad esempio l'amministratore del sistema).

- *Data Handling Functions:* reperimento/modifica dei dati dell'applicazione o dei dati specifici del workflow.

Invoked Application Interface: interfaccia 3. È facile comprendere come una qualsiasi implementazione pratica di sistema di workflow non possa avere le capacità per invocare tutte le potenziali applicazioni che possono esistere in un ambiente eterogeneo. Per questo motivo la Coalition ha pensato di fornire un insieme di chiamate ad APIs adeguate, che elenchiamo di seguito.

- *Session Establishment:* connessione/disconnessione di sessioni d'uso della applicazione.
- *Activity Management Functions:*

  (dal motore di workflow verso l'applicazione)

  - avvia una attività;
  - sospendi/riavvia/termina una attività;

  (dalla applicazione verso il motore di workflow)

  - notifica del completamento di una attività;
  - segnalazione di un evento;
  - ricerca di attributi dell'attività.
- *Data Handling Functions:*

  - fornire i dati rilevanti del workflow (prima dell'attività verso l'applicazione, dopo dall'applicazione al motore di workflow);
  - fornire i dati dell'applicazione.

WAPI Interoperability Functions: interfaccia 4. Un obiettivo importante per la Coalition è la definizione di standard per permettere a sistemi di workflow di produttori diversi di interagire fra loro (ad esempio scambiandosi work-item). Per ora sono stati identificati quattro possibili modelli di interoperabilità che vengono brevemente illustrati qui sotto.

1. **Connected Discrete (Chained)**

   Questo modello identifica un punto di connessione all'interno di un processo A, per collegarsi ad un analogo punto di un processo B. In questo modo viene supportato il trasferimento di una singola porzione del workflow (una istanza di processo o attività), fra due sistemi diversi, che poi operano *idipendentemente* l'uno dall'altro sui due processi.

2. **Hierarchical (Nested Subprocesses)**

   Questo modello permette di "incapsulare" l'esecuzione di un processo da parte di un particolare motore di workflow, all'interno di un processo principale, gestito da un diverso sistema, di cui il primo rappresenta un singolo task. Questa connessione gerarchica può essere reiterata, fino ad avere parecchi livelli di processi innestati.

3. **Connected Indiscrete (Peer-to-Peer)**

   Questo modello propone un ambiente di esecuzione gestito da diversi sistemi di workflow fra loro collegati per fornire un sistema virtualmente unico. In questo scenario, infatti, il processo dovrebbe fluire di task in task, in modo trasparente all'utente, coinvolgendo i vari sistemi quando necessario. È evidente la complessità di un'implementazione simile, vista la necessità di interfacce ampie ed articolate.

4. **Parallel Synchronised**

   Questo modello permette a due processi di essere eseguiti in modo quasi indipendente, e possibilmente sotto il controllo di sistemi diversi, ma richiede l'esistenza di punti di sincronizzazione fra i due processi. La sincronizzazione richiede che una volta che i processi abbiano raggiunto un determinato punto della loro esecuzione, venga generato un evento comune.

Per poter realizzare questi scenari la Coalition propone delle WAPIs che insieme compongono la quarta interfaccia. Le funzioni fornite da queste APIs sono elencate brevemente nel seguito:

- invocazione di attività o sotto processi.
- controllo dello stato di istanze di processi/attività.
- trasferimento di dati del workflow o dell'applicazione.
- coordinamento di punti di sincronizzazione.
- lettura/scrittura di definizioni di processo.

Come si può vedere queste funzioni fanno riferimento a due momenti diversi: la configurazione del sistema e l'esecuzione dei processi. In particolare al primo aspetto si riferiscono tutte le funzioni necessarie a scambiare definizioni di processo (o parti di esse), fra i sistemi; al secondo si riferiscono invece tutte le altre, necessarie a gestire una esecuzione distribuita in vario modo fra più sistemi diversi.

Administration & Monitoring Interface: interfaccia 5. In quest'ultima area di standardizzazione, la Coalition propone una interfaccia comune per le funzioni di amministrazione e controllo del lavoro. Sebbene alcune funzioni di questo tipo siano già state definite nelle interfacce precedenti, questa nuova area risponde alla necessità emersa in ambito aziendale, di una funzione di controllo posta ad un livello più elevato, per vedere il workflow

nella sua totalità, e per avere specifiche capacità in ambito di sicurezza, autorizzazioni, controllo. Il modo più semplice di presentare quest'interfaccia è quello di interporla fra una singola applicazione di gestione ed i vari sistemi di workflow che compongono l'intero ambiente di esecuzione, evidenziando in questo modo la visione totale del lavoro che l'amministrazione del sistema deve avere. Le operazioni principali fornite dalle APIs che compongono l'interfaccia, seppura l'area sia ancora in via di sviluppo, vengono elencate nel seguito (si noti la presenza di funzioni già descritte nelle interfacce precedenti).

- **User Management operations:** assegnazione/cancellazione/sospensione di privilegi di utenti o gruppi di lavoro.
- **Role Management operations:** definizione/cancellazione/modifica di relazioni roulo-partecipante e di attributi dei ruoli.
- **Audit Management operations:** ricerca/stampa/avvio/cancellazione di log degli eventi, tracce di audit, etc.
- **Resource Control operations:** gestione dei livelli di concorrenza fra processi/attività e dei dati di controllo delle risorse.
- **Process Supervisory functions:** gestione delle definizioni di processo e delle istanze (ad esempio cambiare lo stato di una definizione di processo o delle sue istanze, abilitare/disabilitare particolari versioni di una definizione di processo, terminare tutte le istanze di un determinato processo, e così via).
- **Process Status functions:** gestione delle informazioni riguardanti lo stato di un'istanza di processo/attività (ad esempio query specifiche su istanze secondo detrminati criteri, ricerca di dati relativi all'esecuzione, e così via).

1.6.2 Standard alternativi

Una proposta alternativa a quella della Coalition viene da due aziende importanti nel panorama informatico mondiali, come Microsoft e Wang. Insieme, stanno lavorando allo sviluppo di un set di API chiamato *Work Management Application Programming Interface*, (WMAPI), che è simile, come obiettivi, alle WAPI che sono in via di definizione nella Coalition stessa. In realtà, comunque, i due tentativi di standardizzazione sono collegati fra loro per mezzo dei gruppi di lavoro della Coalition, di cui peraltro sia Microsoft che Wang sono membri fondatori.

1.7 Previsioni sul mercato

Le previsioni per il mercato del workflow parlano di continua crescita nei prossimi tre anni. È previsto, data la convergenza fra prodotti form-based ed engine-based, un calo nei prezzi. Ci sarà inoltre, un consolidamento del mercato, e una caduta nel numero di produttori.
È probabile inoltre un'evoluzione della relazione che il workflow ha nei confronti delle applicazioni gestionali e dei tool di sviluppo per sistemi client-server. In particolare si prevede che:

- il workflow all'interno delle applcazioni gestionali si svilupperà come capacità embedded, con limitate possibilità di estensione;
- nei tool di sviluppo di applicazioni, invece, crescerà attraverso le partnerships.

L'analisi dello sviluppo del mercato dei prodotti di workflow nei prossimi anni può essere affrontata identificando e discutendo quattro punti chiave:

- l'*offerta;*
- gli *standard;*
- la *domanda;*
- i *prezzi.*

Vediamoli ora uno per uno.

1.7.1 L'offerta

Ci sono tre fattori base che influenzeranno significativamente il mercato del workflow dal punto di vista dell'offerta:

- le funzionalità dei prodotti;
- la frammentazione;
- la relazione fra il workflow ed altri tipi di tecnologie.

Funzionalità dei prodotti Attualmente la maggior parte dei prodotti rientra in una delle due seguenti categorie:

- sistemi che offrono supporto per costruire applicazioni per i task, e sono semplici ma flessibili;
- sistemi che offrono potenti funzionalità di disegno e gestione di processo.

Il primo tipo di sistemi sono generalmente form-based e relativamente economici: circa 50$ per postazione di lavoro. Il secondo tipo comprende invece sistemi engine-based più costosi: almeno 200$ circa per postazione di lavoro (si noti che questi prezzi valgono in caso di grandi installazioni, per piccoli numeri i prezzi sono più elevati).

Evoluzione dei prodotti esistenti Si prevede una convergenza fra i due tipi di prodotti forniti, guidata soprattutto dai fornitori di prodotti form-based, che probabilmente cercheranno di dotarli di un "motore" che assicuri una più potente gestione dei processi. Contemporaneamente i prodotti engine-based dovranno o ridurre i prezzi, o potenziare ancora (e i margini, come visto nella sezione dedicata ai limiti, ci sono), le funzionalità che li caratterizzano. Inoltre si prevede una migliore capacità di costruzione di applicazioni per i task, attraverso una integrazione spinta con tool di sviluppo quali Visual Basic o Power Builder.
Un altro passo importante verso la nascita di prodotti di workflow general-purpose, sarà lo sviluppo di soluzioni pre-configurate e costruite su misura per particolari tipi di processi. Questo permetterà di ridurre i costi di implementazione e di ampliare così il mercato. In quest'ottica saranno di importanza fondamentale le alleanze fra produttori diversi per fornire prodotti dedicati facilmente integrabili al fine di ottenere soluzioni complete.
L'offerta di prodotti di workflow oggi non può essere certo la soluzione a molti tipi di problemi, così la vera sfida per i fornitori è nello sviluppo di una nuova generazione di prodotti che sappiano combinare la forza di una base di controllo e gestione dei processi con strutture di task e processi più flessibili.

Tecnologie emergenti Un ruolo importante nella dinamica delle funzionalità offerte dai prodotti workflow, sarà giocato sicuramente da tre tecnologie emergenti.

- *Integrazione fra telefonia e computer.* Anche se non è una tecnologia nuova, è solo di recente che i suoi costi sono scesi facendola diventare realmente interessante. Al momento solo pochi prodotti di workflow

offrono una gestione delle chiamate e solo per l'avvio di processi, ma nell'ambito dei servizi al cliente c'è spazio per futuri sviluppi.

- *Mobile computing.* La crescita di questo settore, favorita dallo sviluppo dell'infrastruttura di comunicazioni "wireless", può essere uno stimolo per i prodotti di workflow verso uno sviluppo che vada oltre il modello client-server con client sempre connessi dominante oggi. In realtà tale salto di qualità può essere fatto solo tramite accordi che coinvolgono i produttori di computer, per essere in grado di adeguarsi alle nuove infrastrutture di comunicazione e ai loro requisiti.

- *Agenti intelligenti.* Gli agenti intelligenti sono applicazioni software capaci di eseguire task autonomamente in sostituzione delle persone (questa tecnologia deriva dai decenni di studi sull'intelligenza artificiale). Nel caso del workflow, il loro utilizzo potrebbe essere molteplice: come filtri attivi sull'avvio dei processi, come esecutori di task, come creatori di workflow dinamici in risposta ad eventi di un processo esistente ...

Frammentazione Il mercato dei prodotti di workflow è altamente frammentato: si parla di oltre 240 produttori di questo tipo di software. In realtà, molti dei prodotti forniti non sono prodotti di workflow, ma ne sfruttano il nome (oggi più che mai sulla bocca di tutti) e la confusione su una sua precisa definizione, per affermarsi sul mercato. Ci sono inoltre molti piccoli produttori, con percentuali di mercato irrisorie e anche le aziende più grandi, sono affermate per altri tipi di software e solo di recente hanno cominciato ad offrire prodotti di workflow (per questo anch'esse possono essere considerate "piccole"). Questa caratteristica è peraltro tipica dei mercati nati da poco e basati su tecnologie nuove e in rapida evoluzione e crescita. Si spiega allo stesso modo il forte tasso di "natalità" e "mortalità" di prodotti (e produttori) nuovi, che probabilmente si stabilizzerà nei prossimi anni. In generale, comunque, il mercato è previsto in forte crescita, ma probabilmente, solo pochi produttori saranno in grado di rispondere adeguatamente alle sue esigenze, per questo il numero di produttori esistenti è previsto in diminuzione.

Relazione con tecnologie esistenti Le due tecnologie esistenti più importanti per lo sviluppo del workflow sono:

- applicazioni gestionali;

- tool di sviluppo di applicazioni.

Ad oggi, il workflow è ancora una tecnologia indipendente, ma questa situazione sta per cambiare ed è prevista una sua ampia diffusione e commistione con tecnologie esistenti quali quelle citate. In questo panorama si aprono quindi diverse e, in alcuni casi contrastanti, strade future: integrazione di funzionalità tipiche dei sistemi di workflow in applicazioni gestionali e tool di sviluppo, sviluppo di sistemi di workflow a se stanti caratterizzati da potenza in ambito di gestione di processo, sistemi chiusi o aperti, etc ...

Applicazioni gestionali Il mercato delle applicazioni gestionali, seppur maturo e consolidato, sta subendo e sempre più subirà in futuro degli scossoni dovuti all'introduzione dei prodotti di workflow. La conseguenza più probabile è che, per adeguarsi alle nuove esigenze (e talvolta alle mode del momento), i grossi produttori di software gestionale introducano (ci sono già esempi), funzionalità di workflow nei loro prodotti, o che permettano l'integrazione con prodotti di workflow di altre aziende (magari attraverso alleanze strategiche). Si produrrebbe in questo modo un mercato "ibrido" che andrebbe a tutto vantaggio dei grossi produttori di software gestionale. Non è però escluso che la differenziazione dei prodotti possa portare al successo anche prodotti specifici, magari puntando sulla personalizzabilità ed estensibilità degli stessi

Tool di sviluppo di applicazioni I sistemi di workflow e le applicazioni gestionali sono in gran parte complementari e non devono competere fra loro. Più complesso è il rapporto dei primi con i tool di sviluppo di applicazioni. Al momento i due mercati sono separati, perchè le funzionalità di sviluppo di applicazioni fornite dai prodotti di workflow sono ancora troppo specifiche e limitate. Dall'altra parte, sono pochissimi i produttori di tool di sviluppo ad aver tentato la strada del workflow. ci sono però in futuro possibilità di interazione fra questi due mondi, in vario modo. Può esserci competizione, se i prodotti di workflow si dotano di più ampie e generali capacità di sviluppo di applicazioni, e/o se i produttori di tool di sviluppo decidono di investire di più nel workflow. Può esserci cooperazione, attraverso alleanze fra esponenti dei due gruppi, per poter sfruttare meglio le abilità nei diversi campi.

1.7.2 Gli standard

Non ci sono ancora, come detto, degli standard definiti e affermati per i prodotti di workflow. Questa situazione certo non aiuta lo sviluppo del mercato, perchè gli utenti, in assenza di standard, sono restii ad intraprendere gros-

si investimenti in una direzione o in un'altra, all'interno della straordinaria offerta di prodotti. Ecco perchè i "lavori in corso" in questo ambito sono di particolare interesse e importanza e vedono impegnati un po' tutti i più grossi nomi del software mondiale. Per una trattazione più completa, comunque, si rimanda a quanto già detto in precedenza (sezione 1.6).

1.7.3 La domanda

I recenti successi dei prodotti di workflow sono legati a diversi *trend* più ampi:

- pressione fra le diverse aziende verso una riduzione dei costi;
- una nuova attenzione al "processo", guidata dalla consapevolezza che la competitività è frutto anche di una adeguata revisione dei processi (*Business Process Reengineering*, BPR);
- il passaggio fra processi aziendali basati su documenti cartacei, a processi basati su documenti elettronici.

Settori interessati Storicamente, il settore più avido di prodotti di workflow è quello dei servizi finanziari, per motivi vari che vanno dall'abbondanza di processi inefficienti basati su documenti cartacei alla sua naturale propensione (e disponibilità economica), all'investimento in prodotti di IT. Sebbene ci sia ancora spazio per una crescita del mercato dovuta a questo settore, si prevede che i futuri grossi acquirenti di prodotti di workflow possano essere le pubbliche amministrazioni (con tipici contratti a lungo termine che manterrebbero il mercato a lungo) e il settore della salute (medico-farmaceutico). In generale, comunque, molti altri settori entreranno nel mercato del workflow favorendone l'esplosione.

Applicazioni inter-aziendali Il workflow verrà quasi certamente adottato anche per tutti quei processi che coinvolgono diverse realtà aziendali. Fra questi citiamo solo:

- gestione degli ordini dei clienti;
- servizi ai clienti;
- procedure amministrative interne, come quelle relative ai reclami o al personale.

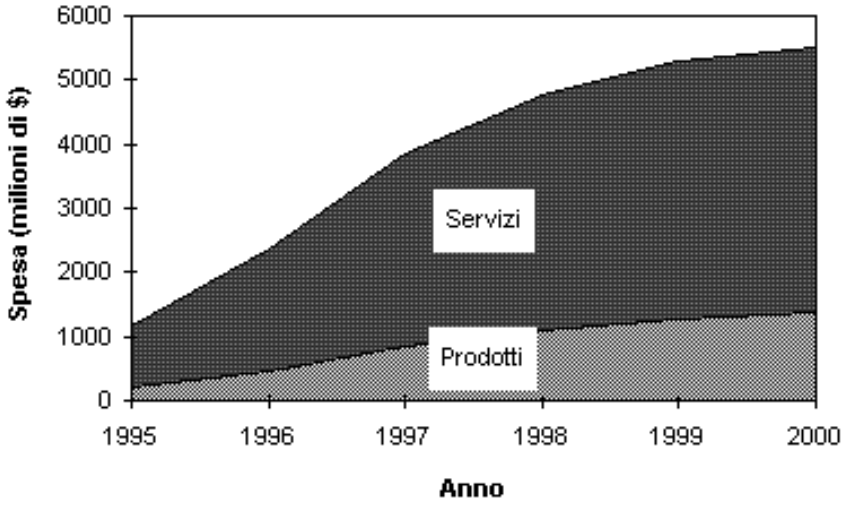


Figura 1.7: Spesa mondiale (milioni di $) in prodotti e servizi di workflow.

I mercati I mercati potenzialmente più importanti per il workflow sono quello nord americano e quello europeo, mentre quello asiatico non è ancora molto sviluppato (anche se in futuro potrà crescere molto). Fra i primi due, contrariamente a quanto si potrebbe pensare, le differenze sono minime, e lo testimonia la presenza di numerosi produttori anche in Europa (lo stesso FlowMark dell'IBM è stato sviluppato a Vienna). Nel vecchio continente, i mercati principali sono Germania e Regno Unito (seguendo la tendenza del software in genere).

1.7.4 I prezzi

Data la notevole concorrenza fra diversi produttori, i prezzi dei prodotti tendono a calare, e in questo senso saranno fondamentali le evoluzioni dei prodotti form-based (i più economici) verso funzionalità oggi tipiche di quelli engine-based.

1.7.5 Lo scenario

Si introducono ora, come conclusione di questa panoramica sul mercato,alcuni grafici che esemplificano meglio le previsioni illustrate sull'andamento del mercato stesso (Fig 1.7 e Fig 1.8), e la tabella 1.2 in cui riassumiamo lo scenario descritto nelle pagine precedenti.

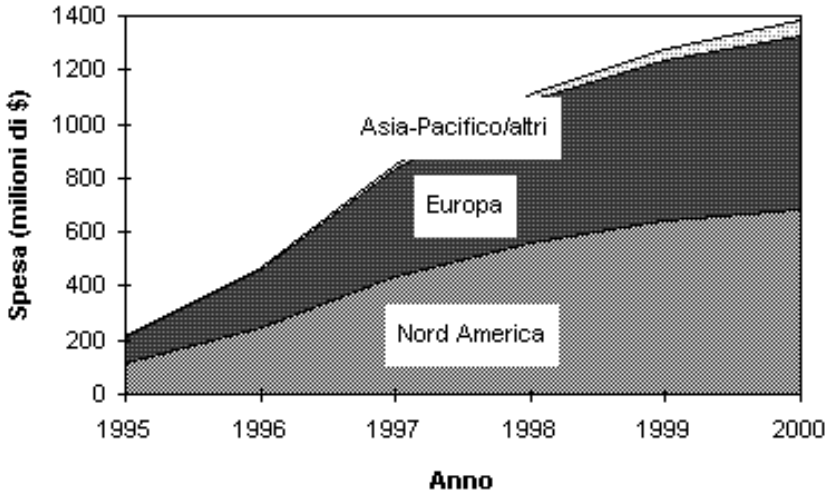


Figura 1.8: Spesa mondiale (milioni di $) in prodotti di workflow per area geografica.

Fattore	Effetto	Implicazioni
Aumento funzionalità nei sistemi a basso costo	Accettazione da parte del mercato di soluzioni ridotte	Crescita nella fascia dei prodotti più economici
	Spinta verso l'alto per i prodotti high-end	Compressione prezzi nella fascia intermedia
Integrazione delle tecnologie emergenti	Miglioramento delle funzionalità del workflow	Crescita del mercato
Sviluppo di standard	Riduzione incertezze degli acquirenti, aumento interoperabilità fra prodotti	Crescita del mercato
Conflitto fra standard diversi	Ritardo nella decisione di acquisto	Ritardo sulla crescita del mercato
Frammentazione e consolidamento	Aumento incertezza degli acquirenti	Inibitore del mercato
Funzionalità di workflow nelle applicazioni gestionali	Implementazione come soluzioni chiuse o proprietarie	Correnti di mercato separate
Alleanze con fornitori di tool di sviluppo di applicazioni general-purpose	Aumento funzionalità dei prodotti di entrambi i settori	Crescita di mercato
Settori interessati	Alcuni settori saturano, altri adottano il workflow	Continua crescita
Applicazioni interaziendali	Adozione ampia e sempre più diffusa	Continua crescita
Mercati geografici	Crescita in Nord America ed Europa; meno nell' area Asia-Pacifico	Crescita del mercato

Tabella 1.2: Fattori che influenzano lo sviluppo del mercato del workflow

Capitolo 2

Modellazione di processi di workflow

L'implementazione e l'automazione di processi aziendali tramite sistemi di workflow passa attraverso vari step, di cui il primo, e più importante, è la progettazione. La progettazione si compone a sua volta di attività diverse che cooperano per produrre un processo completo, efficiente, funzionale. In questo capitolo ci concentreremo in particolare sull'attività di *modellazione* dei processi, mostrando approcci diversi e cercando di evidenziare le differenze fra di essi.

2.1 Introduzione

La prima cosa da notare con particolare attenzione, è che, per i motivi spiegati nel capitolo 1, *non esiste uno standard per la modellazione dei processi di workflow*. Questo comporta una notevole confusione sia per i produttori di sistemi che per gli utenti finali. I primi, infatti, sono costretti ad inventare o riadattare modelli e notazioni grafiche, i secondi si trovano nell'impossibilità di adattarsi a troppe proposte alternative. Non troviamo unità neppure nella ricerca dove ognuno segue strade diverse (anche se spesso si tratta di "diramazioni" di quella tracciata dalla Coalition). Così come nelle interfacce e nelle funzionalità dei prodotti di workflow si cerca uno standard, anche nel campo della modellazione dei processi per l'implementazione tramite sistemi di workflow è necessario trovare un approccio e una notazione comuni e universalmente accettati. Ancora una volta è la W*f*MC a tentare di mettere ordine in questo campo, ma senza avere in realtà una proposta molto precisa. Al contrario, l'intervento della Coalition è molto generale (e generico), e traccia solamente delle linee guida che lasciano ampio spazio ai contributi dei

singoli produttori. D'altro canto, questa è per ora una scelta programmatica del gruppo, che si è concentrato soprattutto sull'interoperabilità fra sistemi diversi. In generale, comunque, molti produttori cercano di attenersi a quanto specificato dalla Coalition, integrandolo però spesso con proprie aggiunte laddove ne sorga la necessità.

2.2 L'approccio della Coalition

Riprendiamo la definizione di processo di workflow data dalla Coalition e che possibile trovare anche in appendice A o nei documenti ufficiali della Coalition stessa [17]. Un *processo aziendale* è un insieme di una o più procedure o *attività collegate* fra loro che collettivamente realizzano un obiettivo aziendale, di solito nel contesto di una struttura organizzativa in cui sono definiti *ruoli* funzionali e relazioni fra essi. Il workflow non è altro che l'automazione di questo processo tramite computer.
Un'altro concetto importante, direttamente collegato a questo, è quello di *definizione di processo*. Questa è la rappresentazione di un processo aziendale in una forma che supporti la manipolazione automatizzata da parte di un sistema di workflow e consiste di una *rete di attività* con le loro *relazioni* e delle informazioni su di esse (*utenti* coinvolti, *condizioni* di innesco o di terminazione, etc.).
Il processo che risulta dalla fase di definizione, infine, si presenta come un insieme coordinato (parallelo e/o sequenziale) di attività connesse fra loro.
Il lavoro del modellatore, dunque, consiste nell'analizzare il processo reale, identificarne le attività salienti, gli attori e le risorse coinvolti e rimettere tutto assieme in modo funzionale ed efficiente. Per maggiore chiarezza e comprensibilità immediata si utilizza una rappresentazione grafica per il processo modellato. Qui comincia la confusione nei simboli e nelle metodologie di cui si diceva sopra. Prova ne abbiamo anche solo osservando i sinonimi di processo forniti dalla Coalition: activity net, rete di Petri, diagramma di flusso ed altri ancora. Fra questi, alcuni, come le reti di Petri o diagrammi di flusso, sono noti ed hanno una rappresentazione precisa e definita, altri, come *Activity Net* (rete di attività), sono propri del mondo del workflow. Proprio quest'ultimo tipo di diagrammi è quello che la Coalition propone come standard per la rappresentazione dei processi.

2.2.1 Activity Net

L'activity net, che nel seguito indicheremo più semplicemente con AN, ha come elemento base della rappresentazione l'*attività* (o *task*). Questa viene

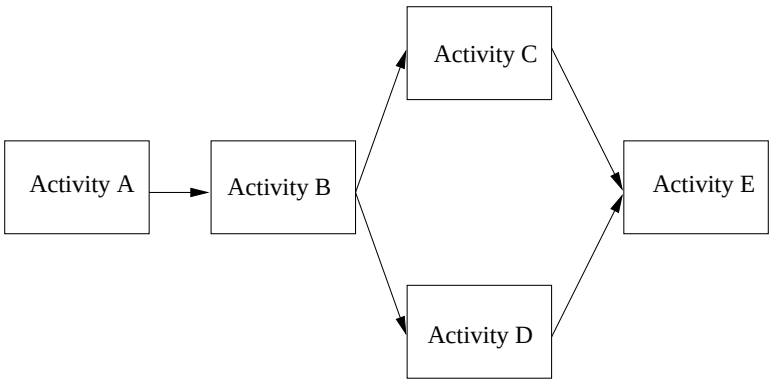


Figura 2.1: Un semplice Activity Net di esempio

rappresentata in modo semplice mediante un rettangolo munito, in genere, di una etichetta che riporta il nome dell'attività stessa. Più attività vengono poi collegate fra loro mediante *archi* orientati, dando origine all'AN (vedi Fig 2.1).
L'esempio riportato non mostra tutte le possibilità di interconnessione fra le attività, che andiamo quindi ad elencare qui di seguito. Come già accennato nella definizione del processo, possiamo avere attività che si susseguono in una sequenza lineare, o correlate le une alle altre in modi più complessi. In particolare, sono tre le configurazioni citate: attività *alternative*, attività *parallele*, *cicli*.

- **Attività alternative: OR** Si hanno quando fra due o più attività che seguono una stessa attività di partenza, è possibile scegliere di eseguirne una sola, in base a particolari condizioni eventualmente poste sugli archi che le collegano. Nella Fig 2.1, è il caso delle attività C e D. La Coalition definisce due costrutti complementari per gestire l'alternativa fra due o più attività: l'**OR-Split** e l'**OR-Join**.

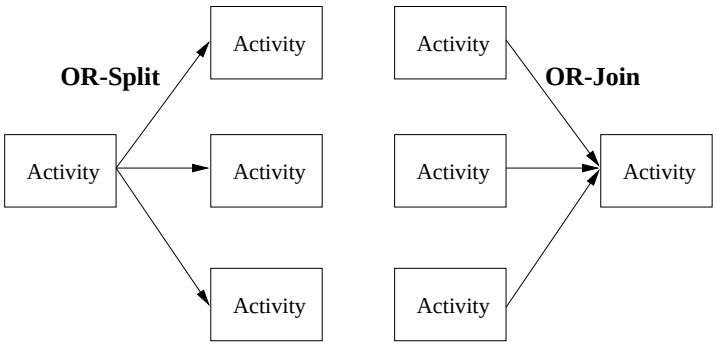


Figura 2.2: OR-Split e OR-Join

- **OR-Split:** è raffigurato a sinistra in Fig 2.2 e rappresenta la separazione dell'esecuzione del processo in due o più possibili percorsi alternativi. Ogni istanza dello stesso potrà intraprendere solo una delle due strade, secondo criteri stabiliti dalle condizioni al contorno.
- **OR-Join:** è raffigurato a destra in Fig 2.2 e rappresenta la convergenza in un'unica attività di due o più percorsi alternativi nell'esecuzione di un processo. Ogni istanza del processo potrà giungere nella attività di "arrivo" solo da uno dei due rami, sempre in virtù delle condizioni al contorno.

- **Attività parallele: AND** Si hanno quando ad una stessa attività di partenza seguono due o più attività che vanno tutte eseguite per portare a termine l'esecuzione del processo. La Coalition definisce due costrutti complementari per gestire il parallelo fra due o più attività: l'**AND-Split** e l'**AND-Join**.

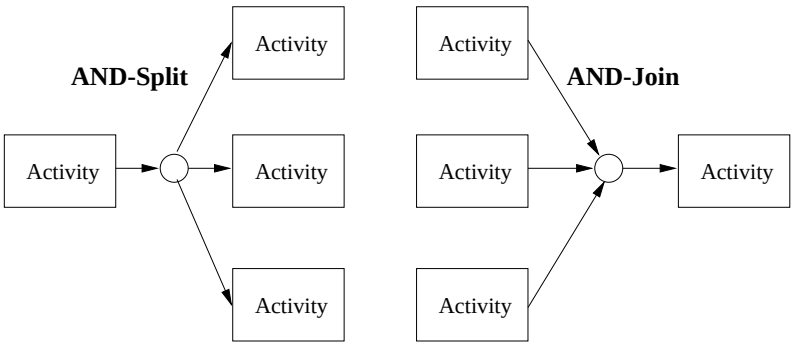


Figura 2.3: AND-Split e AND-Join

- **AND-Split:** è raffigurato a sinistra nella Fig 2.3 e rappresenta la suddivisione dell'esecuzione del processo in due o più percorsi paralleli. Ogni istanza del processo deve, al termine della prima attività, intraprendere tutte e tre le attività seguenti.
- **AND-Join:** è raffigurato a destra nella Fig 2.3 e rappresenta la convergenza in un'unica attività di due o più percorsi paralleli nell'esecuzione di un processo. Ogni istanza del processo potrà intraprendere l'attività di "arrivo" solo quando tutte le attività parallele precedenti sono terminate (si parla per questo dell'AND-Join come punto di sincronizzazione fra attività).

- **Cicli: ITERATION** Si hanno qundo una o più attività devono essere eseguite più volte, iterativamente, fino a che non vengano soddisfatte condizioni che permettano al processo di proseguire. La Coalition fornisce il costrutto di ITERATION che è rappresentato in Fig 2.4. Da notare che all'interno del ciclo possono esserci più attività, in parallelo o in alternativa, e anche cicli, a seconda della complessità del processo che si sta rappresentando.

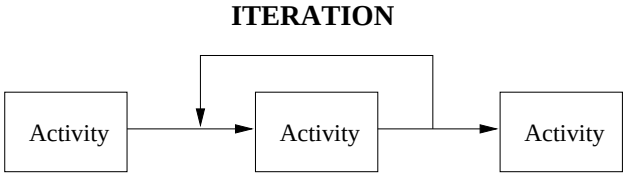


Figura 2.4: Iteration

2.2.2 Osservazioni

Il modello qui presentato è abbastanza semplice e chiaro, ma manca di alcune parti importanti. Nella definizione di processo data all'inizio erano stati sottolineati, scrivendoli in corsivo, alcuni termini. Di questi, però, solo alcuni vengono rappresentati nell'AN così come proposto dalla Coalition. In particolare mancano metodologie di rappresentazione dei *ruoli* e delle possibili *condizioni* cui l'esecuzione è soggetta, nonchè un riferimento preciso ai *dati* utilizzati.

- *ruoli*-Una definizione di processo fa riferimento in genere ad un struttura organizzativa, rappresentata come un insieme di ruoli e delle loro interrelazioni. La mancanza di strumenti grafici per rappresentare questa struttura, e soprattutto i collegamenti fra i ruoli e le attività da svolgere nel processo, risulta assai limitativa per il modellatore, e lo costringe ad utilizzare metodologie proprie (creando confusione negli utenti).

- *condizioni*-Si è parlato di percorsi alternativi (in OR) per le istanze di uno stesso processo: sarebbe importante poter rappresentare in modo semplice le condizioni che determinano la scelta fra questi. La Coalition, però, pur definendo vari tipi di condizioni (vedi Appendice A), non dà uno standard per la loro rappresentazione nell'AN, e anche qui come sopra, dunque, è il modellatore a doversene occupare.

- *dati*- Un processo aziendale in genere utilizza dati di input e produce dati in output; lo stesso sistema di workflow necessita di dati per

controllare e guidare l'esecuzione delle istanze dei processi. Tutto ciò viene notato e commentato dalla Coalition, ma viene poi tralasciato quando si tratta di definire il modello di processo. È a mio avviso una grave mancanza non dare al modellatore strumenti specifici per definire il flusso di dati che "viaggia" sotto al processo e collegarlo al flusso di controllo del processo stesso.

In conclusione, quindi, si può osservare come non ci sia, in una proposta di standard come questa, un concreto tentativo di colmare i vuoti in questo campo. Sembra più che altro un abbozzo, ancora da sviluppare appieno, che propone una direzione, ma non è nè preciso nè esauriente.

2.3 La ricerca

In questa sezione si vuole proporre un approccio alla descrizione del workflow, che, partendo dalla Coalition, cerca di integrarne le mancanze, aggiungendo simboli e specificazioni. Faremo riferimento principalmente ad uno studio inserito nel progetto europeo ESPRIT [19], e ad altri articoli degli stessi autori che riprendono e completano la parte di nostro interesse [20] e [21].

2.3.1 Lo schema di workflow

In questo approccio quello che era l'AN, viene definito come *workflow schema*, e noi in seguito faremo riferimento ad esso come WS.
Il WS è definito come una struttura che descrive le relazioni fra i *task* che compongono un processo. Nel WS vengono descritti quali task devono essere eseguiti, in che ordine, chi ne è responsabile. I WS vengono decritti tramite un linguggio ad hoc, detto Workflow Description Language (WFDL). Non ci soffermeremo sulla descrizione via WFDL dei WS, concentrandoci invece sulla rappresentazione grafica degli stessi.
Così come l'AN, anche la rappresentazione grafica di un WS consiste di task e interconnessioni fra di essi, definiti formalmente nel WFDL. Il simbolo grafico usato per i task è ancora un rettangolo, ma questa volta in esso sono contenute più informazioni, come è possibile apprezzare in Fig 2.5. In particolare, un task è caratterizzato da:

- *nome:* una stringa che identifica il task;
- *descrizione:* poche righe in linguaggio naturale che descrivono il task;
- *precondizioni:* un'espressione booleana che deve essere vera perchè il task possa attivarsi;

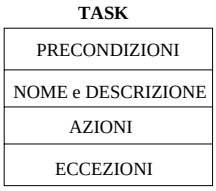


Figura 2.5: Il simbolo grafico per il task di workflow

- *azioni:* una sequenza di istruzioni in WFDL che specifica il comportamento del task;
- *eccezioni:* un insieme di coppie eccezione-reazione per gestire eventi non normali (ancora espresse in WFDL).

Ulteriori novità nella rappresentazione sono rappresentate in Fig 2.6. Il primo simbolo nuovo che troviamo è quello di *Start/Stop*, utilizzato per rendere più chiaro il percorso seguito dal processo, e identificare a colpo d'occhio da dove parte e dove può terminare. Ogni schema ha un solo punto di partenza ma uno o più punti di terminazione. Più interessanti, però, sono i simboli che seguono, che riprendono, estendoli, i concetti di AND e di OR visti nell'AN. In particolare si nota come non si distingue più fra OR o AND, ma solo fra diversi tipi di *fork* (quelli che prima erano *split*), e diversi tipi di *join*.

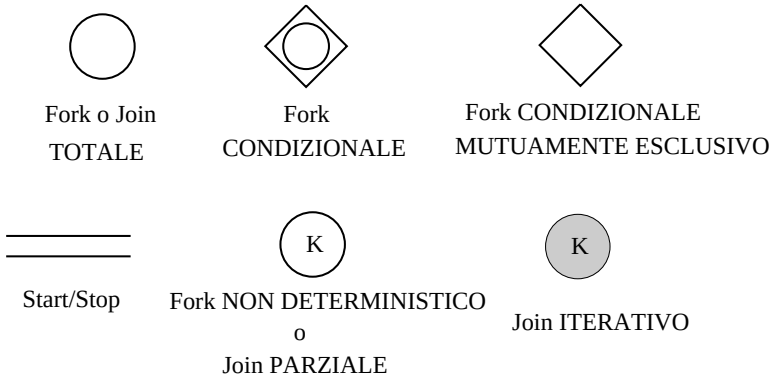


Figura 2.6: I simboli grafici usati nella rappresentazione del workflow

Innanzitutto diciamo che in caso di biforcazioni (fork) o convergenze (join) si parla di particolari tipi di task detti *routing task*, che vengono poi rispettivamente chiamati *fork task* e *join task*. Vediamoli più in dettaglio facendo sempre riferiferimento alla Fig 2.6.

- **Fork task** È preceduto da un solo task, detto predecessore, e seguito da due o più task detti successori. Un fork task può essere:

  - *Totale:* dopo che il predecessore è terminato, tutti i successori sono pronti per l'esecuzione (corrisponde all'AND-Split);
  - *Non deterministico:* alla diramazione è associato un intero k; dopo che il predecessore è terminato, k successori selezionati in modo non deterministico sono pronti per l'esecuzione (non ha corrispondenza nella Coalition);
  - *Condizionale:* ogni successore è associato ad una condizione; dopo che il predecessore è terminato, le condizioni vengono valutate e solo i successori con condizione vera sono pronti per l'esecuzione (non ha corrispondenza nella Coalition);
  - *Condizionale mutuamente esclusivo:* aggiunge al caso precedente il vincolo che solo una condizione può essere vera; così, quando il predecessore è terminato, solo l'unico successore con condizione vera è pronto per l'esecuzione (corrisponde all'OR-Split).

- **Join Task** È preceduto da due o più task, detti predecessori, e seguito da un unico task detto successore. Un join task può essere:
  - *Totale:* il successore è pronto per l'esecuzione solo quando tutti i predecessori sono terminati (corrisponde all'AND-Join);
  - *Parziale:* al join task è associato un intero k; dopo che i primi k predecessori sono terminati, il successore è pronto per l'esecuzione; successive terminazioni di altri predecessori non hanno effetto (non ha corrispondenza nella Coalition, se non quando k vale 1, caso del OR-Join);
  - *Iterativo:* al join task è associato un intero k;ogni qualvolta terminano k predecessori, il join task viene attivato; l'iterazione è implementata resettando a zero il contatore dei predecessori terminati ogni volta che un successore è pronto per l'esecuzione (non ha corrispondenza nella Coalition, se non nel caso di due predecessori e k=1, utilizzato per rappresentare i cicli).

Altre novità introdotte in questa ricerca sono rappresentate dai concetti di *supertask* e di *multitask*, mostrati in Fig. 2.7

- **Supertask** È un particolare tipo di task utilizzato per rappresentare un insieme di task fra loro collegati. In particolare esso ha come un task un nome, una descrizione, delle precondizioni e delle eccezioni; ma manca di azioni, perchè queste sono definite a livello dei task che lo compongono. Seppure dotato di un simbolo simile a quello del task

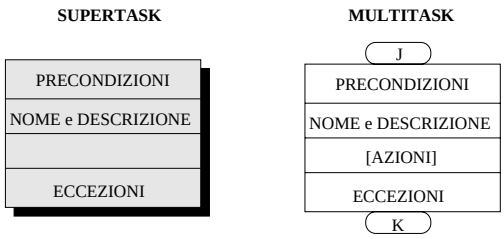


Figura 2.7: Supertask e multitask

semplice, un supertask può essere decomposto in una struttura analoga a quella di un WS: possiede infatti un simbolo di Start e uno o più simboli di Stop. Quando un supertask si attiva, divengono attivi i successori del suo simbolo di Start, mentre basta che sia raggiunto uno dei simboli di Stop perchè il supertask possa considerarsi terminato.

- **Multitask** In molti WS è necessario definire un insieme di task (o supertask), che eseguono esattamente lo stesso lavoro in parallelo, con alcuni parametri che variano fra le diverse istanze. Il multitask è proprio la risposta a questa necessità. Oltre a nome, descrizione, eccezioni, precondizioni ed eventuali azioni (specificate solo se composto di task semplici), il multitask ha un parametro in ingresso, l'intero j; più un'altro, sempre di input, ma opzionale, l'intero k. Il primo indica il numero di istanze del task ripetuto che diventano pronte all'esecuzione quando termina il predecessore del multitask (quanti task identici devono essere lanciati). Il secondo serve a specificare un eventuale valore di *quorum*; ossia quel numero di task all'interno del multitask, che devono essere terminati per considerare terminato anche il multitask stesso (questo numero è ovviamente uguale a j se k non è specificato)

2.3.2 Osservazioni

Rispetto a quanto visto nel lavoro della Coalition, in questa ricerca c'è maggiore approfondimento degli argomenti che ruotano attorno la modellazione di workflow. In particolare c'è un notevole sforzo teso a dare una base solida a questa importante fase della progettazione di sistemi. Vengono infatti analizzati vari tipi di interconnessione fra attività, per avere maggiore flessibilità e rispondenza del modello nei confronti della complessità dei modelli reali. Viene inoltre proposto un linguaggio di definizione del workflow, con una sua sintassi formale che attinge alle Basi di Dati, per dare al modello un assetto formalizzato e univoco. D'altra parte, la notazione è simile a quella già incontrata nella Coalition, seppur con alcune varianti d'obbligo per esprimere

concetti nuovi. Purtroppo non troviamo nel modello una specificazione di come sia possibile mappare il sistema dei ruoli sul processo. Nello studio, infatti, si propone una rappresentazione mediante schema E-R del modello organizzativo e si offre lo spunto per legare ruoli e task (con una relazione molti a molti), ma questa relazione non viene poi investigata a fondo. In particolare si dice che i due modelli, quello organizzativo e quello di processo, vanno definiti separatamente, per avere più flessibilità, e solo in seguito collegati mediante il meccanismo dell'*autorizzazione*. Come però questo possa comparire nel WS, non viene spiegato e rimane a discrezione del modellatore. Anche per quanto riguarda i dati, la situazione è simile a quella della Coalition. In più, però, qui abbiamo i dati di controllo del workflow che compaiono nella definizione dei task.

2.4 Il mercato

Dopo avere visto, seppur in modo rapido e generale, cosa offrono gli standard e un esempio di ricerca, vediamo in questa sezione come il problema della modellazione dei processi di workflow viene affrontato sul campo, ossia dai produttori di sistemi di workflow esistenti. Non entreremo nel merito delle singole scelte implementative, ma cercheremo comunque di dare un'idea dei diversi approcci.

2.4.1 IBM Flowmark

Flowmark è un sistema di gestione di workflow client-server di tipo "engine-based" (vedi Cap 1). Ispirato agli standard della Coalition, di cui peraltro IBM è membro fondatore, è uno dei prodotti di workflow più diffusi e venduti nel mondo. Prevede una parte di definizione dei modelli di processo, cui poi fanno riferimento il motore di workflow e le applicazioni client. Le informazioni sulle definizioni e sull'esecuzione delle istanze dei processi sono mantenute in un database gestito da un DBMS object-oriented. Per informazioni più dettagliate è possibile scaricare dal sito dell'IBM[1] una demo del prodotto che illustra il suo funzionamento; ma qui interessa vedere in particolare le caratteristiche del modello di processo di Flowmark.

Il modello di processo in Flowmark

Gli elementi base del modello (un esempio è riportato in Fig 2.8), che costituiscono il *Process Graph*, sono le *activities*, i *control connectors*, i *data*

[1] Il sito dell'IBM è all'indirizzo *http://www.software.ibm.com/ad/flowmark/* .

connectors, le *conditions*, i *containers*.

- **Activity** Un'attività rappresenta una azione atomica all'interno di un processo aziendale e nel modello è identificata dai nodi del grafo. Graficamente, Flowmark utilizza una ruota dentata per rappresentare un'attività.

- **Control connectors** I connettori di controllo sono utilizzati per specificare l'ordine di esecuzione fra le attività, e nel grafo sono rappresentati da archi orientati.

- **Data connectors** I connettori di dati specificano il flusso di informazioni da una attività ad un altra, e nel grafo sono rappresentati da archi orientati tratteggiati.

- **Conditions** Possiamo avere condizioni sulle attività e/o sulle transizioni (i connettori di controllo del flusso di lavoro). Ogni attività può avere condizioni di partenza e di terminazione, mentre ogni connettore di controllo può avere una condizione di transizione, cioè un'espressione booleana che attiva il percorso in base ai valori dei dati di uscita dell'attività di partenza.

- **Containers** Ogni attività ha un contenitore di dati di input ed uno di output: dal primo riceve i dati necessari all'esecuzione, nel secondo ripone i dati dopo l'elaborazione. Questi dati vengono definiti all'interno di liste di variabili con un nome ed un tipo fra quelli ammessi (string, long, float e strutture). Nel processo, però, i connettori di dati sono definiti a livello di container, e non delle singole variabili.

Per quanto riguarda i costrutti di *branching*, Flowmark fornisce l'equivalente dell'OR-Split (Join), e dell'AND-Split (Join), a n-vie. Tuttavia, la differenziazione fra questi costrutti non è esplicita, ma viene implementata tramite le condizioni di transizione sulle attività e sugli archi del grafo. Questa soluzione da una parte semplifica i grafi, dall'altra, però, rende assai difficoltoso distinguere a prima vista fra un'esecuzione parallela ed una alternativa, complicando il lavoro del progettista che voglia implementare l'uno o l'altro costrutto.
È possibile utilizzare anche costrutti di iterazione attraverso i *blocchi* di attività. questi sono insiemi di attività il cui controllo è effettuato in base alle condizioni di start e di stop del blocco stesso. In particolare, la condizione di uscita serve per gestire le iterazioni all'interno del blocco stesso.
Il sistema dei ruoli in Flowmark, infine, è fisso, ma sufficientemente dettagliato. Oltre al *ruolo*, una persona può vedersi assegnato un *livello* (che è una

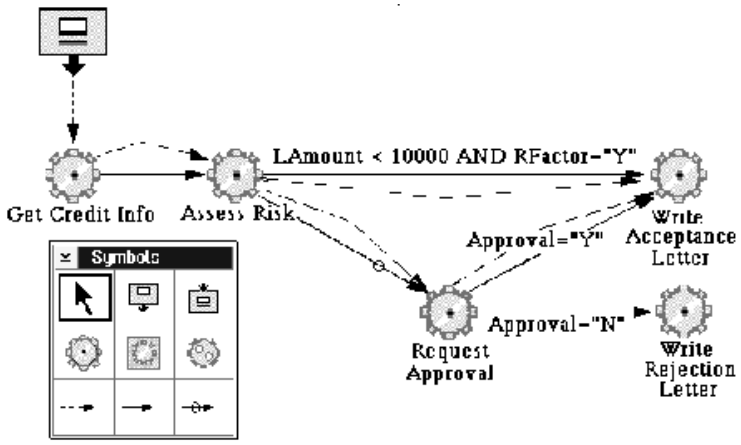


Figura 2.8: Un semplice processo di esempio

misura assoluta, indipendente dai ruoli, delle sue "skill"), dei *sostituti*, può essere membro di una parte dell'*organizzazione*. Tutte queste specificazioni, seppur rigide, sono molto adatte a descrivere una struttura organizzativa anche complessa.

Osservazioni

Questo prodotto, di cui abbiamo potuto visionare solo una "demo", sembra essere molto completo e ben fatto. In particolare, è uno dei pochi prodotti di workflow che permette di mostrare i flussi di dati nello schema del processo, e in questo si pone davanti sia alla Coalition, di cui è "figlio", sia ad alcuni lavori di ricerca. Per contro, gli sviluppatori non possono estendere a piacimento le strutture dati definite in Flowmark, ma devono accontentarsi di ciò che il tool offre.
C'è un buon supporto di tutti i costrutti base del modello proposti dalla Coalition, seppur con qualche complicazione in più dovuta all'unificazione di alcuni simboli (vedi sopra i costrutti di branching).
Il modello organizzativo, infine, pur presente nel prodotto, non entra neppure in questo caso nel modello di processo, ed anzi, il fatto di essere definito separatamente, comporta un continuo passaggio fra le due parti del programma, al momento dell'assegnamento delle attività ai ruoli.

2.4.2 ATI ActionWorkflow

La scelta di analizzare questo prodotto è derivata da due motivazioni: la prima, e ovvia, è stata la disponibilità di una versione completa del pacchetto

(fornitoci dalla ditta FORMULA spa); la seconda invece, è stata l'originalità dell'approccio al workflow che pone questo prodotto lontano da quanto visto finora.
Per una descrizione più dettagliata del prodotto stesso, si rimanda al capitolo 4, qui interessa solo il modello di processo per effettuare il confronto con gli altri modelli presentati.

L'approccio al workflow

L'idea di base del prodotto è la cosiddetta *conversation for action*. In pratica si parte dall'assunto che ogni attività da svolgere prevede due interlocutori (chi la richiede e chi la esegue) che interagiscono in una conversazione. Si può visualizzare questa conversazione tramite un diagramma a stati, in cui i cerchi rappresentano i possibili stati della conversazione, gli archi orientati le fasi in cui è possibile suddividere la stessa (Fig 2.9).

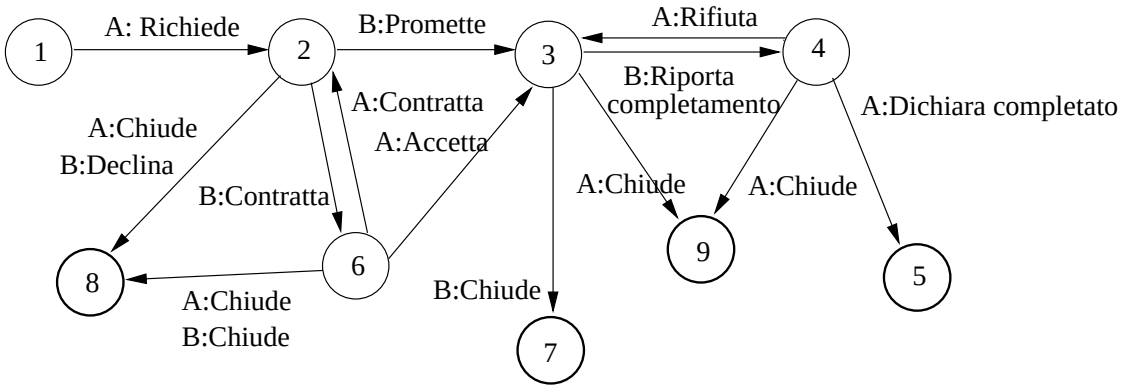


Figura 2.9: Il modello di una conversazione

In essa abbiamo due interlocutori generici, A e B, dei quali il primo inizia la conversazione, specificando le esigenze che vuole soddisfatte al termine della stessa (dette *conditions of satisfaction*). Alla richiesta di A, B può reagire in tre modi: può accettare le condizioni e promettere di soddisfarle (e la rete si evolve nello stato 3); può contrattare sulle condizioni da soddisfare (e la rete si evolve nello stato 6 e da lì in 3 o in 8); può infine rifiutare di soddisfare la richiesta di A (e la rete si porta nello stato finale 8).
Dallo stato 3 la rete può portarsi in 4 quando B dichiara di aver soddisfatto le richieste di A; se questi si dichiara effettivamente soddisfatto, poi, la conversazione termina con successo (stato 5); in caso contrario si può tornare in 3 dove riparte la contrattazione sulle condizioni. Altri comportamenti sono poi possibili, come si vede dal grafo.

Su questo approccio di base è stato costruito il modello di processo del prodotto in questione.

Il modello di processo in ActionWorkflow

Ogni processo aziendale è rappresentato in ActionWorkflow tramite una *mappa*, che è un grafo formato da un insieme di *conversazioni* (o *workflow loop*), collegate fra loro in vario modo.

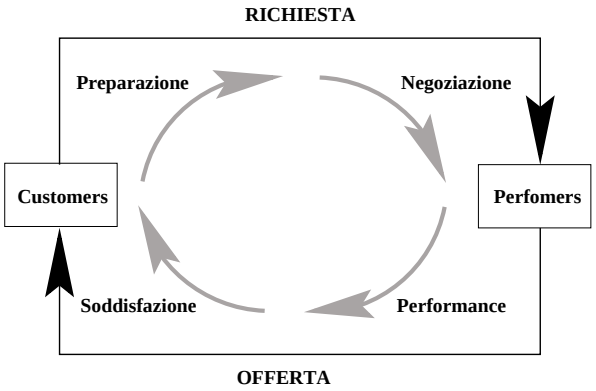


Figura 2.10: La conversazione o workflow loop in ActionWorkflow

Ogni singolo workflow loop coinvolge sempre due attori, un *customer* (cliente), e un *performer* (fornitore), evidenziando il fatto che ogni processo è visto come una *conversazione* fra chi richiede un'azione o un servizio, e chi lo effettua. Ogni workflow è poi diviso in quattro fasi (come si può vedere nella fig 2.10.

1. **preparazione** - Il cliente richiede (o il fornitore offre), la realizzazione di alcune attività specificando le condizioni di soddisfazione.

2. **negoziazione** - Le due parti raggiungono un accordo sulle condizioni.

3. **performance** - Il fornitore completa le attività e dichiara che le condizioni sono soddisfatte.

4. **soddisfazione** - Il cliente conferma che le condizioni sono soddisfatte.

Questa primitiva è la base per costruire poi la mappa del processo di workflow, che risulta quindi organizzata gerarchicamente: c'è infatti il ciclo principale (che rappresenta l'intero processo), e ci sono tanti altri cicli secondari, utilizzati per specificare meglio le attività da compiere nelle varie fasi del processo.

I meccanismi di branching esistono anche in questo prodotto, ed esauriscono le necessità dei processi: connessioni (*link*), seriali, parallele, condizionali.

Il sistema permette poi di definire dei dati da utilizzare nelle applicazioni client e dati di controllo del processo, ma non separa nettamente i due tipi di dati. Non presenta inoltre nessuna primitiva che modelli il flusso dei dati.

Il sistema dei ruoli può essere definito indipendentemente dal processo, ma è importante notare che nel grafo appaiono i partecipanti al workflow, nella forma di clienti e fornitori dei vari cicli. Il mappaggio sui ruoli organizzativi dell'azienda, però, viene fatto a posteriori.

Osservazioni

Il prodotto analizzato è interessante sia per l'originalità dell'approccio, sia per la completezza del modello di processo. Questo infatti, presenta, nell'ottica particolare dell'approccio, tutti i costrutti base già evidenziati negli altri casi. Un punto a favore lo segna la presenza, unico sistema nella nostra breve carrellata, dei partecipanti al workflow direttamente sulla mappa del processo, ma non è una vera e propria primitiva di assegnamento. Non troviamo invece, come detto, la modellazione del flusso di dati e anche la definizione degli stessi lascia spazio a più di un'ambiguità. Ancora una volta, comunque, rimandiamo al capitolo 4 per una dissertazione più ampia sull'intero prodotto.

2.5 Conclusioni

L'analisi condotta in questo breve capitolo, non ha la pretesa di essere esaustiva, ma vuole solo essere di spunto per alcune riflessioni generali sulle problematiche di modellazione dei processi aziendali in ambito di workflow.

La mancanza di standard, già notata nel primo capitolo di introduzione al workflow, si ripercuote negativamente anche in questo ambito, generando confusione e lasciando problemi aperti. La stessa Coalition, come abbiamo potuto vedere, non si occupa con grande entusiasmo del problema della modellazione e si limitaa dettare solo qualche direttiva generale. La ricerca, quindi, cerca di sopperire alle carenze del modello proposto riprendendolo ed ampliandolo. Anche il mercato non si accontenta di quanto fornito dalla Coalition e cerca di sviluppare, con un occhio allo standard, proprie soluzioni.

Qui interessa però evidenziare due aspetti che saranno importanti nel seguito della tesi: la problematica dei dati e quella dei ruoli.

2.5.1 I dati

Il limite già citato nel capitolo 1, esce allo scoperto in questa analisi, sia per quanto riguarda la *definizione* degli stessi, che per la *modellazione* del loro *flusso*. La Coalition per prima trascura l'aspetto dei dati in ambito workflow, concentrandosi solo sul problema del controllo del flusso delle attività; ma anche il mercato e in parte la ricerca sembrano dimenticare l'importanza di questo aspetto.
In realtà, si può ragionevolmente affermare che ogni processo aziendale può essere visto come un insieme di due flussi, quello delle attività e quello dei dati. In particolare, non è sempre vero che il primo è più importante del secondo, ed anzi, in azienda sono molte le procedure dove i dati che passano di ufficio in ufficio sono il *core* del processo. Risulta quindi una mancanza importante quella relativa alla modellazione dei flussi di dati e al loro collegamento con il flusso di controllo nell'attuale panorama del workflow.
Non meno importante è la definizione e la gestione dei dati, che, quando c'è, è povera ed ambigua. C'è spesso confusione, infatti, fra dati del processo (che servono per il suo controllo), e dati delle applicazioni (che costituiscono quel flusso "sommerso" di cui si è detto sopra) e mancano flessibilità e potenza nella loro gestione. In particolare, il workflow sembra trascurare quanto di valido è stato fatto nella tecnologia delle basi di dati e si limita a fornire poco più che qualche variabile di controllo dei flussi.
È auspicabile in questo campo, una forte spinta evolutiva capace di ridare dignità, all'interno dei processi di workflow, ai dati e al loro fluire. In questa tesi si propone proprio una soluzione per questo tipo di problemi, come vedremo in seguito.

2.5.2 I ruoli

Il limite qui evidenziato non è vitale come quello precedente, ma certo ha una sua importanza. Sono in atto tentativi vari di portare nel modello di processo delle primitive che permettano di rappresentare l'assegnamento delle attività ai ruoli, ma quello che si desidera, ancora una volta, è uno standard cui fare riferimento. Nel seguito vedremo anche una possibile soluzione, semplice ma funzionale, per risolvere questo problema.

Capitolo 3

Trattamento dei dati in processi di workflow distribuiti

In questo capitolo si introduce un nuovo modello di replica, chiamato *Dynamic Ownership Model*, (DOM), che riteniamo particolarmente interessante per la modellazione dei flussi di dati in sistemi distribuiti. Del modello verrà data una descrizione, corredata di un esempio per maggiore chiarezza espositiva, in cui saranno evidenziati i concetti chiave e sarà fornito qualche cenno sull'implementazione. Si cercherà poi di analizzarne i limiti e di proporne una soluzione evolutiva. Da ultimo, si analizzeranno le sue caratteristiche confrontandole con quelle dei sistemi di workflow, e fornendo spunti per il successivo lavoro.

3.1 Il modello DOM

3.1.1 Esempio

Descriviamo qui un esempio reale, utile per illustrare il funzionamento del DOM. Ad esso faremo poi sempre riferimento, nel seguito della tesi, anche nei capitoli successivi.

Procedura di inserimento di un minore nelle liste di adozione nazionale(legge 184/83)

La procedura di inserimento di un minore nelle liste di adozione nazionale ha inizio con la segnalazione di un minore abbandonato da parte di privati, enti pubblici o altri istituti al Tribunale dei Minori. Tale segnalazione registrata dalla Cancelleria, viene passata al giudice delegato dal Tribunale dei Minori che avvia il procedimento nel caso rilevi gli estremi per l'apertura di

un caso e quindi di una pratica. Viene quindi condotta una fase istruttoria nella quale vengono raccolti ulteriori dati sul minore (presenza o meno di parenti disposti eventualmente all'adozione, indagini sulle condizioni giuridiche e di fatto del minore, emissione di eventuali provvedimenti urgenti o di sospensione dell'adottabilità). Al termine di questa fase, la pratica passa alla Camera di Consiglio che puó rimandarla indietro per ulteriori indagini o emettere la sentenza sull'adottabilità del minore. In caso di esito positivo, trascorso il periodo stabilito dalla legge per presentare eventuali ricorsi (30 giorni), il minore entra a far parte dell'insieme dei minori adottabili secondo il procedimento di adozione nazionale.

Osservazioni

Osserviamo innanzitutto che l'esempio non copre l'intera procedura così come è descritta dalla legge, ma ciò basta ai nostri scopi.
Notiamo poi, che si tratta volutamente di un esempio in cui esiste un flusso di dati (documenti), fra più persone dislocate in luoghi diversi; vedremo più avanti il perchè di questa scelta. Qui interessa però precisare alcuni punti, relativi agli uffici (o persone), e ai dati coinvolti nella procedura. Per i primi utilizzeremo il termine generale di *ruolo*, col significato già visto nel Capitolo 1; i dati verrano formalizzati in modo da essere poi tradotti in tabelle di un database.

Identificazione dei ruoli coinvolti nella procedura. Dalla descrizione fatta sopra, possiamo estrarre quei *ruoli* all'interno dell'organizzazione *Pubblica Amministrazione*, che hanno parte attiva nel procedimento di adozione nazionale, con riferimento specifico all'inserimento di un minore nelle liste di adottabilità.

- **Cancelleria -** Rappresenta la segreteria del Tribunale, e si occupa di ricevere le richieste di adottabilità e/o adozione e di inserirle nei registri appositi.

- **Giudice delegato -** È il giudice incaricato dal Tribunale di occuparsi dei casi di adottabilità. Deve decidere preventivamente se aprire un caso o meno, e in caso positivo, si occupa delle indagini accurate sul minore in questione.

- **Camera di Consiglio -** È l'organo preposto all'emissione della sentenza di adottabilità, dopo aver esaminato le informazioni raccolte sul minore in questione nella fase istruttoria.

Modellazione dei dati coinvolti nella procedura. I dati su cui puntiamo l'attenzione sono quelli relativi al minore di cui si è richiesta l'adottabilità, che vanno a comporre la *Pratica del Minore*. In essa vengono presentati innanzitutto i dati anagrafici del minore, poi, mano a mano che l'istruttoria procede e vengono alla luce nuove informazioni, si ha l'aggiunta di dati o la modifica di quelli già inseriti. Il caso risulta interessante ai fini del modello di replica da noi proposto, proprio in quanto il *documento* trattato è modificato da diversi agenti in momenti diversi. Annotiamo qui per completezza, che altri documenti possono far parte del processo, noi però ci concentriamo a titolo di esempio sulla pratica citata. Diciamo anche che la pratica stessa può essere vista come un unico documento, ma che a livello di traduzione in un database relazionale, può venire rappresentata anche con più tabelle. Ai nostri fini attuali, comunque, supponiamo di avere un database che contenga almeno la tabella **Pratica**.

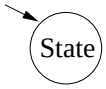
3.1.2 Il modello

Il modello Dynamic Ownership, (DOM), è stato studiato per gestire situazioni in cui un singolo insieme di dati correlati (che indicheremo nel seguito come *data-set*), ha un possessore che cambia durante il proprio ciclo di vita, in base al contesto in cui è inserito e al valore che assume. In particolare, in ogni momento, solo il possessore del data-set lo può modificare, gli altri utenti possono accedervi solo in lettura; è poi il meccanismo di cambiamento dinamico della proprietà del data-set stesso che permette che ognuno degli utenti interessati possa modificarlo a tempo debito.
Illustriamo questa idea aiutandoci col nostro esempio. La *pratica* del minore, può venire inserita solamente dalla *Cancelleria* del Tribunale, poi passa nelle mani del *Giudice* delegato per l'istruttoria, e infine in quelle della *Camera di Consiglio* per la sentenza finale. In termini di proprietà, dunque, ogni nuova pratica è inizialmente in possesso della Cancelleria, poi del Giudice, poi, eventualmente, della Camera di Consiglio.
Per descrivere questa situazione, si può utilizzare il concetto di *iter* di approvazione, descritto mediante un *macchina a stati*, che rappresenta i possibili *stati* che il data-set può assumere, e la dinamica con cui può modificare il proprio stato (il risultato di quest'operazione è il *diagramma stati-transizioni*, che illustriamo nel seguito e che indichiamo brevemente con DST). Per prevenire la concorrenza degli accessi, si utilizza il concetto di unico proprietario per ogni stato. In questo modo si prevengono a monte i problemi di concorrenza, poichè il proprietario può modificare il data-set secondo l'idea del DOM, ma in ogni stato (cioè in ogni possibile momento), è definito un unico proprietario per il data-set stesso; quindi in ogni momento solo un utente

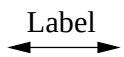
può effettivamente accedere al data-set con permessi read/write. Nel nostro esempio, i possibili proprietari della pratica sono tre, e coincidono coi ruoli identificati sopra: Cancelleria, Giudice delegato, Camera di Consiglio.

Il diagramma stati-transizioni

Uno **Stato** (di un dato) è indicato con un cerchio ed ha un nome significativo. Rappresenta una particolare "situazione" della "vita" del dato stesso.



Uno **Stato Iniziale** è indicato da una freccia entrante ed è il punto in cui è possibile inserire il dato. Ogni grafo deve includere uno ed un solo stato iniziale.



Una **Transizione di Stato** è indicata da una linea orientata, e rappresenta il passaggio del dato da uno stato di origine ad uno di destinazione.

Una **Transizione di Stato Temporizzata** (detta anche **Time-Based**), è indicata da una linea punteggiata e rappresenta una transizione che viene automaticamente eseguita in momenti precisi (giornalmente, mensilmente, etc.) o dopo un determinato intervallo di tempo.

Una **Transizione di Stato per Avocazione**, è indicata da una linea tratteggiata ed è un particolare tipo di transizione *legata* allo stato di destinazione.

Si può notare come siano presenti due tipi di transizioni in più rispetto a quelle "classiche". La transizione temporizzata è di immediata comprensione, quella per avocazione è più complessa. In particolare, quest'ultima risponde all'esigenza di far passare un data-set da uno stato ad un altro "forzando" il sistema delle proprietà. Con essa, infatti, è il proprietario dello stato di arrivo della transizione ad assumere il controllo di un data-set che si trova in uno stato non suo, per farlo passare in uno stato di sua proprietà. Questo meccanismo serve a liberare il data-set dal rigido sistema della proprietà, per gestire situazioni in cui un utente gerarchicamente più elevato possa imporsi su un subalterno. Naturalmente, questa libertà si paga in termini di concorrenza degli accessi.

Per "colorare" il diagramma con i ruoli, assegnando un ruolo ad ogni stato, dobbiamo aggiungere nuovi simboli a quelli precedenti.

Role Name

Ogni stato è collegato ad un singolo **Ruolo** che ne è il possessore. Colori e motivi diversi all'interno degli stati indicano ovviamente ruoli diversi, secondo un'apposita legenda da porre nel grafo.

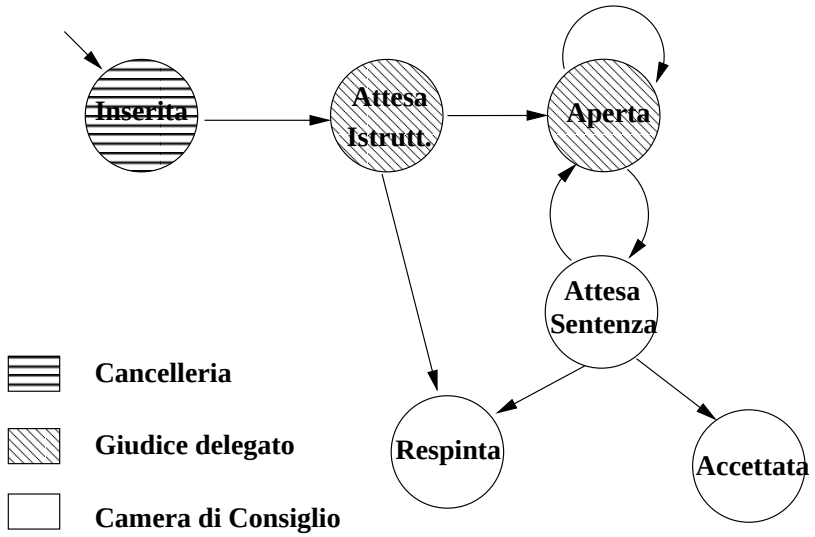


Figura 3.1: Il flusso della pratica di esempio

Modellando il ciclo di vita della pratica secondo questa semplice notazione si ottiene il diagramma stati-transizioni della pratica stessa (Fig 3.1). Collegate al grafo abbiamo poi due tabelle che riassumono i concetti, già rappresentati graficamente, di proprietà degli stati (Tab 3.1), e di transizione di stato (Tab 3.2).

	Stato					
Ruolo	Inserita	Attesa istruttoria	Aperta	Attesa sentenza	Approvata	Rifiutata
Cancelleria	Proprietario	Read/Only	Read/Only	Read/Only	Read/Only	Read/Only
Giudice delegato	Read/Only	Proprietario	Proprietario	Read/Only	Read/Only	Read/Only
Camera di Consiglio	Read/Only	Read/Only	Read/Only	Proprietario	Proprietario	Proprietario

Tabella 3.1: Tabella dei possessori di una pratica

3.1.3 Cenni sull'implementazione del modello

In questa parte si vogliono dare alcune informazioni che serviranno nel seguito, relative all'implementazione del DOM; per una trattazione più completa dell'argomento si rimanda comunque a [22].

Ruolo	Stato					
	Inserita	Attesa istruttoria	Aperta	Attesa sentenza	Approvata	Rifiutata
Inserita	–	Canc.	–	–	–	–
Attesa istruttoria	–	–	Giud. del.	–	–	Giud. del.
Aperta	–	–	Giud. del.	Giud. del.	–	–
Attesa sentenza	–	–	Cam. Cons.	–	Cam. Cons.	Cam. Cons.
Approvata	–	–	–	–	–	–
Rifiutata	–	–	–	–	–	–

Tabella 3.2: Transizioni di stato di una pratica permesse

Il modello, a livello implementativo, prevede che i dati vengano rappresentati come tabelle e replicati su una rete geografica appoggiandosi ad un DDBMS relazionale. Inoltre, per tradurre adeguatamente il DST che modella il flusso dei dati, è necessario modificare alcune tabelle ed aggiungerne alcune di "servizio". Descriviamo brevemente i punti salienti qui citati.

La rete L'ambiente distribuito per il DOM è realizzato su una rete a stella, in cui ogni nodo è un server che mantiene e gestisce gli agenti autorizzati ad eseguire operazioni sui dati. Si assume che ogni server delle rete su cui si implementa il modello possa essere contemporaneamente publisher e subscriber di un *data fragment*, in modo da poter implementare liberamente qualsiasi tipo di *data-flow*. Il nodo centrale è detto *master server* e ha il compito di coordinare la replica in *broadcast* verso i nodi periferici. Le transazioni replicate si muovono in entrambe le direzioni fra server centrale e locale (non è permesso lo scambio diretto di transazioni fra due server locali). I due tipi di pubblicazione sono detti rispettivamente *PUSH* e *PULL* (vedi Fig 3.2).

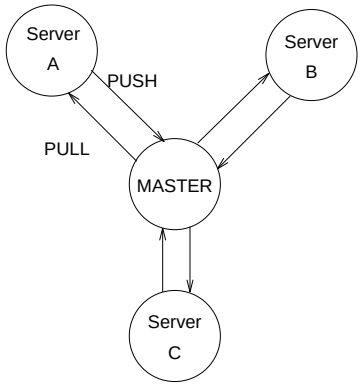


Figura 3.2: Rete a stella

- **PUSH publication -** la pubblicazione nel server locale: contiene i dati collegati al ruolo (o ruoli), locale(i), in accordo al diagramma stati-transizioni. Questi dati sono quelli accessibili in modalità read/write nel sito in questione.

- **PULL publication -** le pubblicazioni nel master server (una per ogni server locale), che contengono i dati che devono essere presenti in ogni server locale. Questi dati sono quelli accessibili solo in lettura nei siti periferici.

I dati replicati si muovono da un server locale verso il master (tramite una pubblicazione di tipo PUSH), e da questo verso tutti gli altri (tramite pubblicazioni di tipo PULL).
La scelta di una rete a stella invece di una connessione point-to-point, è dovuta a due fattori fondamentali:

1. Riduzione della complessità e della ridondanza nella configurazione della replica.

2. Minimizzazione del sovraccarico della rete e delle sue cadute.

Ruoli e siti Il modello proposto descrive un'architettura distribuita per la gestione di flussi di dati che coinvolgono più persone, in siti diversi, con diverse responsabilità sul Sistema Informativo. In particolare, è necessario specificare come i *ruoli* coinvolti vengono mappati nei diversi siti della rete. Nel seguito indicheremo indistintamente il sito o il server che vi risiede, in quanto abbiamo per essi una corrispondenza uno a uno.
All'interno della topologia descritta, al Master Server, che di fatto è il gestore e coordinatore del sistema, non vengono associati ruoli: si vuole in questo modo separare la gestione del sistema dal suo funzionamento vero e proprio (si noti comunque che tale vincolo è dettato da motivazioni logiche, ma può essere rilassato senza comportare problemi).
I siti periferici, invece, sono responsabili dell'esecuzione delle attività da svolgersi sui dati, e, in accordo col modello, devono prevedere la presenza dei ruoli definiti nello stesso. In particolare, ad ogni sito possono venire associati più ruoli, fra quelli coinvolti nel processo, mentre un ruolo deve risiedere in un unico sito. Il mappaggio, sito-ruolo, quindi, si risolve in un'associazione del tipo descritto in Fig 3.3.

Il motivo di un'associazione di questo tipo si spiega facendo riferimento al modello stesso e alle motivazioni che l'hanno ispirato. In esso, infatti, l'idea di base è il superare i problemi di concorrenza in un sistema distribuito sfruttando il concetto di unico possessore del data-set in ogni istante. Quando i

Figura 3.3: Il mapping sito-ruolo

dati sono di proprietà di un ruolo su un server, dunque, solo quel ruolo ha accesso ad essi in modifica. Se altri ruoli sono presenti sullo stesso server, è il sistema (DBMS), locale a gestire la concorrenza, senza sovraccaricare la rete nè creare problemi. Se però lo stesso ruolo fosse presente su di un altro sito, il sistema dovrebbe affrontare la situazione di due possibili modifiche agli stessi dati provenienti da nodi diversi della rete. Per garantire la correttezza dei dati, e quindi dell'intero funzionamento del sistema stesso, quindi, sarebbero necessari quei protocolli distribuiti di gestione delle transazioni che renderebbero "pesante" e lento l'intero sistema, oppure meccanismi di riconciliazione per scegliere quale delle due modifiche accettare sui dati. Questi inconvenienti, però, sono proprio quelli che il DOM vuole evitare, prevenendoli a monte. Il vincolo che prevede un ruolo in un solo sito, inoltre, non risulta così stringente, perchè nella realtà accade spesso che siti diversi (uffici, edifici o sedi), corrispondano a funzioni e ruoli diversi all'interno di un'azienda.
Notiamo infine, che gli utenti reali accedono al sistema in virtù del ruolo che viene loro assegnato all'interno dell'organizzazione, quindi è ancora in base ad esso che vengono effettuati tutti i controlli sugli accessi.

Traduzione del DST Come detto sopra, il sistema cui si appoggia il DOM è un DDBMS relazionale, quindi è necessario passare dalla rappresentazione del flusso di dati in forma di DST ad una concreta mediante tabelle. In particolare, si avranno tabelle che traducono il data-set di cui si è modellato il flusso, più nuove tabelle che implementano i concetti di stato, ruolo, transizione. Per mostrare in modo più chiaro e comprensibile questo passaggio riprendiamo l'esempio introdotto all'inizio del presente capitolo. In particolare, supponiamo che il data-set venga mappato in un'unica tabella che chiameremo *Pratica*, con la seguente struttura:

```
PRATICA(cod_p,descr,..)

create table pratica (
  cod_p int not null,
  descr varchar(20) not null,
  [..]

  constraint pk_ord primary key ( cod_p )
```

)

Supponiamo poi che il sistema distribuito, con topologia a stella, preveda oltre al Master server, tre server periferici che indichiamo con "SERVER1", "SERVER2", "SERVER3".

Tabelle di servizio Le tabelle di servizio servono a mappare le informazioni contenute nel DST e quelle sulla topologia della rete in forma comprensibile ad un DBMS relazionale. Queste tabelle verranno inserite sul Master server e replicate poi a tutti i nodi della rete, perchè sono la base per implementare il flusso di dati. In particolare, servono tre tabelle di servizio per rappresentare correttamente tutte le informazioni; le vediamo qui di seguito.

- **Tabella: STATO** Dalla definizione degli stati e dei ruoli e dall'assegnamento dei primi ai secondi, descritta nel DST, possiamo ricavare una tabella di servizio che chiameremo **STATO(nome,prop,iniz)**. La prima colonna contiene i nomi degli stati ed è una chiave primaria unica, la seconda il ruolo proprietario dello stato, la terza un flag booleano che sarà 1 se lo stato è iniziale, 0 altrimenti. Seguendo l' applicazione d' esempio la tabella diventa:

nome	prop	iniz
INSERITA	CANC	1
ATTESA ISTRUTTORIA	GIUD	0
APERTA	GIUD	0
ATTESA SENTENZA	CAM	0
APPROVATA	CAM	0
RIFIUTATA	CAM	0

- **Tabella: FUNZSRV** Ad ogni ruolo dobbiamo associare il server corrispondente, secondo il mapping descritto sopra; quindi occorre una altra tabella di servizio che chiameremo **FUNZSRV(funz,nomesrv)**. Il primo campo contiene il nome del ruolo (che è chiave primaria), il secondo il nome del server (come definito nella rete). Il master ha ruolo predefinito "MASTER", e solo il ruolo è chiave unica, di conseguenza possiamo avere server che possiedono più ruoli (ma ogni ruolo è associato ad un unico server). Nell'esempio, la tabella diviene:

funz	nomesrv
MASTER	MASTER
CANC	SERVER1
GIUD	SERVER2
CAM	SERVER2

- **Tabella: TRANS** Sempre dal DST, si mappano le informazioni sulle transizioni di stato nella terza ed ultima tabella di servizio chiamata **TRANS(da,a)**. Essa identifica in modo univoco tutte le transizioni di stato, "da" uno stato origine "a" uno stato destinazione. Nell'esempio, la tabella diviene:

da	a
INSERITA	ATTESA ISTRUTTORIA
ATTESA ISTRUTTORIA	APERTA
ATTESA ISTRUTTORIA	RIFIUTATA
APERTA	APERTA
APERTA	ATTESA SENTENZA
ATTESA SENTENZA	APERTA
ATTESA SENTENZA	APPROVATA
ATTESA SENTENZA	RIFIUTATA

Utilizzando queste tabelle possiamo conoscere tutto sugli stati e le proprietà di tutti i server, ad esempio dato uno stato possiamo sapere se è iniziale o no (e quindi consentire l' insert) eseguendo la seguente query su un server:

```
select s.inizi
  from stato s, funzsrv f
 where s.prop=f.funz
   and s.nome=@stato
   and f.nomesrv=@@servername
```

Ricordiamo ancora che le tre tabelle di servizio dopo essere state create sul master vengono aggiunte come articoli di una particolare pubblicazione in broadcast verso tutti i server collegati alla rete, il sincronismo di queste repliche deve essere immediato, in modo tale che la configurazione eseguita sui server trovi tutte le tabelle e le informazioni richieste per il setup.

Nuova struttura delle tabelle Per implementare adeguatamente il meccanismo di cambio dinamico della proprietà dei dati, chiave del modello, non bastano le sole tabelle di servizio, ma è necessario dotare le tabelle che mappano il data-set, di una nuova struttura. In particolare, si dovranno aggiungere due campi e modificare la chiave primaria di cui uno di essi deve entrare a far parte. Precisiamo qui, che la granularità del modello è quella del record, ossia è sulla singola tupla presente nella tabella che si implementa il meccanismo descritto nel DOM.

- **Campo: Stato** Per identificare il ruolo proprietario di una determinata tupla, è necessario memorizzare, per ognuna di esse, lo stato attuale,

fra quelli permessi nel DST, in cui si trova. Per fare questo, il modo più semplice è aggiungere alle tabelle dell'applicazione di cui si implementa il flusso, un nuovo attributo che chiamiamo **Stato**. Questo campo deve avere lo stesso tipo (varchar(10) nel nostro caso), del campo "nome" della tabella di servizio "STATO", e può assumere tutti e soli i valori corrispondenti agli stati presenti nella stessa tabella (e nel DST).

- **Campo: Validità** Per implementare il passaggio di proprietà di una tupla delle tabelle secondo il modello proposto, è necessario poter distinguere, fra le varie copie dei dati presenti sulla rete, l'unica che in un dato istante può essere considerata "originale", e quindi modificabile dal ruolo proprietario. Il metodo adottato nel modello è l'ulteriore aggiunta alle tabelle, di un attributo che chiamiamo **Validità**. Questo campo è un intero che, per convenzione, dovrà avere valore "0" per la tupla "originale" (detta anche "valida"), valore non nullo negli altri casi. Proprio perchè serve ad identificare una particolare tupla, il campo Validità deve entrare a far parte della chiave primaria delle tabelle.

  Vediamo più a fondo il funzionamento del sistema in relazione a questo campo, vista la sua importanza. Quando un utente, con ruolo corretto, accede ad una tupla per modificarla, il sistema recupera la tupla con "Validità = 0", controlla lo stato della stessa (verificando la compatibilità con quelli assegnati al ruolo in questione), e se tutti i controlli danno esito positivo, permette l'aggiornamento. Questo si risolve nella modifica del campo Validità al massimo valore dello stesso presente nella tabella incrementato di una unità, e nel successivo inserimento di una nuova tupla con le modifiche apportate dall'utente, e con il campo Validità forzato a zero. Nelle successive operazioni solo questa sarà originale, le altre servono a mantenere la storia ("versioning"), delle modifiche subite dalla tupla durante il flusso (in particolare valori alti indicano copie recenti, valori bassi le più vecchie).

Dopo le modifiche descritte, quindi, la struttura delle tabelle risulta diversa; diamo qui la nuova versione della tabella *Pratica* utilizzata nell'esempio:

```
PRATICA(cod_p,descr,..,stato,val)

create table pratica (
  cod_p int not null,
  descr varchar(20) not null,
  [..]
  stato varchar(10) not null,
  val int not null
```

```
  constraint pk_ord primary key ( cod_p , val )
)
```

Trigger Le operazioni di modifica dei dati devono essere gestite in modo tale da proteggere i dati da accessi non autorizzati. Alcune situazioni sono gestite direttamente dal DBMS quindi non vengono prese in considerazione (prima di tutto la gestione delle chiavi primarie); l' unico compito che rimane da svolgere è il controllo della compatibilità della transazione da effettuare con lo stato in cui si trova il dato. Queste operazioni vengono effettuate tramite *trigger* posti sulle tabelle i quali si occupano di gestire il controllo su insert, update e delete delle tuple, in accordo a quanto definito nelle tabelle di servizio.

Scenario di replica Diamo qui un breve cenno della configurazione generale del sistema per un'implementazione pratica del modello.

Si vuole configurare il sistema nel seguente modo:

- Ciascun sito periferico pubblica verso il Master Server i dati sui quali ha diritti di possesso
- Il Master Server, che contiene una copia di tutti i dati, genera una pubblicazione di tipo "restricted" per ciascun sito periferico. Gli articoli di questa pubblicazione contengono la frazione di dati sui quali il sito periferico ha diritti di sola lettura. Le informazioni necessarie al Master Server sono contenute nel Dizionario dei Dati (vedi sotto).
- Ciascun sito periferico richiede di sottoscrivere tutte le pubblicazioni del Master che gli competono.
- Ogni volta che su un server locale un dato viene modificato, un pubblicazione PUSH porta l'aggiornamento al Master server, che poi tramite le pubblicazioni PULL, lo rende visibile (in lettura), a tutti gli altri nodi della rete.

Le operazioni di configurazione per ottenere quanto richiesto sopra,prevedono che si operi sul sito master e sui siti periferici nel seguente modo:

Sito Master: le operazioni che devono essere definite sul Master server sono le seguenti:

- Inizializzazione della replica e del distribution server.

- Definizione delle stored procedures di redistribuzione delle RPC provenienti dai server.
- Definizione delle pubblicazioni PULL, una per ogni subscriber, contenenti gli articoli richiesti in lettura e relativi agli stati non di proprietà di ciascun server.
- Definizione delle tabelle di servizio e pubblicazioni per la loro replica verso tutti i siti remoti.
- Inserimento nelle tabelle di servizio dei valori provenienti dalla topologia della replica, dal grafo e dalla configurazione dei server.
- Definizione delle sottoscrizioni alle pubblicazioni dei server remoti.

Siti periferici: le operazioni che devono essere compiute nei server periferici sono le seguenti:

- Inizializzazione della replica come publisher/subscriber.
- Definizione delle pubblicazioni PUSH partizionate agli stati di proprietà del server.
- Definizione delle RPC per l' inserimento locale di righe di proprietà.
- Definizione del trigger sull'insert, l'update e il delete.
- Definizione delle sottoscrizioni alle pubblicazioni del Master server.

Ricordiamo che la gestione della configurazione della replica viene fatta attrverso script in Transact-SQL, i quali sono generati in automatico da un piccolo tool C.A.S.E. descritto ancora in [22], e di cui parleremo nel Capitolo 6.

3.2 Evoluzione del modello

La descrizione del modello presentato in questo capitolo e sviluppato precedentemente a questa tesi, offre lo spunto per alcune riflessioni sui suoi limiti e le sue potenzialità da sviluppare. Il primo passo pratico nel mio lavoro di tesi è stata proprio l'analisi critica del modello, anche alla luce delle conoscenze sul workflow. In questa sezione si cercherà di illustrare il lavoro svolto in questo ambito.

3.2.1 Modifiche iniziali

Stati finali Osservando il diagramma *stati-transizioni* e pensando alla classica forma di uno *state diagram*, è possibile notare subito la mancanza nel primo del concetto (e del simbolo), di *stato finale*. Questa carenza è abbastanza grave, soprattutto dal punto di vista logico: il flusso dei dati in una qualsiasi procedura aziendale, oltre ad avere un principio, ha naturalmente anche una fine. Per tornare all'esempio dell'adozione, una volta che il minore è stato dichiarato adottabile, per la procedura di inserimento nelle liste di adozione nazionale la sua pratica viene archiviata. Potrà magari essere riutilizzata in altre procedure, ma per quanto rigurda la procedura da noi modellata, il flusso termina con l'accettazione o il rifiuto della richiesta. Anche volendo modellare i possibili ricorsi, aggiungeremmo degli stati, ma dopo il tempo tecnico ammesso dalla legge per un ricorso, la pratica ricadrebbe necessariamente in uno dei due stati già citati. Allo stesso modo si può pensare alla procedura aziendale di ricevimento e gestione di un ordine da cliente: la pratica in questione, una volta che l'ordine è stato evaso, termina il suo flusso finendo in un archivio dove può al massimo essere consultata (diviene permanentemente read/only).
In sostanza, dunque, la prima proposta concreta di modifica al modello, è l'aggiunta del concetto di *stato finale*, e del relativo simbolo grafico per il diagramma stati-transizioni. Per quest'ultimo, nella descrizione, dovremo aggiungere la seguente informazione.

Uno **Stato Finale** è indicato da un doppio cerchio ed è il punto in cui termina il processo sul dato. Ogni grafo deve includere almeno uno stato finale.

Questa aggiunta comporta una modifica del diagramma stati-transizioni che diventa come in Fig 3.4. Si noti un'altro aggiornamento del grafo: sulle transizioni sono state aggiunte delle etichette per meglio spiegare il significato dei passaggi da uno stato all'altro. Questa modifica, che qui è marginale, verrà richiamata in seguito, relativamente ad altre considerazioni sul modello.
Tornando all'inserimento del concetto di stato finale nel modello, va notato che esso comporta anche la modifica della tabella di servizio che associa ad ogni stato un proprietario e un flag che indica se lo stato è iniziale o meno. Ad essa va aggiunto un secondo flag che indichi se lo stato è finale o meno. In alternativa, è possibile utilizzare solo il primo flag, considerando il campo come un intero con valori da 0 a 3. In questo modo possiamo gestire tutte le possibilità che si hanno con due flag. Un possibile "mapping" è: `flag=0` se lo stato non è nè iniziale nè finale; `flag=1` se lo stato è iniziale ma

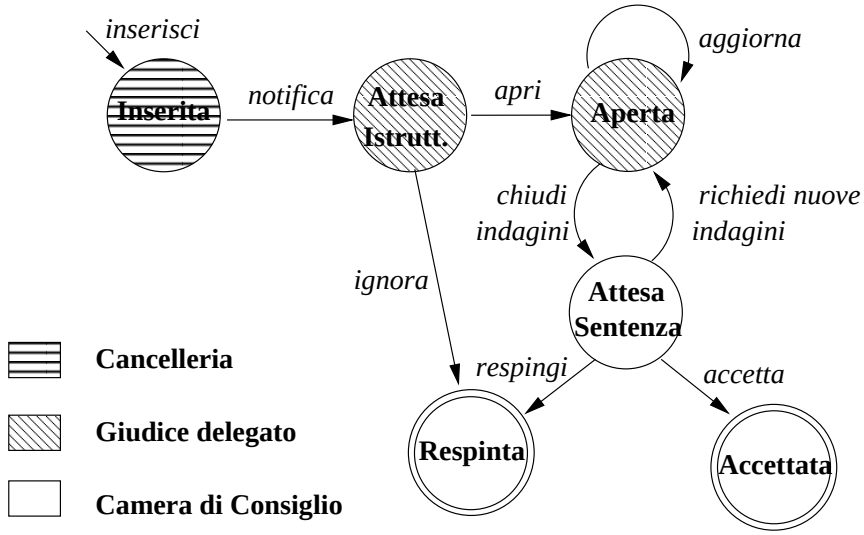


Figura 3.4: Il nuovo diagramma stati-transizioni della pratica

non finale; `flag=2` se lo stato è finale ma non iniziale; `flag=3` se lo stato è contemporaneamente iniziale e finale.

Un'altra decisione importante riguardo lo stato finale è sull'effettivo comportamento da tenere quando il dato giunge in questo tipo di stato. La soluzione più logica è quella di rendere il dato read/only per tutti gli utenti, in accordo al fatto che il suo ciclo di vita termina con l'arrivo in quello stato. Graficamente ciò significa che non potranno esserci transizioni in uscita da uno stato finale, neppure in caso di autoanelli. Il proprietario dello stato, in quest'ottica però, non ha nessun privilegio rispetto agli altri utenti riguardo i dati validi che si trovano in questo stato. La semantica che attribuiamo allo stato finale, infatti, prevede che uno stato finale sia uno stato in cui i dati non sono modificabili. Questa restrizione, derivata dalle motivazioni logiche precisate sopra, non risulta comunque troppo vincolante. Qualora si decidesse di rilassare il vincolo, per implementare flussi più complessi, basterebbe riportare il comportamento dello stato finale a quello comune agli altri stati.

Tabelle collegate Fino ad ora si è parlato del modello rispetto ad un dato generico, che nell'esempio è una tabella di un database, ma non si è entrati nello specifico di situazioni più complesse. In particolare, cerchiamo di illustrare il problema con un'estensione dell'esempio.

Aggiungiamo al database di esempio, oltre alla tabella "pratica", anche una nuova tabella legata ad essa da un vicolo di chiave esterna. Vogliamo modellare la situazione in cui una "pratica" sia composta di "fascicoli" secondo lo

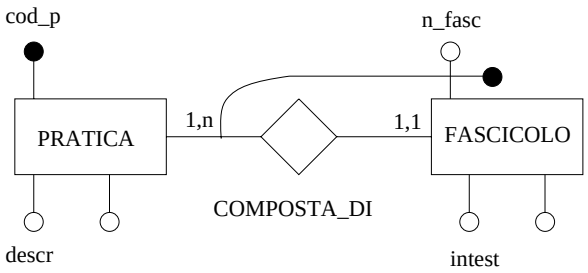


Figura 3.5: Schema E-R della pratica

schema E-R di Fig. 3.5. Tradotto in termini di relazioni ciò significa avere due relazioni siffatte:

```
PRATICA(cod_p,descr,[..])
        PK cod_p

FASCICOLO(n_fasc,cod_p,intest,[..])
        PK n_fasc,cod_p
        FK cod_p references PRATICA.cod_p
```

Fino ad ora, nel modello si è parlato di cosa accade alla tabella "pratica", ma non alla tabella "fascicolo" che da essa dipende direttamente per un vincolo di chiave esterna. In particolare, della prima possiamo modellare il diagramma stati-transizioni e implementare il DOM, aggiungendo i campi "stato" e "validità", mentre per la seconda scegliamo , in prima istanza, di non modificare nulla. La situazione dopo l'implementazione del DOM è allora:

```
PRATICA(cod_p,descr,[..],stato,val)
        PK cod_p,val

FASCICOLO(n_fasc,cod_p,intest,[..],)
        PK n_fasc,cod_p
        FK cod_p references PRATICA.cod_p
```

Supponiamo adesso di effettuare delle operazioni partendo da questi dati:

```
PRATICA:
```

cod_p	descr	[..]	val	stato
1	A	..	0	ATTESA ISTRUTTORIA
1	A	..	1	INSERITA
2	B	..	0	APERTA

`FASCICOLO:`

n_fasc	cod_p	intest	[..]
1	1	x	..
2	1	y	..
3	1	z	..
1	2	k	..
2	2	l	..

- *Scenario 1:* passaggio di stato di una pratica: la pratica con codice 1 da ATTESA ISTRUTTORIA ad APERTA.
  - PRATICA:

```
* tupla (1,A,..,0,ATTESA ISTRUTTORIA) modificata in
        (1,A,..,2,ATTESA ISTRUTTORIA);
* nuova tupla  (1,A,..,0,APERTA)
```

  - FASCICOLO: inalterata.
- *Scenario 2:* modifica di un fascicolo della pratica 2 senza passaggio di stato: nella tupla con chiave (2,2) l'intestazione passa da "l" a "m" (la transizione è l'autoanello APERTA-APERTA).
  - PRATICA: inalterata.
  - FASCICOLO:

```
* tupla (2,2,l,..) modificata in
        (2,2,m,..);
```

Osservazioni e commenti Nello Scenario 1 si vede bene che se avessimo avuto anche per la tabella FASCICOLO i campi aggiuntivi (stato e validità), avremmo avuto altri update da fare senza avere vantaggi. Va notato infatti che anche se la pratica cambia proprietario, non necessariamente cambia la sua descrizione fatta attraverso i fascicoli, che risulta quindi indipendente dallo stato della pratica stessa. In realtà questo dipende molto da come è realizzato il database e dalle singole procedure da implementare, ma

si può risolvere facilmente anche il caso di update espliciti della descrizione della pratica.
Questa situazione si presenta nello Scenario 2. Qui avere campi aggiuntivi potrebbe servire per una questione di "versioning"; altrimenti e' meno dispendiosa ancora la soluzione qui proposta. Puo' pero' nascere un conflitto in un caso particolare (ipotizzando che la pratica sia affidata ad almeno due Giudici che operano in team): l'utente Giudice X esegue l'operazione descritta nello Scenario 2, mentre un secondo utente Giudice Y vuole portare il record con cod_p = 2 da APERTA ad ATTESA SENTENZA. Se X accede solo a FASCICOLO senza "lockare" il record corrispondente in PRATICA, si puo' avere una situazione non voluta. Si puo' ovviare a questo inconveniente facendo si' che in seguito ad una modifica della descrizione della pratica (un update su FASCICOLO) venga eseguito un update anche nella tabella PRATICA, segnalando con una nuova "versione" (incremento del campo validità) che e' avvenuto qualcosa. Così facendo il secondo utente dovrebbe attendere la fine dell'operazione del primo per poter effettuare la propria.
Altro possibile quesito e' sulla proprieta' della tabella FASCICOLO, ossia su chi detiene i diritti di scrittura fra i ruoli presenti. La nostra idea, molto semplice, e' che i record di FASCICOLO con cod_p = n siano modificabili solo da quel ruolo che ha diritto di modifica sul record con chiave (n,0) nella tabella PRATICA, cioe' sulla versione valida del record cui la descrizione si riferisce.
A questo punto proviamo a pensare a cosa accadrebbe se aggiungessimo anche a FASCICOLO i campi ausiliari. Il primo problema è la scelta del diagramma a stati: è lo stesso di PRATICA o è diverso? Trascuriamo il secondo caso, perchè questo comporterebbe un nuovo diagramma e una nuova e diversa implementazione della replica. Nel primo caso, dunque, ipotizziamo che FASCICOLO abbia lo stesso data-flow di PRATICA, cui è legata dalla chiave esterna: non sembra avere molto senso l'idea della modifica ad entrambe le tabelle. Come detto già sopra, sarebbe spesso inutile e ridondante, mentre nei casi in cui può servire, si possono adottare soluzioni alternative.
In base a queste considerazioni, la soluzione proposta è quella di identificare, all'interno del flusso di dati implicato dalla procedura che si vuole implementare, la tabella (o le tabelle) indipendenti, cioè che non hanno nessun vincolo di integrità che le lega ad altre. Di queste sole verranno modellati i singoli diagrammi stati-transizioni, e verrà implementata la replica secondo il DOM. Si dovranno poi identificare le tabelle che dipendono dalle prime, non solo per vincoli di integrità, ma anche logicamente, nell'ambito del flusso di dati. Per queste non serve il grafo, semplicemente esse seguono le tabelle cui sono legate. Questo legame verrà implementato tramite lock opportuni quando un utente accede a record collegati nella tabella principale.

Per dare a queste osservazioni un carattere più preciso diamo alcune definizioni che fissano i concetti sopra espressi.

- **Oggetto Concettuale** Con oggetto concettuale intendiamo un aggregato di dati visti come un'unica entità nell'ambito di un particolare flusso di dati. Nell'esempio fatto, la pratica, intesa non come tabella, ma come l'insieme di tutti quei documenti che la compongono, era l'oggetto concettuale. Esso, quando viene modellato in forma di relazioni (per essere inserito in un database), verrà mappato su più tabelle, legate fra loro da una struttura gerarchica. Al culmine della gerarchia sta la *Tabella Leader*, sotto stanno le *Tabelle Subordinate*.

- **Tabella Leader** È la tabella principale, priva di dipendenze, in cui viene mappato l'oggetto concettuale. Su di essa verrà costruito il diagramma stati-transizioni e implementata la replica. Nell'esempio è la tabella PRATICA.

- **Tabella Subordinata** È una tabella legata alla tabella leader da vincoli di integrità (chiavi esterne), e da legami logici all'interno dell'oggetto concettuale. Spesso è in relazione di *part-of* con la tabella leader che "segue" nel flusso, secondo le regole date sopra. Nell'esempio era la tabella FASCICOLI.

3.3 Confronto con i sistemi di workflow

Considerando le idee di base da cui parte il DOM e pensando a cos'è il workflow, è inevitabile notare delle somiglianze e dei punti di contatto. In entrambi i casi si parla di flussi (di dati nel DOM, di attività nel workflow), di ruoli cui vengono assegnate funzioni (operazioni da compiere), di processi o procedure da automatizzare. In più, inoltre, è possibile mettere a confronto i diagrammi stati-transizioni (che qui indicheremo più semplicemente con DST), con gli Activity Net (AN), trovando convergenze notevoli.

3.3.1 DOM vs workflow

Il primo approccio al confronto è stato fatto provando a vedere se il DOM potesse essere visto come un sistema di workflow, pur con tutte le limitazioni del caso.
Il tipo di modellazione che il workflow fa dei processi aziendali, può essere a ragione definito *orientato all'attività* (o al processo). L'approccio "classico", infatti, vede un processo come "un insieme di *attività* coordinate fra più

persone per un fine comune". Il fulcro del modello, quindi, è proprio l'attività (o task), che è il "mattone" con cui il processo viene costruito. A conferma di ciò, anche l'AN, che modella graficamente il processo stesso, altro non è che una rete di attività collegate in vari modi.
Il DOM, in quest'ottica, può essere visto come un particolare tipo di sistema di workflow *orientato ai dati*. In esso si parte dall'osservazione che la gran parte dei processi aziendali automatizzati mediante workflow sono applicazioni di tipo gestionale, che per loro natura coinvolgono una grande quantità di dati. Il DOM offre allora un cambio di prospettiva, mettendo al centro del modello i dati su cui si opera, piuttosto che le operazioni da fare su di essi.
Per riassumere questa prima osservazione potremmo dire che mentre per l'approccio dei sistemi di workflow, un processo è "un insieme di attività che operano su dati", per il DOM esso è "l'evoluzione dei dati sotto l'azione di varie attività".
Cerchiamo ora di mettere in evidenza il legame trovato sopra, facendo riferimento alla rappresentazione grafica dei processi nei due modelli. In particolare riusciremo a mostrare come da un DST sia sempre possibile ricavare il corrispondente AN; mentre il passaggio inverso possa non essere "indolore".

Passaggio da un DST ad un AN Elenchiamo alcune regole che permettono di mappare un DST in un AN.

- Una *transizione* diviene una *attività* con lo stesso nome (quindi nell' AN dovranno risultare tante attività quante erano le *transizioni* del DST).

- La *transizione* iniziale (quella che entra nello stato iniziale), diviene una *attività* senza frecce in ingresso (*attività* iniziale) ed è la prima *attività* da disegnare nel grafo.

- Dopo ogni *attività* ci si trova nello stato in cui portava la *transizione* corrispondente. Possono quindi essere eseguite (*Or-Split*), tutte le *attività* corrispondenti a *transizioni* permesse da quello stato. Si ripete questo procedimento fino ad aver esaurito tutte le *attività* e tutti i possibili percorsi.

- In base al punto precedente, una transizione che porta in uno stato finale corrisponde ad un'*attività* che non può venire seguita da nessun'altra *attività* (da uno stato finale non si esce con alcuna transizione), diviene cioè una *attività* finale. Rimane il problema di segnalare che se due transizioni portavano nello stesso stato finale, anche le due *attività* corrispondenti devono portare l'AN alla stessa terminazione. La soluzione

proposta è quella di inserire per ogni stato finale una *attività* fittizia chiamata "Fine" (stati finali diversi avranno *attività* di fine diverse, ad esempio "Fine1", "Fine2" ...). Queste non vanno contate quando si confronta il numero di *attività* e *transizioni*.

Forniamo ora un esempio di passaggio per rendere più chiare le regole elencate. Supponiamo di avere il DST di Fig 3.6 e di volerlo trasformare in un AN (che vediamo in Fig 3.7): il procedimento da seguire è indicato qui di seguito.

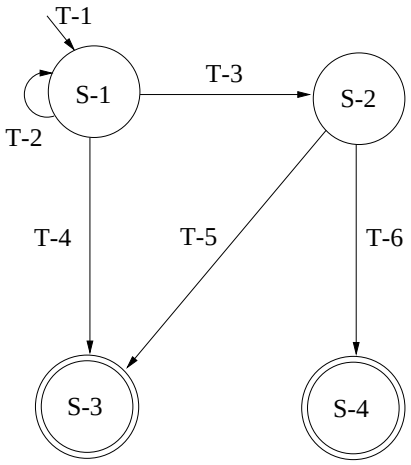


Figura 3.6: Un DST di esempio

- Abbiamo 6 transizioni, quindi si avranno 6 attività.
- Abbiamo 2 stati finali, quindi si avranno 2 attività di "fine" diverse.
- Costruzione:
  - La transizione T-1 diviene l'attività iniziale.
  - Dopo T-1 posso eseguire T-2, T-3, T-4.
  - T-2: dopo T-2 posso eseguire T-2, T-3, T-4.
  - T-3: dopo T-3 posso eseguire T-5, T-6.
  - T-4: T-4 porta in uno stato finale: inserisco Fine-1.
  - T-5: T-5 porta nello stesso stato finale cui portava T-4: la collego a Fine-1.
  - T-6:T-4 porta in uno stato finale diverso dal precedente: inserisco Fine-2.

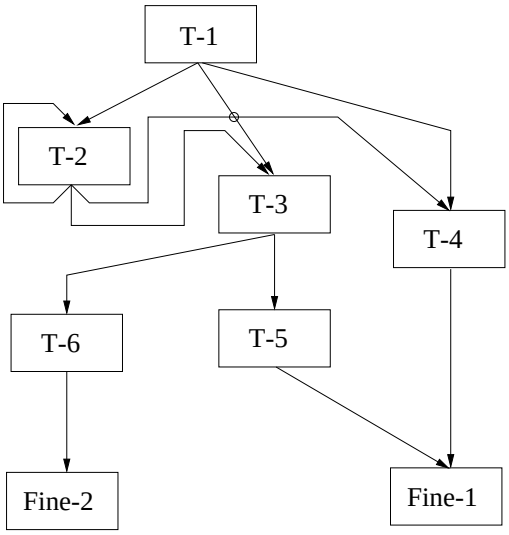


Figura 3.7: L'AN corrispondente al DST di esempio

In un passo successivo si può avere l'assegnazione degli stati ai ruoli. Con la convenzione poi che ogni *transizione* può essere effettuata solo dal ruolo possessore dello stato di partenza della stessa (a parte le *advocation* e le *time-based*), diviene immediato assegnare un ruolo alle *transizioni* e quindi alle *attività* dell'AN, "colorando" quest'ultimo come il DST. Questo passo è anche un tentativo di sopperire al problema individuato nel Capitolo 2, relativo alla mancanza di un adeguato costrutto per rappresentare l'assegnamento dei ruoli alle attività in un AN (per un esempio vedi il Capitolo 5).

Passaggio da un AN ad un DST Prima di elencare anche per questo caso alcune regole di passaggio, bisogna imporre alcune restrizioni dovute alla diversa ottica dei due approcci e ai diversi costrutti supportati.

- Il passaggio è impossibile se l'AN presenta attività in AND, perchè il DST non supporta questo costrutto per sua natura. È infatti impensabile avere due attività contemporanee operanti sugli stessi dati: si avrebbe il problema della concorrenza che il DOM stesso è teso ad evitare col meccanismo dell'ownership. L'unica concorrenza ammessa può al limite presentarsi in lettura, ma questa non implica attività parallele. In più, si può notare come in genere attività parallele in un processo di workflow siano legate a dati diversi, quindi sta a noi trovare il percorso nell'AN che riguarda i nostri dati. Eventualmente può risultare necessario collegare ad un unico AN più DST, uno per ogni flusso di dati diverso (per ogni oggetto concettuale). In realtà si può proporre

l'idea di attività concorrenti sull'oggetto concettuale, ma solo se queste operano su partizioni diverse dello stesso. Anche la modellazione con DST può essere affrontata, ma risulta complessa e non conveniente.

- Il passaggio può avvenire solo se sappiamo risolvere i problemi pratici di rappresentazione che si presentano:
  - più flussi di dati necessitano molti DST ma un solo AN (come già detto sopra);
  - non sono definiti i nomi degli stati;
  - problema nei diagrammi con attività ripetute (vedi in seguito).

Elenchiamo ora le regole di passaggio che abbiamo elaborato nel caso in cui non ci siano problemi di attività parallele:

1. ogni attività escluse quelle di fine (se presenti), diviene una transizione con lo stesso nome.
2. una attività priva di frecce in ingresso diviene la *start transition*.
3. una attività che ha come unico successore una attività di fine diviene una transizione verso uno stato finale.
4. dopo una attività (transizione) x, ci si trova in uno stato (cui diamo un nome del tipo *Stato-n*, con n intero progressivo); le attività che seguono la x sono tutte le transizioni uscenti da quello stato. Se dopo due attività diverse x e y, si possono eseguire le stesse attività, allora x e y portano nello stesso stato.

Come esempio di passaggio nel caso più semplice, eseguiamo al contrario il mappaggio dell'esempio precedente.

- Abbiamo 8 attività, ma due sono di fine, quindi si avranno 6 transizioni.
- Abbiamo 2 attività di fine, quindi si avranno 2 stati finali distinti.
- Costruzione:
  - L'attività T-1 diviene la transizione iniziale. Come tale porta in un primo stato che indichiamo con S-1.
  - Dopo T-1 si possono eseguire in OR-Split T-2, T-3, T-4. Queste tre attività divengono tre transizioni in uscita da S-1.

- T-2: dopo T-2 si possono eseguire in OR-Split T-2, T-3, T-4. Dal confronto col punto precedente, quindi, T-2 porta in S-1, è cioè un autoanello.
- T-3: dopo T-3 si possono eseguire in OR-Split T-5 e T-6. Di conseguenza T-3 porta in un nuovo stato che chiamiamo S-2.
- T-4: dopo T-4 c'è l'attività di fine Fine-1. T-4 porta quindi in un nuovo stato che è finale e che chiamiamo S-3.
- T-5: dopo T-5 c'è l'attività di fine Fine-1. T-5 porta nello stesso stato finale cui portava T-4, cioè S-3.
- T-6: dopo T-6 c'è l'attività di fine Fine-2. T-6 porta in un nuovo stato che è finale e che chiamiamo S-4.

Non riportiamo i grafi relativi all'esempio perchè già rappresentati sopra nelle Fig 3.7 e 3.6.

3.3.2 Osservazioni e commenti

Come si può evincere dall'esempio, nel caso più semplice di AN, è facile ricostruire il corrispondente DST. Anche in caso di assenza di attività fittizie di fine (come accadrebbe in una modellazione fatta seguendo la Coalition), poi, è possibile procedere allo stesso modo, semplicemente inserendo uno stato finale diverso dopo ogni attività che non abbia attività successive.
Problemi maggiori si hanno con topologie più complesse dell'AN, ad esempio in caso di anelli nell'esecuzione di attività. Seguendo le regole sopra elencate, si avrebbero transizioni con gli stessi nomi che partono da stati diversi e portano nello stesso stato. Se graficamente ciò non crea complicazioni, può crearle sul piano logico e del significato del processo.
Ulteriore complicazione può aversi quando dopo un'attività si presenta un OR-Split fra un'attività "normale" ed una fittizia di fine. In questo caso, infatti, si avrebbe nel DST una transizione che esce da uno stato finale, costrutto da noi non ammesso. Per sopperire a tale problema, si può scegliere di ammettere l'uscita da uno stato finale, o inserire degli stati intermedi fittizi che mantengano il significato dell'AN senza snaturare il DST.

Un'ultima considerazione si può fare riguardo le citate attività parallele. Ammettendo di permetterle su porzioni distinte dell'oggetto concettuale (come segnalato sopra), si dovrebbero inserire nel DST dei costrutti nuovi. A seguito dell'AND-Split (il cui simbolo può essere mutuato dalla Coalition), infatti, l'oggetto concettuale si considera in uno stato *unione* fra gli stati presenti nei percorsi paralleli. Si inserisce poi uno stato aggiuntivo al termine di ogni ramo, prima del punto di sincronizzazione, che potremmo

chiamare *rilasciato-n*, dove n è il contatore dei rami. Solo quando lo stato *unione* è dato dall'unione di tutti gli stati *rilasciato-n* si può procedere con lo stato successivo all'AND-Join. È evidente, però, la complicazione introdotta nel modello da questo costrutto, che si ripercuote inevitabilmente anche sull'implementazione, cosicchè pare opportuno non seguire questa strada.

In generale, quindi, appare un po' forzato il tentativo di vedere il DOM come sistema di workflow, o per lo meno di metterlo al pari dei sistemi esistenti. È innegabile che esistano delle somiglianze e vari punti di contatto, ma le differenze rimangono, soprattutto a livello tecnico-implementativo. Al DOM, infatti, manca tutta quella struttura di supporto che rende un sistema di workflow capace di gestire i processi, manca cioè il "motore" di workflow, mancano le worklist, mancano strutture di processo più complesse. Con piccole modifiche, però, il DOM potrebbe essere trasformato in un sistema di workflow limitato e dedicato ad applicazioni orientate ai dati. Dalla sua, infatti, ha il vantaggio di partire già come sistema distribuito, mentre la gran parte dei sistemi di workflow esistenti non offrono questa potenzialità.

Una seconda strada da seguire, invece, può essere quella di integrare il DOM con un sistema di workflow, per mettere assieme gli aspetti positivi di entrambi superando i rispettivi limiti. Come vedremo nel Capitolo 5, è questa la strada che nella tesi si è pensato di seguire.

Capitolo 4

I sistemi utilizzati

In questa tesi si sono utilizzati due sistemi commerciali per la parte implementativa del lavoro: ActionWorkflowSystem della Action Technology Inc. e Microsoft SQL Server. In particolare, i due sistemi, il primo WFMS, il secondo DDBMS, sono quelli su cui si è basato il progetto descritto nel Capitolo 5 e la sua implementazione (Capitolo 6). Nel presente capitolo si vuole dare una descrizione delle caratteristiche salienti di questi due sistemi, per maggiore completezza e chiarezza di trattazione. In particolare, si farà riferimento a [18] per ActionWorkflow e a [11] per SQL Server.

4.1 Action Workflow System

ActionWorkflow è un sistema completo per sviluppare ed eseguire workflow. Come molti altri prodotti concorrenti, è basato sul paradigma *client-server* e non possiede le caratteristiche per essere un sistema distribuito. Nella versione a nostra disposizione (quella per sviluppatori), si presenta come pacchetto software composto da due componenti di base più una applicazione di esempio. I due componenti sono il **Process Builder** e il **Process Manager**, che rispondono alle esigenze rispettivamente di progettare e disegnare il modello di un processo aziendale il primo, di gestirne l'esecuzione il secondo. Poichè poi al Process Manager manca totalmente una interfaccia grafica che permetta l'interazione fra utenti e istanze del processo in esecuzione, viene fornita una applicazione di esempio, sviluppata in Visual Basic 4.0, da utilizzare come base per capire il funzionamento del programma e sviluppare nuove applicazioni.
ActionWorkflow, nella sua installazione completa, non è autosufficiente, in quanto necessita di un database come depositario di tutte le informazioni necessarie al suo funzionamento (più una sua copia come archivio). Questo

database non viene gestito direttamente dal programma, ma da un DBMS distribuito (indicato come *back-end server*). Dalla ditta produttrice sono state sviluppate versioni per Lotus Notes e Microsoft SQL Server (quella a nostra disposizione), ma ne è prevista anche una per Oracle. Sebbene il supporto del DDBMS permetta in linea teotica una gestione distribuita dei processi, ActionWorkflow non sfrutta queste potenzialità, ma lo utilizza come un "normale" DBMS.

4.1.1 Il Process Builder

Il primo passo nella realizzazione di un workflow per automatizzare un processo aziendale è l'analisi del processo stesso e la costruzione di un modello in un formato comprensibile all'elaboratore. Il Process Builder è la parte di ActionWorkflow dedicata a questa importante fase del lavoro. È un tool grafico che permette infatti di disegnare la *mappa* del processo e di definire poi i ruoli coinvolti e i dati su cui il processo opera.

L'approccio

Ciò che caratterizza il prodotto e lo distingue dalla gran parte dei suoi concorrenti (e lo allontana dai dettami della W*f*MC), è l'approccio utilizzato nella progettazione del workflow (vedi per un'introduzione ad esso il capitolo 2). Non troviamo infatti nessun riferimento ad *attività* o *task*, mentre l'unico termine utilizzato è quello di *workflow*. Il processo è visto come un insieme di workflow, uno principale, gli altri secondari (sottoprocessi). Ogni singolo workflow coinvolge sempre due attori, un *customer* e un *performer*, evidenziando il fatto che ogni processo è visto come una *conversazione* fra chi richiede un'azione o un servizio, e chi lo effettua. Ogni workflow è poi diviso in quattro fasi: preparazione, negoziazione, performance e soddisfazione (come si può vedere nella fig 4.1). La conversazione può essere affrontata da due punti di vista, richiesta e offerta, con riferimento a chi fra performer e customer è parte attiva in essa, ed ha un obiettivo: il performer deve soddisfare il customer entro un ben definito periodo di tempo. Questo approccio enfatizza il consenso e la responsabilità: entrambe le parti devono concordare sul da farsi. In particolare, il performer è responsabile per ciò che ha accettato di fare; mentre il customer è responsabile di controllare se quanto stabilito è stato fatto correttamente.
Essendo basato su una metodologia così ben precisata, ActionWorkflow è stato dotato di una serie di regole che ogni processo deve seguire per poter soddisfare i tre principi seguenti:

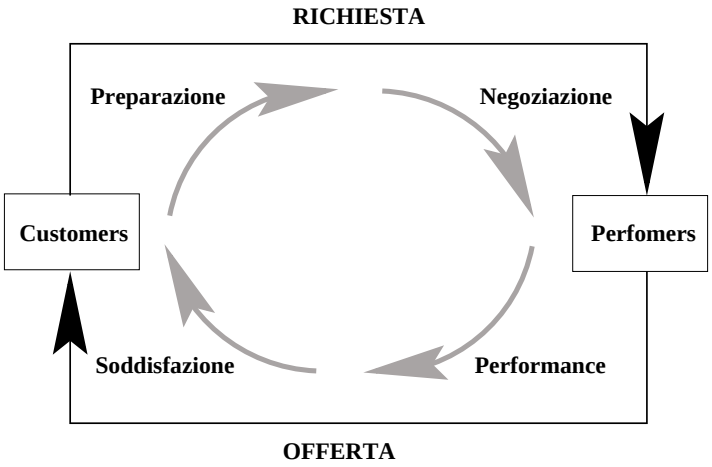


Figura 4.1: La conversazione o workflow loop in ActionWorkflow

- *consistenza logica:* nessuna definizione di un elemento del processo deve essere in contrasto con gli effetti di un'altra definizione;
- *semplicità:* i processi non devono essere inutilmente complessi e gli elementi ridondanti vanno eliminati;
- *completezza:* tutti gli elementi essenziali vanno inclusi.

Il ciclo di workflow

Un ciclo di workflow è la rappresentazione grafica di una conversazione. In esso viene definito chi può parteciparvi, che azioni (o *atti*) può intraprendere (fra un insieme predefinito), su che dati lavora e quanto tempo può durare l'intero ciclo. Nell'applicazione finale questo si traduce in form che presentano i dati corretti all'utente corretto al momento giusto, e in eventuali richiami all'utente stesso per il rispetto dei limiti di tempo.

I partecipanti Agli utenti finali, indicati nel programma come *identità*, vengono assegnati dei *ruoli* che permettono loro di partecipare con determinati doveri e diritti al processo. I ruoli possono essere funzioni temporanee rilevanti solo per un particolare workflow, oppure permanenti all'interno della struttura organizzativa dell'azienda. Non c'è limite al numero di ruoli che un'identità può impersonare e neppure al numero di identità che possono impersonare uno stesso ruolo. Per ogni ciclo di workflow, inoltre, una delle identità può essere definita come ruolo di "default", ossia come destinatario preferibile per l'esecuzione di determinati atti all'interno delle istanze del

processo. Notiamo che le identità non vengono definite, per maggiore generalità, in questa fase, ma solo in seguito, quando si vuole effettivamente eseguire il processo.
I cicli di workflow devono necessariamente includere un *customer* ed un *performer*. Possono anche includere un osservatore (*observer*), che fornisce dei commenti ma che non può partecipare attivamente al ciclo. I customer e i performer abilitati a dare il via al processo sono detti *iniziatori*.
ActionWorkflow System permette la modifica "al volo" dei legami ruolo-identità e ruolo-partecipante. Tuttavia questa flessibilità è limitata dal piccolo numero di attributi delle identità: nome, e-mail, indirizzo di posta.

Atti e stati Per eseguire una conversazione, i partecipanti scelgono fra un numero limitato di azioni o *atti*. Questa scelta modifica lo stato della conversazione. Ci sono nove possibili stati raggiungibili mediante due serie di sedici atti.

- **Atti principali:** attiva, inizia, richiedi, offri, commenta, accetta l'offerta, fai una contro-proposta, accetta la contro-proposta, rifiuta l'offerta o la contro-proposta, notifica il completamento, dichiara soddisfazione, rifiuta di accettare, cancella.
- **Stati principali:** preparazione, negoziazione, accettazione, performance, rifiutato, soddisfatto, cancellato.

ActionWorkflow System fornisce due serie di atti di default, chiamati *template*: una per le conversazioni iniziate dai customer (template di richiesta), una per quelle iniziate dai performer (template di offerta). Il progettista copia il template rilevante, poi sceglie gli atti da eseguire in base al processo da sviluppare ed eventualmente li rinomina per una migliore comprensione da parte degli utenti finali.

Campi dei dati e delle maschere ActionWorkflow System distingue fra dati locali e globali, ma non separa esplicitamente i dati rilevanti per il processo da quelli relativi ai vari task. Tutti vengono indistintamente gestiti dalle maschere di front-end (non fornite dal sistema, ma a carico del progettista). I dati globali vengono definiti a livello di processo per essere condivisi da tutti i cicli di workflow; i dati locali, invece, sono rilevanti solo per il ciclo di workflow cui vengono associati. Questi dati possono essere presentati ai partecipanti in quattro modalità (read-only, modificabili, da inserire, nascosti), a seconda dell'identità degli stessi, del loro ruolo all'interno del workflow, dello stato in cui cui il workflow si trova attualmente. Le maschere possono

anche mostrare, in modalità read-only, dati locali associati a differenti cicli di workflow per fornire agli utenti una visione più completa dello svolgimento del processo. La scelta di quali dati mostrare all'utente, comunque, dipende dalle applicazioni che verranno realizzate per gestire l'interfaccia coi processi.

Tempo di ciclo e deadline Chi disegna il processo può specificare quanto ogni fase deve/può durare, e inserire notifiche automatiche, dopo la scadenza dei termini previsti, agli utenti interessati. Quando un processo è completato, ActionWorkflow calcola due totali:

- il *computed process-cycle time*, che è il tempo per ogni singolo ciclo di workflow
- il *business process-cycle time*, che fornisce il tempo totale per l'esecuzione di tutti i cicli.

Questi due totali sono identici nel workflow principale che non può essere completato finchè non sono terminati tutti i workflow subordinati. Questi dati sono importanti per un controllo della performance, in quanto possono essere confrontati con i dati corrispondenti stimati dai modellatori del processo.

La business process map

I processi di ActionWorkflow sono chiamati *business process map*. Si compongono di una gerarchia di cicli di workflow, dei quali il principale è detto *workflow primario*, gli altri, sottoprocessi, sono chiamati *workflow figli*. I workflow primari possono avere un numero qualsiasi di "figli", collegati ad essi da link. Ogni link è abilitato da un atto nel workflow primario o abilita atti nel workflow figlio. È importante notare il particolare significato del workflow primario all'interno di una mappa. Esso rappresenta infatti l'intero processo aziendale, e può terminare solo quando i processi figli (che sono sue specificazioni), sono terminati. Dal workflow primario, quindi, si può (e si deve), avere una visione globale del processo aziendale, visione che nei sottoprocessi viene ampliata e particolareggiata. Questo approccio comporta per il progettista un grosso sforzo di modellazione, in quanto egli deve essere in grado di identificare il ciclo principale di un processo aziendale e scendere poi sempre più a fondo nella sua struttura per evidenziare le attività che lo compongono. Come già accennato sopra, inoltre, le mappe devono soddisfare alcune regole di consistenza elencate sui manuali. Nel controllo di questa consistenza è fondamentale uno strumento omonimo presente nel Process Builder e capace di segnalare e identificare gli errori di modellazione

commessi. Solo dopo aver passato il controllo di consistenza una mappa è pronta per essere passata al motore di workflow.

Script ActionWorkflow System è fornito di un linguaggio (ActionWorkflow Language), formato da una serie di statement che possono riferirsi a workflow, stati, atti e dati. Tramite operatori aritmetici e relazionali, combinati in espressioni logiche, i progettisti possono definire particolari comportamenti del processo, controllare condizioni, chiamare altri programmi già a build-time nel Process Builder. Questi script verranno eseguiti quando un partecipante farà un particolare atto o quando il processo entrerà in un determinato stato durante l'esecuzione. Per poter sfruttare appieno queste potenzialità, però, è necessaria una profonda conoscenza della logica di processo di ActionWorkflow e una buona conoscenza del linguaggio di scripting (che assomiglia al Visual Basic). Non stupisce quindi che nel kit per sviluppatori (a nostra disposizione), sia stata inserita una applicazione di esempio che può essere analizzata in ogni sua parte per prendere confidenza col sistema, dalle mappe all'esecuzione vera e propria delle istanze.

4.1.2 Componenti Funzionali

Nella Fig. 4.2 sono mostrati i comonenti funzionali di ActionWorkflow System. Dalla parte del server, ActionWorkflow System consiste di un motore di workflow, il Process Manager, due file di rete e cinque tabelle di database (che nella installazione standard è chiamato *aws*). Il Process Manager utilizza le tabelle delle transazioni, dei nomi e delle definizioni dei processi, per determinare quali atti siano stati completati, quali possano essere intrapresi, e per gestire le istanze di processo. Le definizioni dei processi sono conservate in un direttorio condiviso, detto Map Directory. L'altro file di rete, il "message log", mantiene memoria di ogni istanza eseguita. Il Process Manager consiste di quattro processi:

- una interfaccia utente, l'*Administrator*, attraverso cui l'amministratore di sistema accede alle tre tabelle citate ed al file di log;
- un *Transaction Manager* che modifica la tabella delle transazioni in accordo all'esecuzione delle istanze, mantenendo l'integrità delle stesse;
- uno *Scheduler* che gestisce la tabella di schedulazione per definire quando e quanto spesso il Process Manager deve far partire un processo;
- un *Follow-up Manager* che interroga la tabella delle transazioni per determinare se e quando un partecipante deve essere avvertito di un ritardo nell'esecuzione dei suoi compiti.

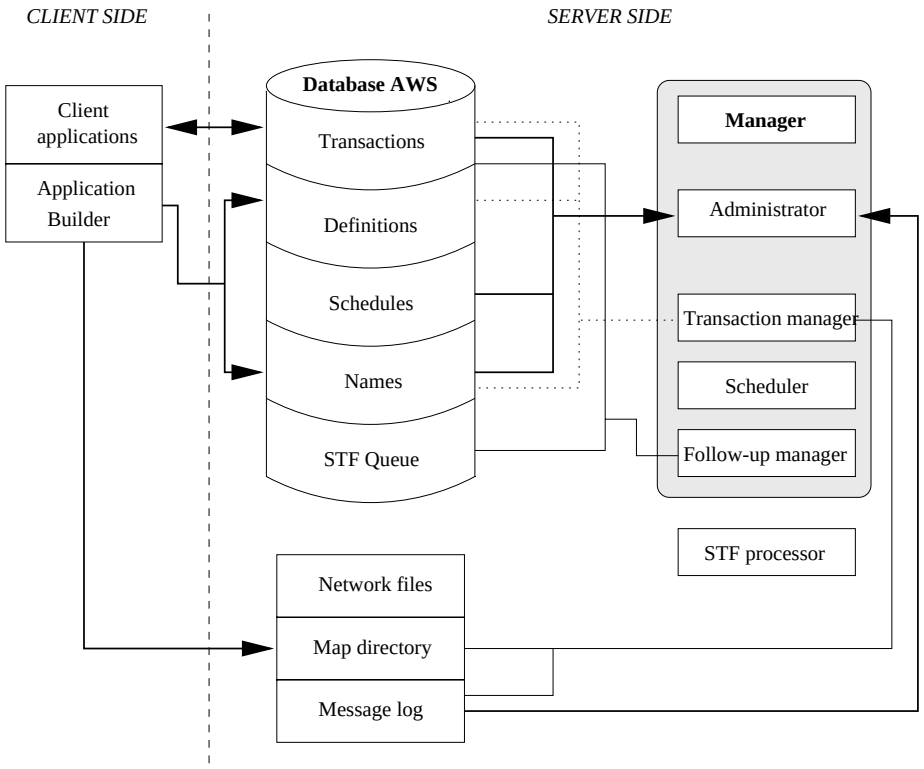


Figura 4.2: I componenti funzionali di ActionWorkflow

Il Follow-up Manager memorizza i messaggi nella quinta tabella (la coda STF), gestita dall'*STF (Standard Transaction Format) Processor*, che si connette ad ActionWorkflow System tramite il sistema di posta elettronica dell'utente. Il processore STF interroga la tabella ad intervalli regolari (definiti dall'amministratore), crea la corretta intestazione per il messaggio e lo spedisce. Dalla parte del *client*, le applicazioni accedono la tabella delle transazioni, mentre l'Application Builder (il tool di design di ActionWorkflow System), gestisce le tabelle delle definizioni e dei nomi e il direttorio delle mappe.

È importante notare qui che ActionWorkflow System fornisce all'utente anche un set di API (nelle versioni a 16 o 32 bit), che forniscono l'interfaccia fra le applicazioni client e il sistema stesso. Sono descritte ampiamente nei manuali, ed è possibile utilizzarle nello sviluppo di applicazioni previa installazione nel sistema. Rappresentano quelle WAPI's di cui si è parlato nel capitolo 1, ma, ovviamente, queste sono proprietarie di ActionWorkflow System e dedicate solamente ad esso.

Per quanto riguarda il database *aws*, possiamo dire che esso è la "batteria" del sistema di cui il Process Manager è il "motore". Esso, infatti, serve co-

me "accumulatore" di dati, ma anche come "fornitore" degli stessi. In esso vengono memorizzate, all'inizio, tutte le informazioni necessarie a ricostruire la definizione di processo (la mappa generata dal Process Builder più il mappaggio ruoli-identità). Queste informazioni vengono poi lette dal Process Manager durante l'esecuzione delle istanze e in base ad esse vengono aggiornate le tabelle del database dedicate al mantenimento dello stato delle stesse. Sul piano pratico, il database è formato da più di 100 tabelle sulle quali i vincoli di integrità sono mantenuti tramite indici *unique* (per le chiavi primarie), e tramite trigger (per le chiavi esterne). Queste caratteristiche sono molto importanti nell'ambito della presente tesi, per le implicazioni che avranno sul progetto; quindi ad esse faremo riferimento ancora in seguito.

ActionWorkflow Administrator

È l'interfaccia verso il motore di workflow del sistema, residente di solito sul server NT su cui è installato anche SQL Server. Si lancia dal menù di avvio del computer e si presenta all'utente con un'interfaccia scarna ed essenziale che permette solo poche operazioni di gestione. Innanzi tutto, per poter lavorare è necessario connettersi all'SQL Server su cui è installato il database di ActionWorkflow (e il processo *aws_srvn*), poi lanciare manualmente il Process Manager. Da notare che se più installazioni di ActionWorkflow System sono fatte su diversi server di una rete, è possibile scegliere a quale server connettersi, e quindi con quali processi lavorare. Ottenuta la connessione e fatto partire il manager, è possibile inserire nel sistema nuovi account (identità), installare o modificare definizioni di processo generate con il Process Builder, far partire, schedulare, fermare istanze dei processi installati. Non risulta però possibile, come già accennato sopra, interagire attivamente con le istanze di processo come normali utenti del sistema di workflow; per partecipare ai processi, infatti, è necessario essere forniti di un'adeguata interfaccia utente a form, che il sistema non ha. Lo sviluppatore, quindi, si deve incaricare non solo della modellazione del processo secondo le complesse regole di ActionWorkflow per la generazione delle mappe, ma anche della creazione delle applicazioni utente che utilizzino il sistema. Questa limitazione (che nella versione per Lotus Notes non è presente in quanto ActionWorkflow sfrutta le form di questo sistema), garantisce certo ampia libertà ai programmatori e rende il prodotto espandibile e "aperto", ma li costringe anche ad un lavoro non immediato e semplice prima di poter anche solo provare un processo reale. Fortunatamente, a corredo del sistema è fornita una applicazione di esempio che permette al programmatore di entrare più facilmente nella logica di ActionWorkflow System. Vedremo nella sezione seguente di cosa si tratta.

4.1.3 La Call Tracking Application

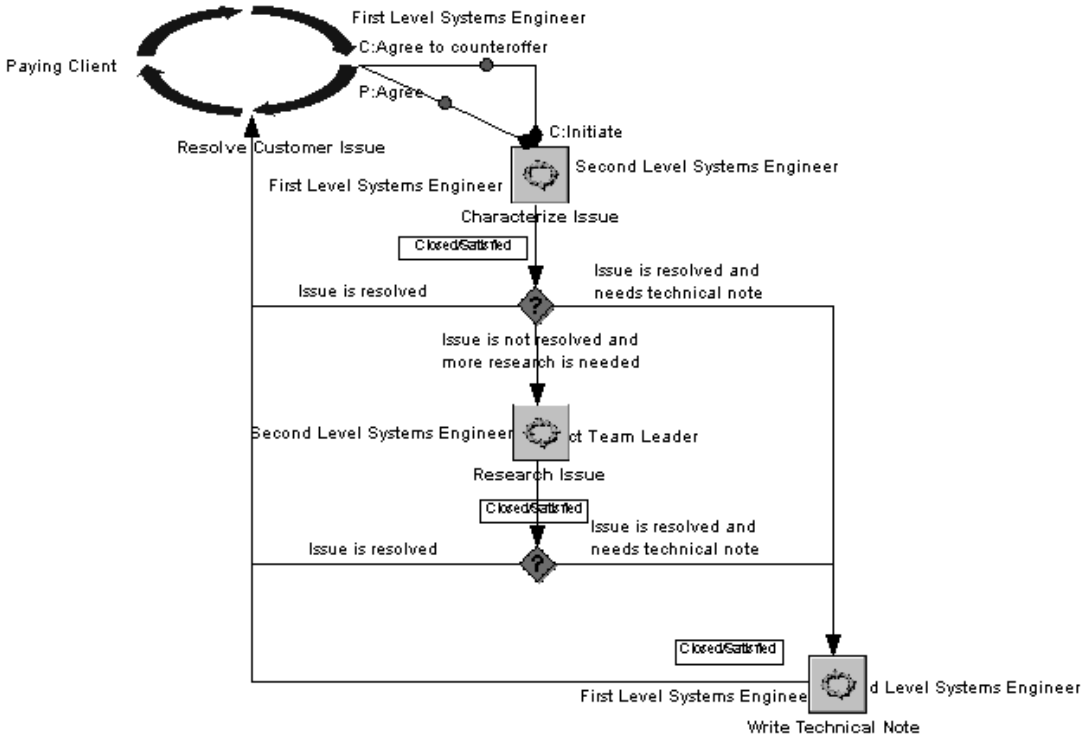


Figura 4.3: La "mappa" della Call Tracking Application

La Call Tracking Application è una applicazione scritta in Visual Basic 4.0 (ma si possono utilizzare per la programmazione delle form altri linguaggi quali il Visual C++ o Power Builder), che esemplifica il "cosa" ed il "come" di ActionWorkflow System, e si pone come punto di partenza necessario, ma valido, per lo sviluppo delle proprie successive applicazioni client per ActionWorkflow. Con il file eseguibile, infatti, vengono forniti anche tutti i sorgenti, da studiare, copiare, modificare, o in alcuni casi, riusare subito (ad esempio la libreria di funzioni base per l'interazione col sistema è già pronta per l'uso in nuove applicazioni). Inoltre, gli stessi manuali di ActionWorkflow System fanno frequente riferimento a quest'esempio, che viene spiegato e commentato in ogni sua parte.
L'applicazione riguarda la procedura aziendale di analisi e correzione di problemi nel software prodotto dalla ditta stessa, in seguito ad una richiesta di un cliente. La mappa (in Fig 4.3), si articola in diversi cicli di workflow che vengono attivati o meno a seconda che il problema nel software sia risolto subito o richieda invece un intervento più approfondito. Una volta prepa-

rata la mappa (cosa già fatta in questo caso dagli sviluppatori della Action Technology), bisogna "generarla" dal Process Builder scrivendo un file in una particolare forma interpretabile dal Process Manager (tale file ha estensione ".awo", cioè ActionWorkflow Object). Questa operazione (che include anche il controllo di consistenza di cui si è detto in precedenza) chiude la sessione di lavoro sul Process Builder. A questo punto, dal Process Manager, si "installa" la definizione di processo, si aggiungono le "identità" necessarie, e si effettua il mappaggio fra queste ed i ruoli organizzativi preventivamente aggiunti. Solo ora è possibile dare il via al processo di workflow vero e proprio utilizzando l'eseguibile dell'applicazione.
L'applicazione si presenta con una finestra principale dalla quale è possibile effettuare il login al sistema con una delle identità specificate precedentemente. Una volta entrati, si possono poi aprire delle sotto-finestre che mostrano rispettivamente le istanze di workflow attive nel sistema, le worklist degli utenti e, per ogni azione possibile, le eventuali scelte alternative. In particolare, la finestra che tratta delle worklist mostra non solo gli atti che l'utente connesso deve eseguire come performer (nella "pending by me" list), ma anche quegli atti che egli ha richiesto come customer ad altri (nella "pending to me" list).
Il sistema si occupa di controllare che gli accessi siano corretti (i login con username e password sono definiti dall'Administrator tramite la sicurezza di SQL Server) e che gli atti siano intrapresi dagli utenti che ne hanno i diritti secondo il precedente mappaggio identità-ruolo. Ogni tentativo di eseguire atti non permessi viene quindi segnalato da un messaggio di errore.
Nell'ottica del sistema di workflow, dunque, ogni partecipante al processo si connette al sistema ed esegue le operazioni dovute tramite l'interfaccia dell'applicazione; quando il processo è terminato ("completed"), l'istanza viene archiviata nell'apposito database (chiamato *awsarch*). A questo punto su di essa non si può più operare, ma è possibile rivederne l'esecuzione per identificare l'origine di eventuali problemi o anche solo come controllo di efficienza del processo stesso.
Dal punto di vista strettamente tecnico, l'applicazione di esempio è ridotta all'essenziale, con un 'interfaccia semplice (ma intuitiva), e nessuna aggiunta rispetto alle funzioni di base. In realtà, comunque, questo è un fatto positivo, soprattutto per chi si deve cimentare nella comprensione del suo funzionamento interno (studiando i file sorgenti), per poi ricostruire sull'esempio nuove applicazioni. Di grande utilità è il file che contiene, già implementate tramite le API di ActionWorkflow System, tutte le funzioni citate sopra: dal recupero delle worklist, all'esecuzione di un atto, al login nel sistema. Per i futuri sviluppatori, quindi, sarà sufficiente includere il file nel progetto Visual Basic per poter utilizzare, con semplici chiamate a procedura, tutte le

funzioni necessarie.

4.1.4 Osservazioni e commenti

Il sistema che ho avuto modo di provare ed utilizzare per un buon periodo di tempo, ha mostrato aspetti sia positivi che negativi, un po' su tutti i fronti.

ActionWorkflow è certamente un sistema di workflow completo, spaziando dal design dei processi aziendali fino alla gestione dell'esecuzione degli stessi. In più, si propone come sistema aperto e altamente programmabile, soprattutto per quanto riguarda l'interfaccia utente. Tutto ciò, però, comporta una difficoltà oggettiva nell'interagire intuitivamente con un sistema con queste caratteristiche. Fin dalla modellazione dei processi, infatti, l'utente è costretto a scontrarsi con un tipo di approccio particolare e, soprattutto, considerevolmente diverso dalla gran parte delle proposte presenti sul mercato (che in generale seguono le linee guida tracciate dalla Coalition). Superato questo primo ostacolo, poi, non c'è modo di realizzare un proprio progetto di workflow senza prima aver predisposto (programmato), una applicazione che realizzi l'interfaccia. In questo la Call Tracking Application è certo un buon aiuto, ma non toglie lo sviluppatore dall'impaccio di programmare anche solo per realizzare un piccolo prototipo.

Il problema più grosso che ho riscontrato nel sistema, però, è la sua fragilità, che si traduce in frequenti malfunzionamenti. Questi si non si sono mai presentati lavorando con il Process Builder, mentre il Process Manager ne è stato soggetto più volte. In particolare, i problemi più comuni si sono avuti nel far partire il processo del Process Manager dall'Administrator e nell'installazione di nuove definizioni di processo dopo che era avvenuta la cancellazione di una non più utilizzata. Per risolvere i problemi le uniche soluzioni erano far ripartire la macchina, o se non bastava, reinstallare (purtroppo non solo una volta), il pacchetto. In complesso, quindi, l'approccio al prodotto, già non semplice per le caratteristiche del sistema, è stato rallentato e sfavorito dalla sua "debolezza", e l'impressione avuta non è delle più positive.

4.2 Microsoft SQL Server 6.5

Microsoft SQL Server 6.5 è il DDBMS commerciale utilizzato nella presente tesi. In questo sezione si vuole descrivere in particolare il suo sistema di replica dei dati e alcune altre sue caratteristiche ad esso legate.

4.2.1 Il sistema di replica di SQL Server 6.5

SQL Server 6.5 Replication è basato sulla modalità Loose Consistency e sulla metafora Publishing / Subscribing (vedi Appendice B). Ciò significa che il flusso dei dati è unidirezionale, dal publisher ai subscriber, ed in generale le applicazioni considerano i dati replicati ai subscriber come read-only (anche se in realtà sui dati replicati non abbiamo vincoli di update per permettere al subscription server di rendere persistenti le variazioni giunte dal publisher).
Il processo di replica in SQL Server 6.5 avviene tramite due operazioni distinte e sequenziali:

1. Processo di Sincronizzazione

2. Processo di Replica

Durante il processo di sincronizzazione, che viene eseguito una sola volta durante la configurazione dell'applicazione, l'obiettivo è di inviare a tutti i subscriber una copia completa dei dati (e dello schema) presenti nel publication server.
Terminata questa prima parte, che è obbligatoria, può iniziare il processo di replica dei dati che avviene ad intervalli di tempo definiti secondo le due diverse modalità:

- Transaction Based: vengono memorizzate tutte le transazioni inerenti la pubblicazione in esame che verranno spedite (al tempo opportuno) ai subscriber per la riesecuzione remota.

- Scheduled Refresh: al tempo opportuno (definito da utente) tutti i dati della pubblicazione (l'ultimo aggiornamento) vengono inviati ai server remoti.

Componenti del sistema di replica

I principali componenti, illustrati nella Fig 4.4, sono:

- Synchronization Process Prepara i file di sincronizzazione iniziale contenenti sia lo schema (intensione) che i dati (estensione) delle tabelle da replicare; memorizza i file nella directory di lavoro di SQL Server sul distribution database e registra i job di sincronizzazione nel distribution database. Il processo di sincronizzazione ha effetto solo sui nuovi subscriber.

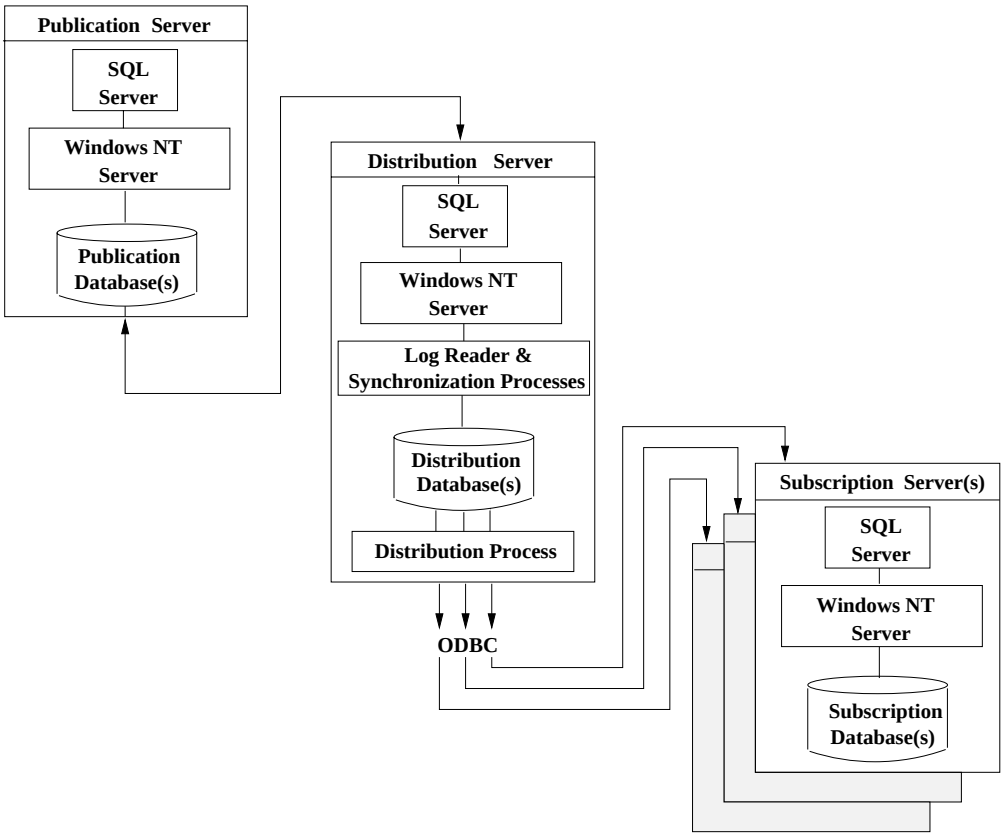


Figura 4.4: I componenti del sistema di replica di SQL Server

- Distribution Database E' uno "store-and-forward" database che mantiene traccia di tutte le transazioni che sono in attesa della distribuzione ai subscriber. Il Distribution Database riceve le transazioni speditegli dal publisher tramite il log reader process e le mantiene sino a quando il processo di distribuzione le porta ai subscriber. Il Distribution Database viene utilizzato solo per i processi di replica e non contiene dati e tabelle degli utenti.

- Log Reader Process Il Log Reader Process è il processo incaricato all'invio delle transazioni marcate dal Transaction Log del publisher come coinvolte al processo di replica, al Distribution Database, dove le transazioni sono poste in attesa per la distribuzione ai subscriber.

- Distribution Process E' il processo che porta le transazioni ed i job di sincronizzazione iniziale mantenute nelle tabelle del Distribution Database ai subscriber (in particolare nelle tabelle di destinazione dei Database ai siti di replica).

Tutti e tre i processi (Log Reader Process, Synchronization Process, Distribution Process) sono presenti nel distribution server (come sottosistema di SQL Executive) e possono essere utilizzati solamente dal system administrator.

Ruoli dei server nella replica

Il server può assumere tre ruoli differenti nel processo di replica di SQL Server.

- Publication Server (Publisher) E' il server che rende disponibili i dati per la replica. Il publication server gestisce il publication database, genera i dati da pubblicare estraendoli dalle tabelle opportune, invia copia di tutti i cambiamenti avvenuti al distribution server. Viene lasciata ampia libertà sulla configurazione dei dati pubblicati:

  - Pubblicazioni sicure: ciascuna pubblicazione ha uno stato di sicurezza marcato tipo unrestricted (default) oppure restricted. Una pubblicazione di tipo unrestricted è visibile e può essere sottoscritta da tutti, mentre una pubblicazione restricted limita la possibilità di sottoscrizione ad un elenco di server specificato.
  - Pubblicazioni di tabelle partizionate verticalmente: possiamo creare articoli di replica costituiti da alcune colonne di una tabella (frammentazione verticale, vedi Fig 4.5).

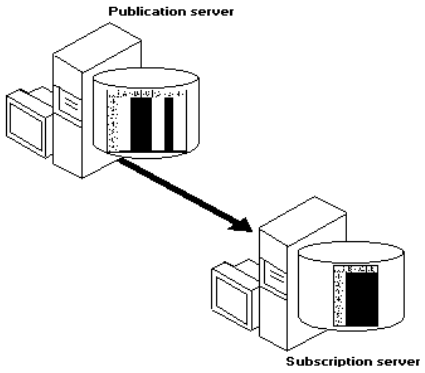


Figura 4.5: Partizione verticale

– Pubblicazioni di tabelle partizionate orizzontalmente: possiamo creare articoli di replica costituiti da alcune righe di una tabella (frammentazione orizzontale, vedi Fig 4.6).

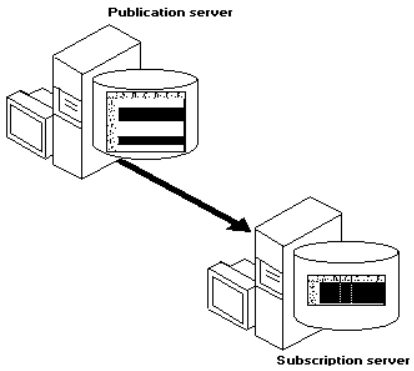


Figura 4.6: Partizione orizzontale

– Pubblicazioni di tabelle partizionate in modo mixed: possiamo partizionare una tabella in modo verticale e orizzontale contemporaneamente (vedi Fig 4.7).

- Distribution Server (Distributor) E' il server che contiene il distribution database: riceve tutte le transazioni relative ai dati pubblicati, le memorizza nel suo distribution database e, al momento opportuno, dipendentemente dal tipo di replica definito, trasmette ai subscription server le transazioni o i dati relativi alla pubblicazione. Può risiedere sia nella stessa macchina del publisher che in una separata.

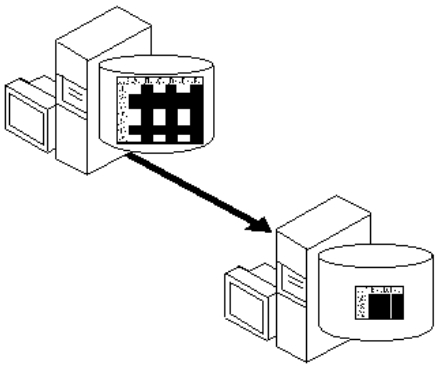


Figura 4.7: Partizione mista

- Subscriber Server (Subscriber) E' il server che riceve e gestisce i dati replicati. Il progettista può configurare in modo flessibile i dati da sottoscrivere:

  - Sottoscrizione selettiva di pubblicazioni: un subscription server può decidere di sottoscrivere nessuna, alcune o tutte le pubblicazioni di un publisher;
  - Sottoscrizione selettiva di articoli: un subscription server può decidere di sottoscrivere alcuni o tutti gli articoli di una pubblicazione.

Molto spesso i ruoli di publisher e di distributor coesistono nella stessa macchina. Inoltre il ruolo di publisher e subscriber non sono esclusivi ed un singolo server può eseguire entrambi (naturalmente riferendosi a pubblicazioni diverse).

Processo di sincronizzazione

Il processo di sincronizzazione iniziale viene eseguito in fase di configurazione dell'applicazione ed assicura che lo schema della tabella ed i dati del database sul sito di pubblicazione e sul sito di replica siano identici: terminato il processo di sincronizzazione, i subscriber sono pronti a ricevere gli aggiornamenti dei dati modificati tramite il processo di replica.
Infatti, quando viene creata una pubblicazione, una copia dello schema (compresi gli eventuali indici) e dei dati viene memorizzata su file nella directory di lavoro di SQL Server (nei file .SCH e .TMP rispettivamente). Questi file rappresentano l'insieme di sincronizzazione e vengono creati per ciascun

articolo della pubblicazione. Dopo aver creato l'insieme dei file di sincronizzazione, il processo di sincronizzazione può portare una copia dei file a ciascun Subscriber che, in questo modo, dispone di una copia esatta dei dati e dello schema del Publisher.
Non appena il Publisher ha generato l'insieme di sincronizzazione, tutti gli aggiornamenti, inserimenti e cancellazioni dei dati pubblicati sono memorizzati dai processi di Log descritti, ma potranno essere ricevuti da ciascun Subscriber per aggiornare i dati di replica solamente quando sarà terminato il proprio processo di sincronizzazione iniziale. In questo modo SQL Server garantisce che le modifiche ai dati di replica siano portate al Subscriber quando quest'ultimo possiede una copia esatta dei dati attuali nel Publisher. Inoltre, quando durante la sincronizzazione l'insieme dei dati viene distribuito, solamente quei Subscriber che sono in attesa della sincronizzazione iniziale ricevono i dati; gli altri Subscriber, che hanno già terminato la sincronizzazione oppure hanno ricevuto le ultime modifiche dei dati pubblicati non compiono il processo di sincronizzazione. Poiché è SQL Server che gestisce automaticamente la coda dei Subscriber in attesa, il carico di lavoro dovuto alla sincronizzazione viene ridotto.
La sincronizzazione iniziale può essere compiuta sia in modo automatico che manuale: in particolare si può fissare l'intervallo di tempo tra due sincronizzazioni successive delle nuove pubblicazioni. Tutti gli elementi di una pubblicazione (gli articoli) vengono sincronizzati simultaneamente, in modo da preservare l'integrità referenziale dei dati.

Automatic Synchronization La sincronizzazione automatica viene eseguita da SQL Server: per fare questo viene mantenuta su file una copia (snapshot) della tabella e dei dati degli articoli da pubblicare. Il processo di sincronizzazione crea un job di sincronizzazione che viene posto nel Distribution Database; quando il job viene attivato, invia i file di sincronizzazione a tutti i subscription database che sono in attesa di sincronizzazione. Quindi il job di sincronizzazione viene inviato dal Distribution Database come se fosse un qualsiasi altro job di aggiornamento.
Come detto, quando il processo di sincronizzazione iniziale viene inviato ai siti remoti, solo i subscriber che sono in attesa partecipano alla sincronizzazione, gli altri restano indifferenti. Questo permette di ridurre i carichi di lavoro.

Manual Synchronization La sincronizzazione manuale è eseguita dall'utente. Come per la sincronizzazione automatica, il publisher genera i file di sincronizzazione contenenti gli snapshots dello schema e dei dati; nel processo manuale l' utente ottiene una copia su nastro dei file e si incarica di inserirli

nei subscription server. Al termine del processo di sincronizzazione manuale l' utente comunica a SQL Server che il processo è terminato correttamente e si può procedere con la fase di replica (analogamente a quello che faceva direttamente il sistema nel processo automatico).
La sincronizzazione manuale è particolarmente utile quando la rete che collega publisher e subscriber è lenta, costosa oppure la quantità dei dati da copiare è molto elevata.

No Synchronization Quando si dichiara una subscription, possiamo specificare di non volere la sincronizzazione di un particolare articolo. In questo caso SQL Server assume che la sincronizzazione sia sempre terminata, in modo che ogni variazione dei dati sul publisher viene comunicata ai subscriber durante il primo processo di replica successivo alla definizione della subscription (senza aspettare il completamento della sincronizzazione).
Nel caso di no synchronization deve essere il progettista della pubblicazione a farsi carico della consistenza dei dati replicati in fase iniziale. Il vantaggio di questa modalità risiede nella maggiore agilità del processo di copia dei dati replicati, in quanto non compare il sovraccarico dovuto alla sincronizzazione.

Snapshot Only Quando si definisce una subscription, esiste l'opzione Snapshot Only per la quale SQL Server sincronizza gli articoli pubblicati con le tabelle di destinazione e ripeterà l'operazione ad intervalli definiti nel tempo (a scadenza giornaliera, mensile, ...). In questo modo le modifiche sui dati del publisher non vengono inviate alle repliche, che vengono aggiornate solo durante la successiva operazione di sincronizzazione. La metodologia è stata introdotta come Replica Scheduled Refresh.

Processo di replica delle pubblicazioni

Come detto, la replica consiste nell'allineamento delle copie dei dati presenti ai siti remoti con l'ultima versione aggiornata presente nel publication server. Le modalità previste sono due: Transaction Based e Scheduled Refresh.

Transaction Based Per le pubblicazioni definite Transaction Based l'obiettivo è quello di trasmettere ai siti remoti la sequenza corretta delle transazioni avvenute sui dati di una pubblicazione. Quando giungono ai siti remoti, tali transazioni vengono eseguite e, poiché si parte da una situazione di dati sincronizzati, il risultato finale è quello di avere in ciascun sito una copia dei dati presenti nel publisher.
La determinazione dell'insieme delle transazioni spedite durante il processo di replica è basato sui Log file: esiste un processo, chiamato Log Reader

Process, che compie il monitoraggio dei log delle transazioni dei database abilitati alla pubblicazione. Quando una transazione viene compiuta su di una tabella di tipo published, tale transazione viene marcata per la replica ed inviata (dal Log Reader Process) al distribution database. Le transazioni vengono qui mantenute in attesa di poter essere inviate ai subscriber per l' aggiornamento dei dati di replica.
Naturalmente solo le transazioni che hanno terminato con il commit sono spedite ai server di replica; inoltre, poiché l'invio delle transazioni è basato sul log, siamo sicuri che le transazioni sono spedite dal distribution database e ricevute dal subscriber nello stesso ordine in cui vengono eseguite dal publisher e quindi portano le varie copie dei dati nello stesso stato dell' originale.

Scheduled Refresh Per le pubblicazioni definite Scheduled Refresh non interessa conoscere le transazioni che modificano i dati poiché vengono trasmessi ai subscriber tutti i dati che appartengono alla pubblicazione. L'operazione viene eseguita ad intervalli prefissati dall'utente ed equivale, sostanzialmente, ad una sincronizzazione in cui vengono inviati solo i dati e non lo schema delle tabelle.

Modalità di subscription: PUSH e PULL

Per poter ricevere la copia dei dati, un server deve definire una subscription verso le pubblicazioni di interesse: esistono due modalità di abbonamento (tipo push e pull) a seconda che l'attenzione sia rivolta al publication server o al subscription server.

Push Subscription Una subscription di tipo push viene utilizzata quando l'interesse dell' applicazione è posto sul publication server. La copia dei dati di una applicazione avviene attraverso l'invio contemporaneo dei dati stessi da parte del publisher a tutti i subscriber presenti.
Il principale vantaggio dell' utilizzo del "push" è la semplificazione e la centralizzazione delle procedure di replica, non dovendo gestire separatamente ogni sito di replica.

Pull Subscription Una subscription di tipo pull viene utilizzata quando l'interesse dell' applicazione è posto sul subscription server. La replica è basata sulla richiesta da parte di un subscription server ad un publisher per l'invio della replica di una pubblicazione.
Il vantaggio risiede nella maggiore autonomia di un subscriber nei confronti delle operazioni di replica: ogni sito di replica può decidere quando e quali

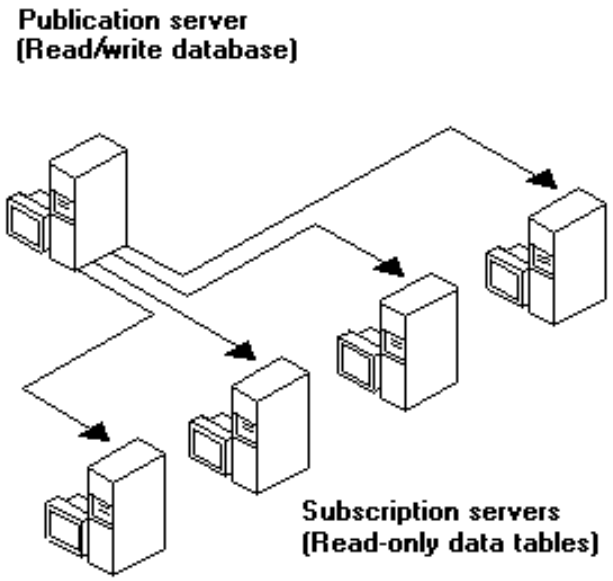


Figura 4.8: Tipologia central publisher

pubblicazioni (e articoli) richiedere tra quelli in abbonamento, escludendo quelli di minor interesse.

Tipologie di replica

Central Publisher Lo scenario è quello di un publisher centralizzato che replica i dati ad un numero N di subscriber. Il publisher è il proprietario (primary-owner) dei dati, mentre i subscriber utilizzano le informazioni in modalità read-only. Il Distributor Server (che invia fisicamente i dati) può risiedere sia nel publisher che in una stazione separata, quando i carichi di lavoro rendono pesante la gestione su piattaforma singola.

Publishing Subscriber In questa situazione abbiamo due server che pubblicano gli stessi dati: il publisher invia i dati ad un solo subscriber il quale ripubblica i dati a tutti gli altri subscriber. Questo metodo risulta utile quando il publisher invia i dati su di una rete lenta e costosa: utilizzando il subscriber come sender/receiver possiamo spostare il carico di lavoro in una parte della rete con caratteristiche superiori (di velocità, di costi, ...).

Central Subscriber Abbiamo diversi publisher che replicano i loro dati in una tabella di destinazione comune di uno stesso subscriber. Questa tabella di destinazione è ottenuta dall'unione delle varie tabelle dei publisher (che

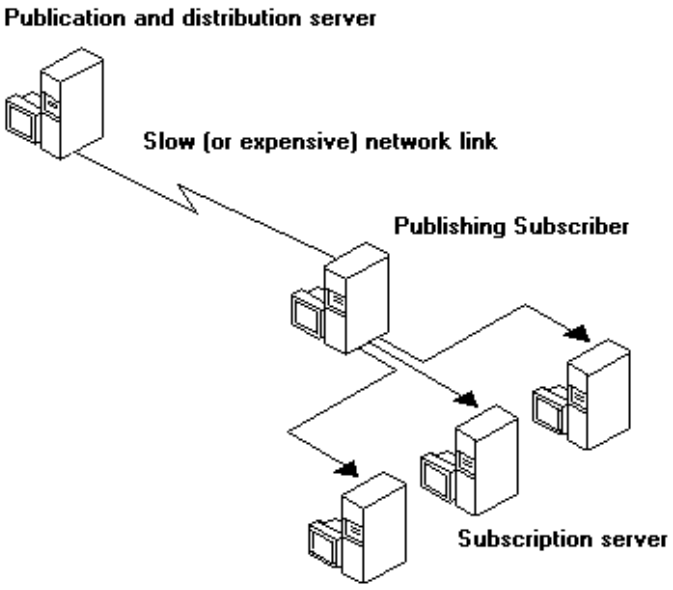


Figura 4.9: Tipologia Publishing Subscriber

non sono tra loro sovrapposte) e ciascuna partizione della tabella di destinazione viene aggiornata da un publisher in tempi e modalità che possono essere diverse in funzione del publisher considerato.

Publisher/Subscriber È possibile avere una situazione in cui due server sono contemporaneamente publisher e subscriber l'uno dell'altro. Il server A pubblica la publication 1 verso il server B, il quale a sua volta pubblica la publication 2 verso il server A.

Multiple Publisher of One Table In questo scenario abbiamo una tabella che è mantenuta su vari server. Ciascun server è proprietario di una partizione orizzontale della tabella (per la quale è publisher) mentre è subscriber verso gli altri server per la rimanente parte dei dati. Ogni server può aggiornare i dati di cui è proprietario e vede (read/only) i rimanenti dati: le operazioni riguardanti i permessi sulle modifiche ai dati sono regolate da Stored Procedures.

Replica di dati di tipo text ed image

Il processo di Replica è limitato per quanto riguarda colonne di tabelle che contengono dati di tipo text ed image, poiché è possibile definire solamente

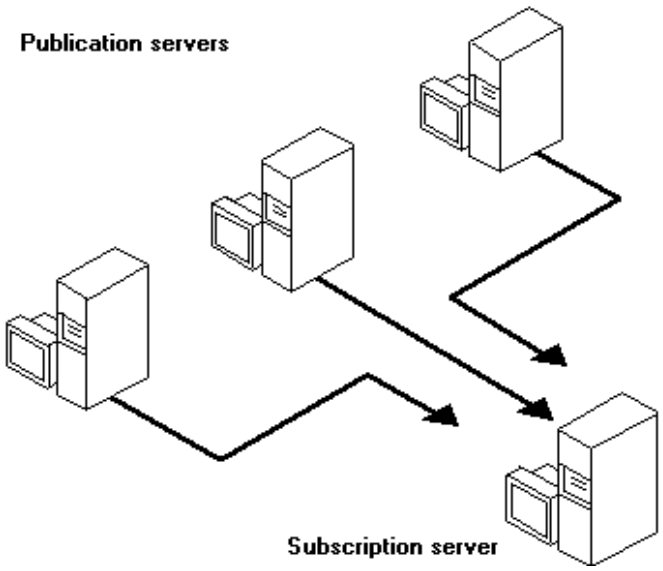


Figura 4.10: Tipologia central subscriber

la modalità di replica Scheduled Refresh mentre quella Transaction Based non è permessa.
Possiamo rendere trasparente l'utente rispetto a questa limitazione attraverso una doppia pubblicazione della tabella che contiene campi di testo o immagine. Ad esempio possiamo pubblicare un articolo A che contiene i campi immagine e testo della tabella e definire per questo articolo una replica di tipo Scheduled Refresh (ad esempio ad intervalli di 1 ora) e poi definire un articolo B che contiene gli altri campi della stessa tabella con una modalità di replica Transaction Based. Avendo previsto di pubblicare in entrambi gli articoli l'attributo TIMESTAMP posso avere una visione globale della tabella determinando anche se le due parti fanno riferimento alla stessa versione.

Replica e ODBC

Un distribution server si collega a tutti i subscription server come un client ODBC (*Open Data Base Connectivity*). Per questo, la replica richiede che il driver ODBC 32-bit sia installato su tutti i distribution server. L' installazione dei driver necessari su Windows NT viene eseguita automaticamente dal programma di setup di SQL Server 6.5. Per quanto riguarda i subscriber, non è richiesta la configurazione degli ODBC Data Sources, poichè il processo di distribuzione utilizza direttamente il nome di rete del subscriber per stabilire la connessione.

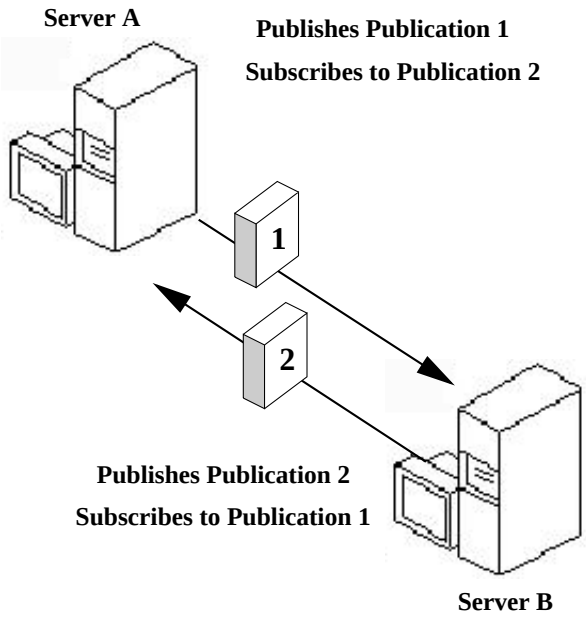


Figura 4.11: Tipologia Publisher/Subcriber

4.2.2 L'ambiente software di implementazione

Per illustrare meglio il funzionamento e le potenzialità di SQL Server, occorre introdurre alcuni aspetti legati all'ambiente software utilizzato per parte del progetto descritto in questa tesi.

Come vedremo più avanti, in questa applicazione verranno generati degli script (ovvero file contenenti uno o più programmi T-SQL[1] da eseguire parte in batch e parte da programma), di inizializzazione e gestione della replica nei sistemi distribuiti in modo completamente automatico, facilitando il compito dell' amministratore del sistema. Con pochi dati sarà possibile configurare completamente le repliche in tutti i server della rete ed utilizzare automaticamente il modello di replica DOM. In più, altri script serviranno a creare dei database necessari all'applicazione e a modificarne degli altri, in accordo alle specifiche date.

Nei prossimi paragrafi introdurremo alcuni strumenti software che permettono di scrivere ed eseguire transazioni distribuite e chiamate a procedure remote, che sono stati necessari per l'implementazione del lavoro svolto.

[1] Il Transact-SQL (*T-SQL*) è una estensione Microsoft del linguaggio ANSI SQL, consente controllo di flusso, stored procedures, triggers e comandi specifici per l' amministrazione dei database.

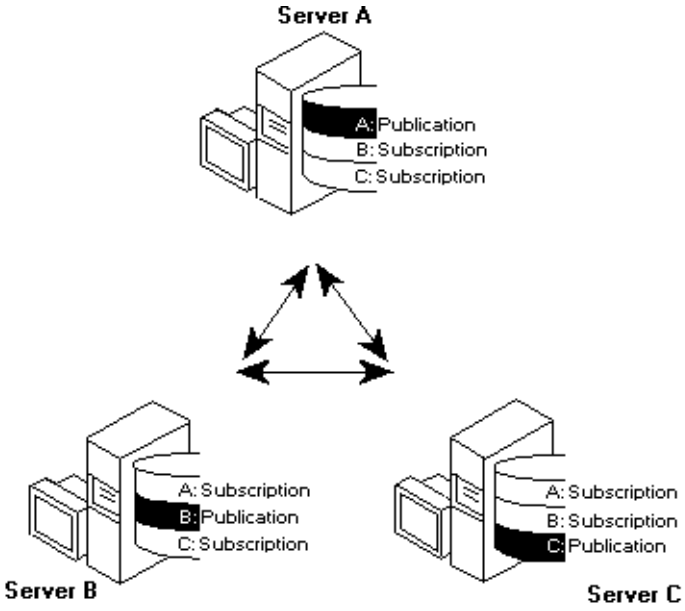


Figura 4.12: Tipologia multiple publisher of one table

Transact-SQL

In questo paragrafo descriviamo brevemente le strutture che mette a disposizione il linguaggio Microsoft T-SQL e che non sono disponibili nell' ANSI SQL. Le variazioni sono soprattutto legate alla gestione delle procedure che permettono di mantenere la consistenza dei dati. Questo viene ottenuto mediante la scrittura di "trigger".

Stored procedures Il T-SQL, come molti RDBMS commerciali, consente la definizione di stored procedure, cioè insiemi di comandi SQL definiti nel seguente modo:

```
create procedure <nome> [<parametri>]
as
<statements>
go
```

La prima volta che si esegue una stored procedure viene compilata e memorizzata dall' SQL Server in una apposita area associata al database nel quale è stata definita. In questo modo l' esecuzione successiva è più veloce e riduce l' overhead imposto dal caricamento e dall' interpretazione della procedura ad ogni esecuzione.

Trigger Un trigger è un tipo particolare di procedura che è eseguita automaticamente quando un utente esegue un comando di modifica (insert, update o delete) su una tabella. I trigger sono di solito utilizzati per effettuare controlli sui dati immessi e per verificare l' integrità dei dati e mantenere uno stato consistente dell' intera base di dati. La definizione di un trigger avviene tramite le seguenti istruzioni:

```
create trigger <name>
on table_name
for INSERT,UPDATE,DELETE
as sql_statements
```

Quando un utente immette uno statement SQL che altera il contenuto della tabella associata al trigger, il trigger viene automaticamente attivato dopo l' esecuzione dello statement. Prima dell' esecuzione delle istruzioni interne al trigger esiste una piccola parte di inizializzazione svolta automaticamente dal DDBMS, vengono infatti definite una o due tabelle temporanee che contengono le righe della tabella modificate. Tali tabelle vengono chiamate *inserted* e *deleted*. Avremo una sola tabella nel caso di un trigger associato ad istruzioni di INSERT o DELETE. Nel caso di INSERT viene creata solo la tabella *inserted*, che contiene le tuple immesse dalla istruzione utente. Nel caso di DELETE è creata solo la tabella *deleted* con le tuple da eliminare prelevate dalla tabella del database. L' UPDATE possiede entrambe le tabelle, la inserted contiene le tuple dopo la modifica, la deleted le tuple originali prima della modifica.
Il trigger può verificare se abbiamo eseguito un'UPDATE su una colonna utilizzando lo statement `IF UPDATE()`. In caso affermativo effettua una comparazione tra i dati prima e dopo la modifica, e nel caso di errore, dopo aver visualizzato (in genere), un messaggio e settato le variabili interne di errore di SQL Server (mediante l' istruzione `RAISERROR`) effettua un `ROLLBACK TRANSACTION` che termina la transazione annullando le modifiche fatte. Quindi le righe modificate tornano al loro valore originale. Altrimenti in caso di UPDATE corretto il trigger si conclude senza rollback e la transazione termina col commit rendendo permanenti le modifiche.
Occorre evidenziare alcune cose: il trigger non sostituisce le normali procedure di controllo dell' integrità dell' SQL Server, in caso ad esempio di immissione di un duplicato di una chiave primaria sarà il DDBMS a informarci dell' errore e il trigger non viene eseguito.
Il trigger può prevedere la gestione di modifiche a più righe contemporaneamente; ma spesso si deve gestire ogni riga in modo separato. A questo scopo vengono utilizzati i *cursori*, cioè opportune strutture che referenziano una

riga alla volta dalla query a cui sono associati. Con l' istruzione `FETCH NEXT` possiamo prelevare una riga indirizzata dal cursore e forzarla nella variabile specificata. L' utilizzo dei cursori è abbastanza costoso in termini di prestazioni, ma è necessario all' interno dei trigger per la gestione singola di query su righe multiple.

Replication Stored Procedures Introduciamo ora alcune stored procedure fornite dal sistema dell' SQL Server per la gestione della replica. Queste procedure sono create dal DBMS all' inizializzazione del sistema di replica e sono associate al database master. Sono utilizzate intenamente dall' SQL Server quando l' utente decide di configurare ed utilizzare la replica sfruttando l' SQL Server Enterprise che è un programma che consente di amministrare i siti locali e remoti in modo visuale. Tuttavia, non sempre la gestione standard della replica soddisfa i requisiti richiesti dagli utenti, di conseguenza l' amministratore del sistema può decidere di configurare autonomamente un sistema di replica ad-hoc per l' applicazione, in modo tale da avere una gestione personalizzata.

Le procedure di gestione della replica in generale possono essere eseguite solo dagli utenti con i previlegi di amministratore, e in alcuni casi solo dai proprietari del database.

In totale le procedure per la replica fornite dall' SQL Server sono 45 suddivise in 6 gruppi funzionali:

1. Server configuration
2. Publication administration tasks
3. Subscription administration tasks
4. Replication operations
5. Replicated transaction management
6. Scheduling

Presentiamo ora per ogni gruppo funzionale le procedure principali, tralasciando quelle che hanno un utilizzo marginale o servono solo in fase di debug. Vedremo brevemente una descrizione del loro funzionamento.

Server configuration Le procedure per la configurazione del server consentono di inizializzare il processo di replica sul proprietario delle informazioni da distribuire. Infatti è chi possiede inizialmente le tabelle che decide chi può abbonarsi e chi no.

- `sp_addpublisher` permette di aggiungere un nuovo publisher server all' elenco dei server collegati al sito in cui ci troviamo. E' possibile specificare un parametro opzionale che consente di identificare il sito remoto come publisher server per il sito di distribuzione in cui viene eseguita.

- `sp_addsubscriber` permette l' aggiunta di un subscription server e aggiunge un login remoto "repl_subscriber" che viene mappato in remoto come "sa" cioè come utente "system administrator".

- `sp_dboption` consente di modificare gli attributi del database specificato, infatti per pubblicare o sottoscrivere un database occorre settare degli opportuni flag booleani: "published" e "subscribed".

Sono presenti inoltre le corrispondenti funzioni per eliminare dei server dalla lista di server connessi: `sp_dropsubscriber` e `sp_droppublisher` e funzioni di debug: `sp_helpdistributor`, `sp_helpserver`.

Publication administration Sono le stored procedure utilizzate dal sistema di replica per l' amministrazione delle pubblicazioni, cioè per la definizione dei dati da pubblicare.

- `sp_addpublication` crea una pubblicazione e ne definisce il tipo. Una pubblicazione può essere definita "restricted" per permettere la pubblicazione solo verso i siti autorizzati.

- `sp_addarticle` crea un articolo (cioè una tabella intera o una partizione) e lo aggiunge alla pubblicazione.

- `sp_articlecolumn` seleziona una colonna della tabella e crea una partizione verticale che può poi essere associata ad un articolo nella pubblicazione.

Le altre funzioni di questo gruppo (oltre alle corrispondenti funzioni "drop" per rimuovere pubblicazioni e articoli e alle funzioni "help" di debug) permettono di modificare gli attributi di un articolo già definito in una pubblicazione, in modo tale da riconfigurare la replica mantenendo definite tutte le pubblicazioni e i server. Infine la `sp_replstatus` che consente di modificare lo stato di una tabella (da replicata a non replicata e viceversa) permettendo di sospendere temporaneamente la pubblicazione di una tabella.

Subscription administration Queste sono le procedure per la gestione delle sottoscrizioni (subscriptions) degli articoli pubblicati da un publisher.

- `sp_addsubscription` consente di aggiungere una subscription all' articolo e settare le opzioni relative al sincronismo: automatico, manuale o nessuno. Viene eseguita sul publisher per definire quale sito è abbonato alla pubblicazione.
- `sp_subscribe` è la duale della precedente, viene eseguita sul subscriber server per aggiungerer un articolo alla subscription.

Anche in questo caso sono presenti le funzioni "drop" per rimuovere le subscription dai server e le "help" per visualizzare lo stato attuale per il debug.

Replication operations Sono le stored procedures utilizzate per le operazioni base della replica, la sola realmente importante è:

- `sp_replica` consente di definire una tabella abilitata alla replica, deve essere aggiunta nel creation script della tabella in modo tale che sia accessibile anche dai siti subscribers.

Le altre gestiscono parte delle operazioni per la creazione delle statistiche utilizzate dall' SQL Server Performance Monitor, cioè il programma che consente il profiling delle operazioni sul database.

Replicated Transaction Management Queste procedure sono definite *extended stored procedure*, sono stored procedure caricate dinamicamente quando devono essere eseguite. Sono memorizzate in file .DLL e la loro esecuzione è gestita e controllata direttamente dal SQL Server. Queste procedure sono di utilizzo meno frequente, pertanto anche se non sono precaricate in memoria le prestazioni globali non ne risentono.

L' unica procedura utilizzata è la:

- `sp_MSkill_job` che permette di eliminare dei job (cioè delle transazioni da replicare) dalla coda dei comandi pronti per la replica, questo permette di recuperare uno stato corretto dopo un errore nella replica.

Questa procedura è importante in quanto il meccanismo di replica di SQL Server è particolarmente debole e sensibile ai malfunzionamenti. Quando la replica di una transazione fallisce (ad esempio perchè la chiave è duplicata o a causa di un errore interno al modulo ODBC del SQL Server) l' amministratore deve riportare manualmente il sistema in uno stato corretto esaminando il contenuto di alcune tabelle di sistema contenenti i job non eseguiti (e quindi quelli che tengono bloccato il sistema) e rimuoverli mediante la procedura.

Replication scheduling Queste procedure sono di fondamentale importanza in quanto si occupano della definizione dei tempi e dei modi in cui devono essere schedulati i task, in particolare quelli di sincronizzazione e distribuzione della replica.

- `sp_addtask` consente di aggiungere un nuovo task associato ad una pubblicazione allo scheduler. Questo task deve essere di tipo "Sync" e si occupa della sincronizzazione periodica delle repliche.
- `sp_droptask` rimuove un task dallo scheduler.
- `sp_helptask` visualizza tutti i task presenti nello scheduler del sistema.

Microsoft Distributed Transaction Coordinator

Le applicazioni in un sistema distribuito sono costituite da componenti che risiedono su siti differenti e comunicano unicamente attraverso messaggi sulla rete. I componenti danno contemporaneamente modularità e distribuzione. Strutturando una applicazione come un insieme di componenti indipendenti si crea il problema della loro gestione. Se un componente fallisce, questo non deve interferire con il funzionamento degli altri, deve esistere un metodo per isolare e limitare la propagazione degli errori. Questo si realizza mediante le transazioni e attraverso i protocolli che sono riportati nell'Appendice B.
La gestione e il coordinamento delle transazioni distribuite in SQL Server è svolta da un modulo denominato *Microsoft Distributed Transaction Coordinator* MS-DTC che riveste entrambi i ruoli di coordinatore (transaction manager) e di partecipante (resource manager), tutti i server del sistema distribuito devono avere il MS-DTC attivo e configurato, in modo tale che possano comunicare durante l' esecuzione delle transazioni distribuite. E' possibile utilizzare il MS-DTC anche per configurazioni client/server, questo permette di utilizzare lo stesso componente per collegare un server appartenente ad un sistema distribuito con dei client locali come rappresentato nella figura 4.13.

Per iniziare una transazione distribuita è possibile utilizzare l' istruzione T-SQL `BEGIN DISTRIBUTED TRANSACTION`. Chi inizia la transazione con il `BEGIN` è il coordinatore che decide sul commit della transazione, lo chiameremo *commit coordinator*. Il coordinatore comunica con i MS-DTC subordinati (i partecipanti) per portare a termine la transazione (utilizzando il protocollo 2PC) e decidere sull' abort o commit globale. In figura 4.14 è rappresentato il protocollo gestito dall' SQL Server.

Il MS-DTC può essere installato parzialmente sui client, dove è sufficiente avere un substrato che non esegue compiti di coordinamento.

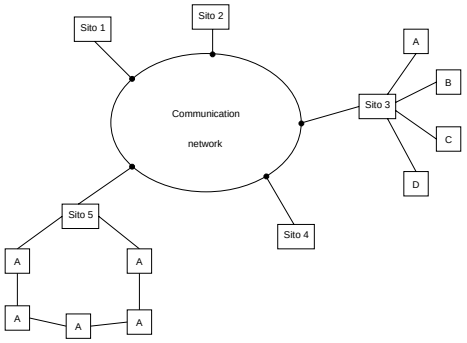


Figura 4.13: Esempio di sistema misto: distribuito e client/server

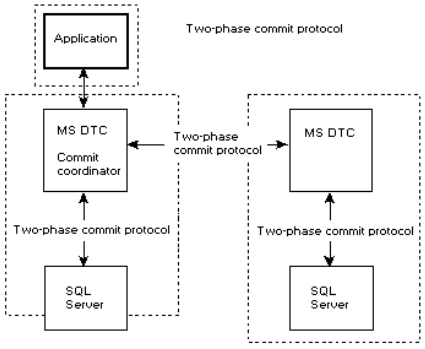


Figura 4.14: Implementazione del two-phase commit protocol nel MS-DTC

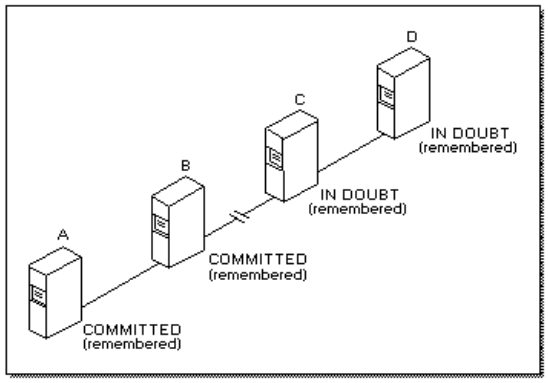


Figura 4.15: Recovery nel MS-DTC

Nel caso di problemi tali da interrompere l' esecuzione della transazione è necessario intervenire manualmente. Consideriamo una situazione composta da quattro server: A (il coordinatore), B, C e D collegati tramite una rete a bus, e supponiamo che sia accaduto un errore sulla rete tra due siti (ad esempio tra B e C) dopo che la fase 1 del 2PC è terminata e il coordinatore ha scritto il pre-commit sul log. La transazione è rimasta in uno stato indeterminato: B ha ricevuto il commit da A, ma C e D non sono raggiungibili quindi la fase 2 non termina e i siti non raggiunti rimangono in uno stato "dubbio".
L' amministratore deve manualmente forzare il server C ad effettuare il commit, essendo la linea di comunicazione tra C e D integra anche D registra il commit.
Inizia la seconda fase, D elimina la transazione e manda la conferma dell' esecuzione verso C, quest' ultimo a sua volta vuole comunicare verso B ma trova la linea ancora interrotta. A questo punto tutti i server hanno effettuato il commit, ma la seconda fase è bloccata sul server C. L' amministratore deve forzare B ad eliminare la transazione come effettuata, quindi B invia ad A la conferma e la transazione distribuita termina.

Capitolo 5

Progetto di un supporto alla gestione di dati distribuiti per applicazioni di workflow

5.1 Motivazioni

Il workflow riguarda l'automazione di procedure in cui documenti, dati e compiti da svolgere vengono passati fra più partecipanti secondo un ben preciso insieme di regole, per raggiungere un obiettivo comune. La maggior parte dei sistemi di workflow si sono concentrati sull'implementazione del flusso di controllo dei processi, trascurando in tutto o in parte gli aspetti relativi ai dati e al flusso loro collegato. Questa debolezza appare evidente nei casi in cui si voglia utilizzare un sistema di workflow di questo tipo per gestire applicazioni orientate ai dati, nelle quali cioè sono i dati il centro del processo. In questi casi, infatti, sarebbero necessarie le capacità di gestione dati tipiche dei DBMS, che mancano però quasi completamente agli attuali sistemi di workflow, nonostante molti di essi utilizzino dei database esterni come deposito dei dati sia del processo sia delle applicazioni.

Inoltre, sempre più nelle aziende medio/grandi, cresce l'utilizzo di applicazioni distribuite, che vanno adeguatamente supportate per mezzo di strategie di replica capaci di soddisfare la necessità di avere ed elaborare i dati localmente. Mediante la replica, si possono fornire gli utenti dei dati giusti, nel posto giusto, al momento giusto. Questi meccanismi, forniti dai DDBMS, implicano però un cambio di prospettiva per le aziende e i sistemi: dal tradizionale approccio client-server ad uno distribuito. Questo passo non è ancora stato compiuto, se non in rarissimi casi, dai produttori di sistemi di workflow, i quali per la maggior parte rimangono ancorati al paradigma client-server.

Da queste osservazioni risulta evidente che per un'adeguata gestione dei processi aziendali, in particolare di quelli *data-intensive*, siano necessarie le funzionalità caratteristiche della tecnologia del workflow e di quella delle basi di dati distribuite.

L'idea della tesi è proprio quella di proporre un'integrazione fra i due tipi di tecnologia per rispondere alle esigenze sopra citate, sfruttando al meglio le potenzialità di entrambe. Infatti, con un sistema integrato in questo modo, possiamo sfruttare il sistema di workflow per il flusso di controllo del processo e il DDBMS per il flusso di dati legati al processo stesso, fornendo un ambiente distribuito per applicazioni di workflow orientate ai dati.

In questo capitolo presentiamo il progetto sviluppato sulla base di queste idee, facendo riferimento a due sistemi commerciali: ActionWorkflow System (Action Technology Inc.), come sistema di workflow e il suo server di appoggio SQL Server (Microsoft), come DDBMS. Il primo, originariamente basato su una architettura client-server (vedi anche Capitolo 4), verrà potenziato con funzionalità distribuite per mezzo del sistema di replica di SQL Server; il secondo verrà arricchito con il modello di replica analizzato nel Capitolo 3, il Dynamic Ownership, che permette di gestire flussi di dati fra diversi utenti.

Esempio di riferimento Nel seguito utilizzeremo ancora l'esempio dell'Adozione per chiarire meglio la nostra proposta; qui se ne vogliono sottolineare ancora le caratteristiche che lo rendono adatto al problema.

Ogni procedura di inserimento di un minore nelle liste di adozione nazionale comporta la gestione di diversi documenti legali che passano di ufficio in ufficio e di partecipante in partecipante venendo eventualmente modificati nei vari passaggi. Questi documenti, inoltre, possono essere essi stessi formati da altri documenti (ad esempio la pratica può essere divisa in fascicoli; i fascicoli possono riferire testi di legge e così via). A livello pratico, questa mole di informazioni deve essere modellata per poter essere memorizzata in forma elettronica in un database. In più, le informazioni devono essere rese disponibili agli utenti (i partecipanti alla procedura), nei siti (uffici), opportuni e con i corretti permessi per la loro modifica. L'altro requisito fondamentale di questo esempio, è la necessità di controllare l'esecuzione della procedura, rilevare informazioni relative al suo svolgimento per eventuali controlli futuri, automatizzare per quanto possibile le operazioni di passaggio di documenti fra i partecipanti e le azioni di modifica degli stessi. In parole semplici, si richiede di implementare la procedura come un processo di workflow. In conclusione, quindi, tre sono i punti fondamentali che caratterizzano l'esempio:

- necessità di adeguate funzionalità di gestione dati;

- ambiente distribuito;
- implementazione della procedura mediante un sistema di workflow.

Queste caratteristiche sono proprio quelle che il nostro progetto si propone di garantire.

5.2 Workflow distribuito

In questa sezione ci proponiamo di mostrare come sia possibile portare un sistema di workflow originariamente client-server ad operare virtualmente come un sistema distribuito. Il modello di workflow utilizzato e la modellazione dell'esempio vengono qui introdotti.

5.2.1 Modello di workflow

Per motivi di generalità della trattazione, utilizzeremo come modello di riferimento quello proposto dalla Coalition (vedi Capitolo 2), anche se poi non mancherà il collegamento al modello proprio del sistema utilizzato (mostrato in Fig 5.5).
Analizzando le specifiche dell'esempio citato, possiamo innanzitutto ricavare il modello della procedura così come è mostrato in Fig 5.1.
Con le convenzioni specificate per i DST, poi, possiamo "colorare" l'AN dell'esempio con i ruoli fornendo un'informazione più completa di quella che la Coalition permette (il grafo definitivo è in Fig 5.2).

5.2.2 ActionWorkflow: da client-server a virtualmente distribuito

ActionWorkflow System viene ilusstrato nel Capitolo 4, richiameremo qui soltanto i concetti più importanti legati al contesto specifico.
ActionWorkflowSystem è un WFMS di tipo *client-server* che funziona in connessione con un *back-end server*, nel nostro caso MS-SQL Server, che si occupa della gestione dei suoi due database. In particolare, il workflow engine, ActionWorkflow Process Manager, è la parte server del sistema, e si interfaccia direttamente con SQL Server per interagire coi database *aws* (che è il database di lavoro, su cui cioè vengono memorizzate le definizioni di processo, lo stato ed i dati delle istanze attive nel sistema), e *awsarch* (che è il database di archivio). In ultima istanza, infatti, tutte le azioni compiute sulle istanze dei processi dagli utenti del sistema sono trasformate dal Process Manager in operazioni di lettura/scrittura/modifica su tabelle specifiche di

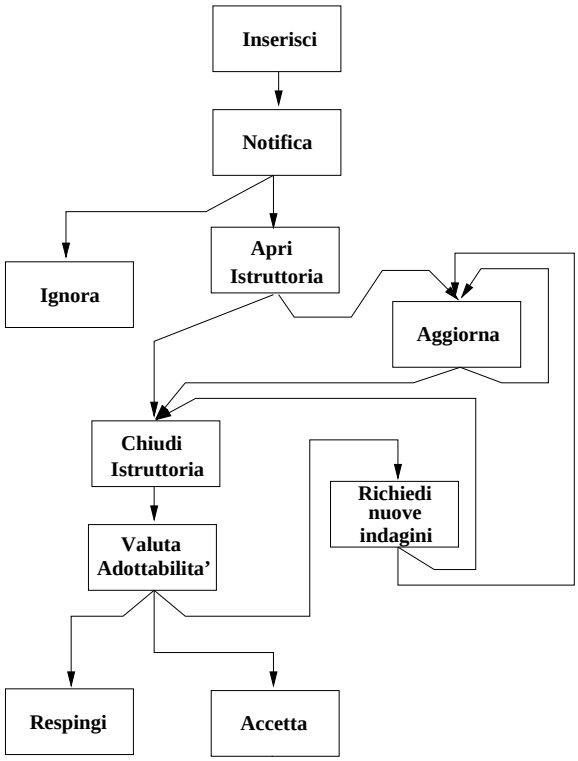


Figura 5.1: Il workflow della pratica di esempio

aws. Da questa osservazione di base nasce l'idea su cui si fonda il progetto spiegato nel presente Capitolo. Se infatti il funzionamento del sistema di workflow si basa su tabelle di un database, è ragionevole pensare che replicando i dati in esse contenute si possano sincronizzare due o più server di ActionWorkflowSystem facendoli funzionare come se fossero un unico sistema distribuito.
I passi necessari a rendere virtualmente distribuito ActionWorkflow, sono i seguenti:

- installazione del Process Manager (e automaticamente dei database di servizio), in ogni sito della rete;
- replica del database *aws* in ogni sito della rete;
- sincronizzazione di tutte le copie dei database per mezzo del sistema di replica di SQL Server.

Una volta portate a termine queste operazioni di configurazione, è possibile far operare virtualmente tutti i server sulla stessa istanza di processo, anche se in realtà ci sono tante istanze "gemelle" quanti sono i server della

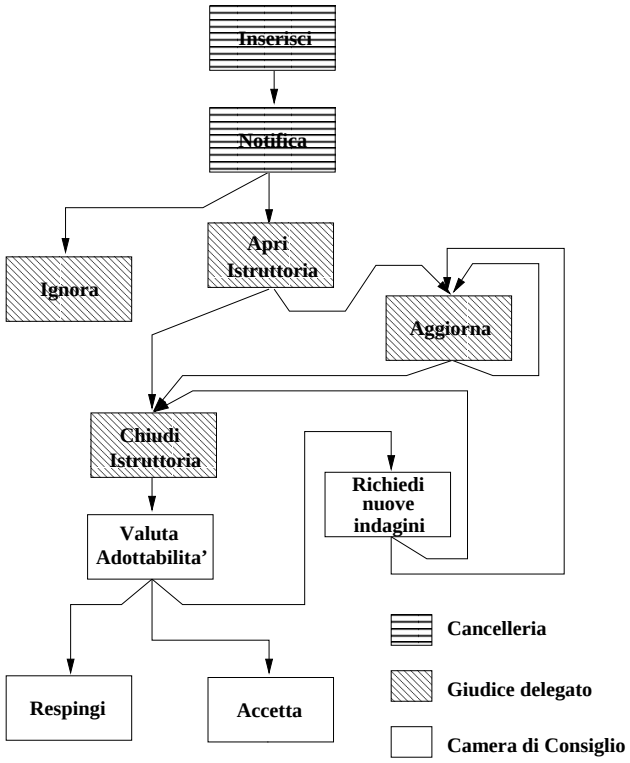


Figura 5.2: Il workflow con i ruoli della pratica di esempio

rete attive una in ogni sito. Per questa particolarità continueremo a definire il sistema proposto come *virtualmente distribuito*, in quanto in un sistema realmente distribuito l'istanza attiva sarebbe una sola che viene condivisa fra i vari siti.

5.3 Data-flow distribuito

Come precisato in precedenza, al sistema di workflow manca la capacità di un'adeguata gestione dei dati e dei documenti coinvolti nel processo da automatizzare. Per le sue caratteristiche, ampiamente illustrate nel Capitolo 3, il modello di replica Dynamic Ownership si presta bene ad essere utilizzato per risolvere il problema. Con esso, infatti, possiamo tracciare il data-flow e implementarlo adeguatamente in un sistema distribuito. Richiamiamo brevemente i concetti chiave del modello, illustrando le caratteristiche che lo rendono particolarmente adatto ad'accoppiamento con un sistema di workflow.
Innanzitutto, il modello prevede il passaggio di dati fra più utenti che a turno

intervengono su di essi modificandoli, fino al termine del loro ciclo di vita. La corretta gestione delle modifiche ai dati, che non presenta problemi di concorrenza, viene implementata mediante il meccanismo del cambiamento dinamico della proprietà dei dati stessi e la restrizione che solo il proprietario dei dati ha i permessi necessari per effettuare su di essi operazioni di modifica. In sostanza, quindi, il DOM si presenta come un meccanismo semplice ed efficace per implementare flussi di dati in ambienti distribuiti, coinvolgenti più utenti: una sorta di sistema di workflow orientato ai dati, per i quali fornisce capacità di gestione nettamente superiori ai sistemi commerciali.

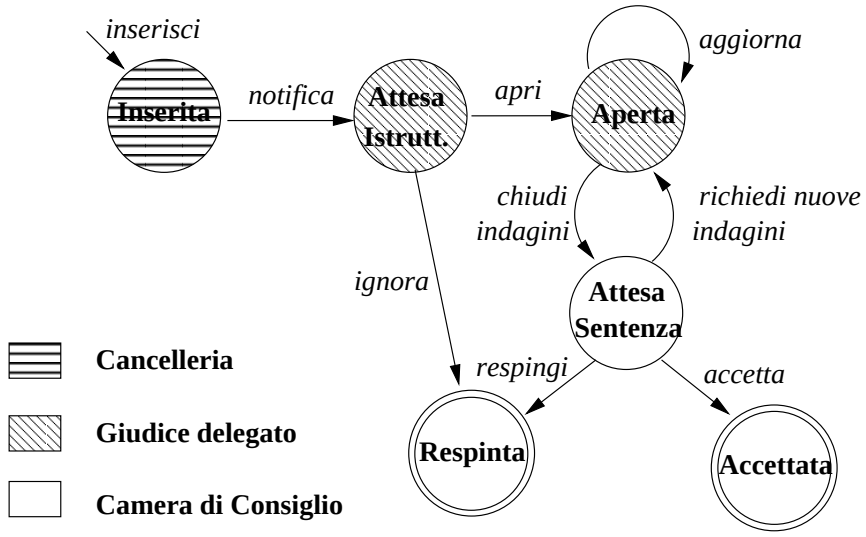


Figura 5.3: Il diagramma stati-transizioni della pratica

Ciò che dobbiamo fare, quindi, all'interno del nostro progetto, per implementare il flusso di dati, è disegnare il diagramma stati transizioni della pratica (che riportiamo in Fig 5.3), ed utilizzare poi il DOM per attivarlo.

5.4 Architettura del sistema

In questa sezione descriviamo l'architettura funzionale del sistema proposto, spiegando come dovrebbe avvenire l'integrazione fra ActionWorkflow System ed SQL Server con il suo sistema di replica. L'idea base per accoppiare il flusso di controllo e quello dei dati è di associare ad ogni azione del sistema di gestione di workflow la corrispondente azione sui dati e il passaggio degli stessi fra i DBMS locali, sfruttando il sistema di replica. Soluzioni analoghe sono state proposte anche in [1] e [2], dove la gestione dati (documenti), è affidata a Lotus Notes [8, 9, 10]. In questo modo, tutti i dati necessari o

utili per un'attività di workflow, sono resi disponibili al corretto sito locale, mentre loro copie read/only sono presenti negli altri siti.

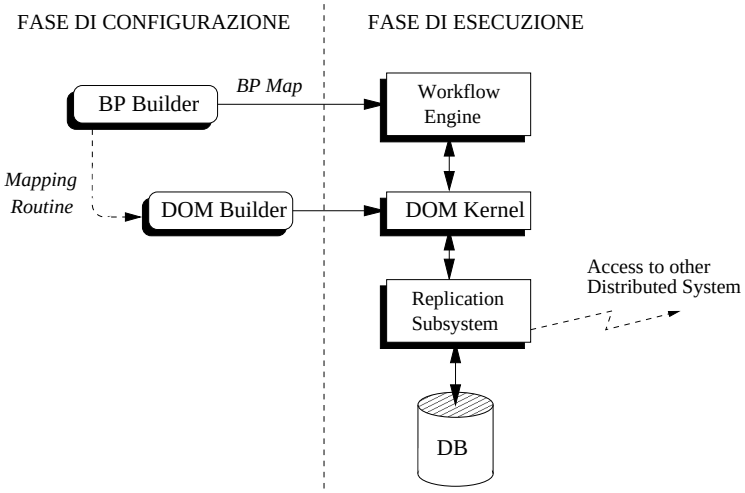


Figura 5.4: Architettura del sistema

L'architettura, mostrata in Fig 5.4, evidenzia le due fasi del funzionamento del sistema: la fase di *configurazione* e quella di *esecuzione*.

5.4.1 Fase di configurazione

In questa fase vengono definiti il modello del processo, il flusso di dati e di controllo e viene impostata la replica per la distribuzione dei dati. In particolare, vengono utilizzati i seguenti sistemi:

- **ActionWorkflow Process Builder**
- **DOM Builder**

ActionWorkflow Process Builder È il modulo software di ActionWorkflow System che permette di definire il modello del processo da implementare. Tramite questo tool, è possibile disegnare la "mappa" del processo (un esempio è mostrato in Fig 5.5, si noti la differenza con l'Activity Net), definire i dati del workflow, i ruoli partecipanti ad esso, controllare la correttezza e la coerenza del modello preparato alle regole di ActionWorkflow. La prima cosa da fare è proprio disegnare il processo e renderlo disponibile per l'esecuzione tramite l'Administrator, compilandolo e generando così il file con estensione *awo* che può essere installato nel sistema (è la cosiddetta Business Process Map). Inoltre, per permettere l'integrazione con il DOM, è necessario definire negli script il mapping fra lo schema di workflow e i dati di cui vogliamo

controllare il flusso (nella Fig 5.4 questa operazione è rappresentata dalla "Mapping routine").

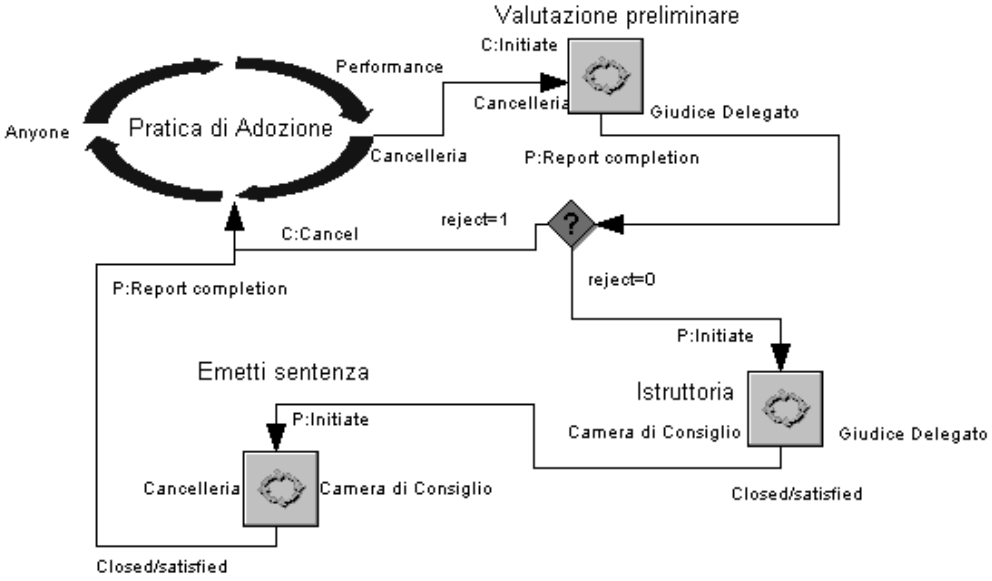


Figura 5.5: Il workflow della pratica di esempio secondo ActionWorkflow

DOM Builder È il nostro modulo software che permette la connessione fra il flusso di controllo e quello dei dati. L'idea è di partire dagli script definiti nel Process Builder estendendo poi il linguaggio di workflow mediante una libreria dedicata per automatizzare il mapping citato. Il progettista dovrebbe nel Process Builder utilizzare questa libreria per definire la routine di mappaggio, in modo che poi il DOM Builder possa da essa automaticamente generare i trigger e le procedure necessarie ad implementare il DOM stesso. L'idea è di automatizzare il più possibile le operazioni di configurazione del sistema, divise però nei due momenti distinti di costruzione del processo di workflow e implementazione del DOM sul flusso di dati. La routine di mappaggio è tesa ad unificare le due fasi, permettendo al progettista di concentrare gli sforzi in un unico ambito ed operare poi in modo semplice e immediato nell'altro. L'idea della routine di mappaggio nasce dall'osservazione della possibilità di identificare, nel file ".awo" (la mappa compilata), le procedure scritte nel linguaggio di scripting di ActionWorkflow. Una volta definita la "Mapping routine", quindi, nel DOM Builder basta leggere dal file citato la parte relativa ad essa, per avere tutte le informazioni necessarie all'implementazione corretta e semi-automatica del data-flow e della replica dei dati.

5.4.2 Fase di esecuzione

Durante la fase di esecuzione, il sistema di workflow distribuito gestisce le istanze dei processi. Il funzionamento, per la natura del sistema, risulta essere dato dall'azione combinata di tanti motori di workflow quanti sono i siti, i quali si scambiano informazioni sull'esecuzione delle istanze tramite il mccanismo di replica del loro database di servizio. Inoltre, parte della gestione dei processi, quella relativa al flusso di dati, è lasciata al DOM Kernel e al DDBMS. Vediamo dunque meglio i ruoli di questi sottosistemi nella fase di esecuzione.

- **Workflow Engine:** è l'ActionWorkflow Process Manager che utilizza il database di servizio *aws* per memorizzare le mappe dei processi una volta compilate e per tutte le operazioni di gestione delle istanze degli stessi (esecuzione di atti, terminazione di atti, modifica dello stato delle istanze, aggiornamento dei dati di controllo del workflow, ...). Per una descrizione più dettagliata si rimanda al Capitolo 4, qui interessa ricordare solo alcune caratteristiche del sistema. Il Process Manager consiste principalmente di un processo "scheduler" per organizzare l'esecuzione delle istanze dei processi secondo determinate politiche, e di un "transaction manager" che si occupa dell'esecuzione vera e propria. In particolare, questo secondo processo deve essere attivo perchè le transazioni generate dall'esecuzione di un atto o da una qualsiasi operazione sulle istanze di un processo vengano rese effettive anche sul database *aws*. Ricordiamo ancora che il motore del sistema distribuito, così come noi l'abbiamo implementato, è in realtà un insieme di singoli Process Manager adeguatamente coordinati via replica, che operano solo virtualmente sulle stesse istanze.

- **DOM Kernel:** è uno "strato" software che si occupa della gestione del mapping fra flusso di controllo e di dati e della realizzazione dei cambiamenti dinamici di proprietà dei dati. Viene generato automaticamente dal DOM Builder e consiste di trigger, RPC, stored procedure.

- **Replication Subsystem:** è il componente di SQL Server su cui si basa la gestione/implementazione sia del flusso di dati dell'applicazione (tramite il DOM), sia del flusso di dati del workflow (per abilitare la distribuzione di ActionWorkflow). Come già indicato nel Capitolo 3, il sistema è implementato su di una rete a stella, il cui centro, detto *sito master*, coordina la replica verso i siti periferici i quali a loro volta mantengono le informazioni sui ruoli abilitati a compiere le operazioni in locale. La replica relativa ai dati dell'applicazione è implementata

tramite pubblicazioni di tipo **PUSH** (dal server locale verso il master), e **PULL** (dal master server verso la periferia). Ricordiamo ancora (rimandando sempre al Capitolo 3 per i dettagli), che per l'implementazione del DOM i dati modificati in un sito vengono replicati con una pubblicazione PUSH, verso il master, che poi li rende disponibili in lettura a tutti gli altri siti con altrettante pubblicazioni PULL. È il DOM Kernel ad occuparsi della definizione e dell'inizializzazione delle pubblicazioni sfruttando il diagramma stati-transizioni del DOM Builder. La replica relativa ai dati contenuti nei vari database *aws*, viene implementata ancora tramite il DOM Builder (opportunamente arricchito delle funzionalità necessarie), e deve funzionare da/verso il sito master, per un corretto funzionamento del workflow distribuito. Come vedremo nel capitolo seguente, quest'ultima operazione è una delle più complesse e "pesanti" dell'implementazione dell'intero sistema.

- **DB:** in accordo ai requisiti del DOM, in ogni sito della rete, i database dell'applicazione vengono arricchiti con nuove tabelle (le tabelle di servizio del DOM), nuovi trigger e stored procedure, campi aggiuntivi nelle tabelle per le quali verrà implementato il flusso (campi "stato" e "validità"). In particolare, in ogni database della rete verranno memorizzati tre tipi di dati:
  - Tabelle del DOM: le tabelle di servizio viste nel Capitolo 3;
  - Trigger di controllo: le procedure di gestione delle transizioni di stato;
  - Dati dell'applicazione: i dati su cui il processo implementato opera durante l'esecuzione delle istanze.

5.5 Setup del sistema

In questa sezione elenchiamo i passi da compiere, nel corretto ordine, per impostare e configurare correttamente l'intero sistema.

1. *Installazione di ActionWorkflow System ed SQL Server in ogni sito della rete.* Come detto sopra, per fornire le funzionalità distribuite richieste dall'applicazione, ogni singolo sito deve essere dotato sia dell'engine di workflow (ActionWorkflow Process Manager), sia del server DDBMS (SQL Server), per gestire il flusso di controllo e di dati a livello locale. Il Process Builder di ActionWorkflow invece, può essere installato indipendentemente dal resto, su qualsiasi sito; ma il suggerimento

è di renderlo disponibile solo sul Master Server, per coerenza e semplicità. È da lì, infatti, che partono le impostazioni per l'intero sistema, quindi pare logico sceglierlo come centro anche per la definizione dei processi.

2. *Installazione della definizione del processo sul Master Server.* Una volta preparata mediante il Process Builder la opportuna definizione di processo (la mappa), questa deve essere importata nel Process Manager del Master Server. In questo modo il database *aws* del Master verrà aggiornato con tutte le informazioni necessarie a riconoscere e gestire il processo.

3. *Sincronizzazione di tutti i motori di workflow del sistema.* Per fornire un'unica definizione di processo per l'intero sistema, è necessario replicare le tabelle dell'*aws* del Master verso tutti i siti periferici della rete. In questo modo, il processo viene installato (e quindi eseguito), come se fosse uno solo gestito in modalità distribuita. Notiamo che poichè all'atto dell'installazione vengono generati automaticamente degli ID di processo tramite contatori, è fondamentale che prima di quest'operazione tutti i database di servizio di ActionWorkflow siano "puliti". Segnaliamo inoltre, ma lo vedremo meglio nel prossimo capitolo, che la configurazione della replica di *aws* viene eseguita quasi totalmente in automatico per mezzo del DOM Builder.

4. *Autorizzazioni agli utenti per la partecipazione al processo.* Il motore di workflow controlla gli accessi al sistema attraverso il sistema dei ruoli che deve essere opportunamente definito. Nel nostro caso, però, quest'operazione deve essere eseguita facendo riferimento alla allocazione dei ruoli sui diversi siti. In particolare, per prevenire conflitti di update concorrenti sulle istanze e sui dati, ogni ruolo dovrà essere legato ad un solo sito, mentre sullo stesso sito potranno risiedere più ruoli. Questo vincolo (che deriva dal DOM), non è in realtà troppo restrittivo: è facile pensare infatti, che ogni ruolo coinvolto in un processo aziendale sia legato anche fisicamente ad uno specifico ufficio o dipartimento (sito).

5. *Generazione del DOM Kernel.* Dalla "mapping routine" del Process Builder, il DOM Builder genera il DOM Kernel. Ancora una volta, l'operazione viene effettuata sul Master Server, poi replicata a tutti i siti. Il DOM Kernel, la cui composizione è stata già data poco sopra, viene memorizzato nel database dell'applicazione sotto forma di tabelle, trigger, campi aggiuntivi, e fornisce il sistema delle opportune funzionalità di gestione del flusso di dati all'interno del processo.

Una volta eseguiti i passi di configurazione elencati sopra, è possibile far partire una nuova istanza di processo. Escluso il primo passo, che viene eseguito una volta per tutte, tutte le operazioni vanno ripetute ogni volta che si vuole installare un nuovo processo nel sistema. In particolare, ogni cambiamento della configurazione va eseguito prima di tutto sul Master server, per evitare problemi con la replica dei dati.

Capitolo 6

Implementazione del sistema

Lo scopo di questo capitolo è l'illustrazione del lavoro svolto per modificare il DOM Builder che permette di generare gli script per il DOM, con un eseguibile di nome **Wgen**, e per implementare il sistema proposto nel Capitolo 5.
Nel corso di questa parte del lavoro di tesi sono stati utilizzati i pacchetti di Microsoft "Visual C++ 5.0" e "Visual Basic 4.0" (il secondo in modo marginale), e il linguaggio di scripting di SQL Server 6.5, cioè il "Transact-SQL". In particolare, il primo è servito per la modifica del programma che realizza il DOM Duilder, *Wgen*, implementato nella sua prima versione in "Visual C++ 4.0"; il secondo per lo studio e la modifica degli esempi di applicazione di interfaccia per ActionWorkflow; il terzo infine è il linguaggio utilizzato per scrivere tutti i trigger, le stored procedure, ed eseguire in generale tutte le operazioni di configurazione di SQL Server e del suo sistema di replica.

6.1 Il DOM Builder

6.1.1 La prima versione

Il DOM Builder da poche ma mirate informazioni è in grado di produrre una serie di *script* completi in linguaggio Transact-SQL che permettono di configurare SQL Server per la replica secondo il paradigma del Dynamic Ownership Model (vedi Fig 6.1). Con esso è possibile definire la topologia della rete di server, le tabelle che entrano nel flusso di dati distribuito, il DST dei dati, le specifiche della replica (ad esempio la modalità di sincronizzazione delle copie, lo scheduling dei task, etc.). La versione *1.0* (sviluppata in una tesi precedente [22] e il cui funzionamento è illustrato in Fig 6.1), ri-

sulta carente in alcune parti, ponendosi come primo prototipo da sviluppare ulteriormente.

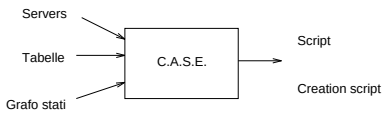


Figura 6.1: Funzionamento del C.A.S.E.

In particolare, le mancanze principali del programma nella sua prima versione sono:

1. Gestione limitata degli errori, sia nel programma stesso, sia negli script generati.

2. Incapacità di collegamento a database esistenti. Questo fatto comporta la reimmissione della struttura del database da utilizzare nell'applicazione, all'interno del programma, anche se il database è già presente in SQL Server.

3. Generazione degli script di configurazione di SQL Server limitata:

   - non vengono generati gli script per la costruzione o modifica delle tabelle definite da programma, cioè le tabelle modificate con i campi aggiuntivi e i nuovi vincoli di integrità. Questi script sono però necessari, in quanto vengono richiamati negli script di configurazione della replica generati da programma;
   - la creazione di *device* in SQL Server viene effettuata senza controllo (numero di device unico);
   - negli script si utilizza il database selezionato all'interno del programma che però, se non esiste, deve essere costruito manualmente esternamente al programma;
   - gli stessi script vengono generati in forma di file da eseguire poi manualmente in SQL Server.

4. Gestione tabelle:

   - non c'è gestione campi "null";
   - non c'è gestione chiavi esterne, nè all'interno del database, nè nelle tabelle di servizio;
   - limiti un po' bassi per il numero tabelle (MAXTAB=30), di un database e per il numero dei campi in una tabella (sempre 30).

6.1.2 Le modifiche apportate al DOM Builder

Il mio lavoro sul programma è stato rivolto principalmente alla risoluzione dei problemi legati al reperimento delle informazioni in caso di database esistenti. Su questa linea, poi, ho cercato di rendere l'integrazione del programma con SQL Server la più ampia possibile, evidenziando e sfruttando la possibilità di esecuzione di istruzioni SQL (o Transact-SQL), direttamente da programma. Ho migliorato la gestione complessiva degli errori e ampliato le funzionalità di gestione delle tabelle del database. Nella pratica il mio intervento ha apportato le seguenti modifiche al programma:

Connessione via ODBC a database esistenti È stata inserita nel programma una voce di menù (e un pulsante corrispondente), con la funzione di aprire una connessione ODBC verso qualsiasi DBMS che che supporti questo protocollo. Per abilitare questa connessione si è fatto uso della classe *CDatabase* (già presente nel pacchetto di sviluppo come componente della MFC Library, come le altre classi citate nel seguito), ed il suo metodo *Open*. In questo modo è possibile aprire una finestra di dialogo da cui si può scegliere la sorgente dati (database), cui connettersi o configurare per la connessione nuove sorgenti. Una volta stabilita la connessione, eventualmente fornendo i dati di login per i DBMS che ne facciano richiesta (è il caso di SQL Server, ma non di Access, ad esempio), si possono recuperare tutte le informazioni sul database, sfruttando apposite classi messe a disposizione dalla libreria del Visual C++ (che a loro volta utilizzano nei metodi chiamate al set di API proprio dell'ODBC). In particolare, le operazioni effettuate da programma vengono elencate qui di seguito.

- Lettura dei cataloghi del database selezionato e reperimento delle informazioni sul numero e nome delle tabelle presenti (classe *CTables*).

- Col nome delle tabelle come parametro, lettura e archiviazione della struttura delle stesse nella classe proprietaria *Tab* (opportunamente modificata rispetto alla versione precedente per supportare l'aggiunta delle nuove informazioni). In particolare, per ogni campo si memorizzano nome, tipo e ammissibilit'ø meno dei "null" (informazioni reperite mediante la classe standard *CColumns*). Si mantengono poi informazioni sulle chiavi primarie (dalla classe *CPrimaryKeys*), mediante una struttura dedicata e un flag nell'elenco dei campi, e su quelle esterne (dalla classe *CForeignKeys*), mediante una struttura che per ogni campo della tabella che fa parte di una foreign key mantiene il nome della chiave, il nome del campo referenziato, il nome e il possessore della

tabella referenziata; in più è presente un contatore che mantiene il numero di campi facenti parte di foreign key. Come vedremo in seguito, inoltre, è stato possibile recuperare le informazioni relative agli indici presenti sulle tabelle (dalla classe *CStatistics*), memorizzate ancora nella classe *Tab* estesa opportunamente.

Notiamo qui, per completezza, che queste informazioni non sono sempre reperibili via ODBC, a causa delle differenze fra i driver forniti per i diversi DBMS. In particolare si distinguono due livelli di compatibilità dei driver ODBC, con supporto crescente per le informazioni da reperire. La classe *CPrimaryKeys*, ad esempio, sfrutta delle API che il driver di Access a nostra disposizione non riconosce; mentre funzionano con quello di SQL Server.

- Creazione di una finestra di dialogo apposita per mostrare all'utente quanto letto dai cataloghi: nome del database, nome delle tabelle e, per ogni tabella, la sua struttura (nome e tipo dei campi, etc.). Dalla finestra è possibile in seguito selezionare la tabella per la quale verrà implementato il DOM. Le informazioni qui raccolte, poi, vengono convogliate nelle strutture già approntate nella versione precedente del programma con cui si è mantenuta piena compatibilità.

Generazione degli script aggiuntivi L'implementazione del DOM prevede che alla tabella selezionata vengano aggiunti i due campi "stato" e "validità", e che il secondo entri a far parte della chiave primaria della tabella stessa. Nella prima versione del programma, creando da zero le tabelle, non si presentava nessun problema in questa fase. Leggendo però tabelle esistenti e volendo modificarle opportunamente, sono stati incontrati due problemi notevoli:

- Problema dell' ALTER TABLE: non è possibile aggiungere ad una tabella campi che non ammettano i NULL (requisito fondamentale perchè quei campi possano entrare a far parte di una chiave primaria). In particolare, quindi, non è possibile implementare il DOM su tabelle esistenti senza ricrearle da zero.

- Problema delle FOREIGN KEY: se la tabella che viene modificata con l'aggiunta dei campi "stato" e "validità", è referenziata da altre tabelle mediante vincoli di foreign key, non è possibile mantenere intatti i vincoli di integrità referenziale fra le tabelle senza modificare anche queste ultime.

La scelta effettuata per risolvere il primo problema (e sfruttata poi anche per il secondo), è stata quella di costruire un database copia di quello originale, ma arricchito con le tabelle di servizio e con le tabelle del DOM ricreate ad hoc con tutte le caratteristiche necessarie (le altre tabelle sono ricreate inalterate). Questa scelta ha comportato la generazione di due nuovi tipi di script:

1. *Script di creazione del database di copia*
2. *Script di creazione delle tabelle del database*

Nel primo tipo di script, ho creato ed utilizzato una stored procedure per SQL Server ("mynewdev", che è possibile trovare in Appendice C), per creare un device per i dati e uno per il log. Con essa è stato possibile risolvere il problema, presente nella prima versione, della generazione di un ID unico per i device di SQL Server. Nel secondo tipo di script è bastato trasferire su file, secondo la corretta sintassi Transact SQL, le informazioni già raccolte via ODBC.
Per il secondo problema, è stato necessario identificare tutte le tabelle dipendenti da quella selezionata e modificarle aggiungendo ad esse un campo fittizio, "val_ref", dello stesso tipo di "validità" (integer). Questo campo, poi, è stato inserito nelle foreign key per referenziare il campo "validità" della tabella selezionata. Solo così, infatti, era possibile mantenere i vincoli d'integrità esistenti, dato che non è ammesso per una foreign key referenziare solo una parte della chiave di un'altra tabella. Per la trasparenza all'utente, poi, si è dato a questo campo il default "0", in modo che il record referenziato sia sempre quello "valido" secondo il DOM.
Da ultimo, si è pensato di generare sia i singoli file di script (uno per la creazione del database, più uno per ogni tabella da creare), sia un unico file contenente tutti gli altri opportunamente ordinati (per facilitarne l'esecuzione). Questo secondo file, in particolare, è stato creato fondendo in modo intelligente gli altri, per non generare errori dovuti a mancato rispetto delle dipendenze fra le tabelle. In particolare, la prima operazione è la creazione dei device e del database; in seguito va effettuato il "drop" delle tabelle partendo da quelle non referenziate da nessun'altra tabella e infine il "create" che richiede però un ordine inverso. Un'apposita procedura di ordinamento delle tabelle è stata scritta e inserita nel programma.

Esecuzione degli script aggiuntivi Nella prima versione del programma, mancando la connessione diretta con il server, non era stata esplorata la possibilità di eseguire da programma gli script di configurazione della replica, e questa operazione era lasciata all'utente, da riga di comando di SQL Server.

Nella versione attuale, invece, è stata messa in luce questa possibilità, che è stata sfruttata per eseguire gli script aggiuntivi (ma non è difficile estenderla per eseguire anche quelli di replica). La funzione utilizzata è ancora una volta un metodo della classe *CDatabase*, chiamato *ExecuteSql*. Esso, utilizzabile dopo essersi connessi via ODBC ad una fonte dati, prevede un parametro, in formato stringa, che può essere un qualsiasi statement Transact SQL, e dà all'utente la possibilità di intercettare eventuali errori nell'esecuzione. Nel programma è bastato quindi leggere comando per comando (fermandosi ad ogni istruzione "go") il file di script generale, ed eseguirlo. La scelta fatta in caso di errore, poi, è stata di interrompere l'esecuzione e invitare l'utente a rieseguire lo script direttamente su SQL Server. Da notare anche qui un problema: l'istruzione di "go" che separa due comandi in un file di script, non viene accettata eseguendo gli statement da programma, mentre funziona correttamente nell'esecuzione da SQL Server. La scelta è stata quella di lasciare il "go" nei file di script, salvo poi saltarlo se l'esecuzione era effettuata da programma.

Infine, va detto che sull'esecuzione degli script è stata lasciata libertà di scelta all'utente, che una volta creati i file, può decidere di lanciarli subito (mediante un apposito pulsante aggiunto alla toolbar), o di eseguirli a posteriori, da SQL Server.

Sequenzializzazione delle operazioni Nella prima versione, come detto, molti controlli mancavano, e l'utente poteva compiere operazioni non corrette se non perfettamente a conoscenza del programma. Mediante opportuna abilitazione/disabilitazione di pulsanti o voci di menù, si è cercato di aiutare l'utente a comprendere la corretta sequenza delle operazioni da compiere in particolare per quelle azioni che, per motivi logici e di prevenzione di errori, non ammettono un'esecuzione "random".

Ossrvazioni Il lavoro svolto sul DOM Builder ha riguardato solo alcuni aspetti dello stesso, quindi rimane molto ancora da fare. In particolare, si possono implementare le modifiche al modello Dynamic Ownership indicate nella tesi, prima fra tutte la gestione degli stati finali, e migliorare l'interfaccia verso l'utente, superando alcuni limiti (quali quello di non poter operare su uno stesso DST in due sessioni di lavoro diverse, o su due DST nella stessa). Il nostro scopo, però, era semplicemente quello di eliminare il problema dell'interfacciabilità con SQL Server, ed è stato risolto.

6.2 Il sistema distribuito

L'implementazione del progetto descritto nel Capitolo 5, è stata svolta solo in parte, per motivi di tempo e a causa di problemi nella realizzazione che hanno complicato notevolmente il lavoro. In particolare, si è cercato di dare soluzione al primo e fondamentale problema posto dal progetto: rendere ActionWorkflow System un sistema virtualmente distribuito.

6.2.1 Distribuzione di ActionWorkflow System

Prove preliminari

Le prime prove effettuate sul sistema, per avere un'idea della possibilità o meno di approntare un'implementazione fisica che riflettesse il progetto sviluppato, sono state effettuate su un'installazione standard. In particolare, si è andati a modificare opportune tabelle di *aws*, cercando poi di vedere la "reazione" del Process Manager. Questo primo approccio ha dato esiti positivi, in quanto il Process Manager riconosceva le modifiche effettuate, comportandosi come se fossero avvenute in modo "normale" (cioè come se fossero state effettuate dall'Administrator o da un'applicazione). Questa semplice prova, seppur in apparenza banale, risulta invece di grande importanza, in quanto nel sistema virtualmente distribuito, le tabelle nei siti periferici vengono aggiornate via replica, ma questi le devono "vedere" come avvenute direttamente in loco, attraverso l'Administrator o l'applicazione client utilizzata come interfaccia. I risultati ottenuti in prima istanza, quindi, ci hanno permesso di proseguire nel lavoro.

Replica di *aws*

Accertata la possibilità di agire sul sistema forzandolo a gestire informazioni ricevute in modo non usuale (direttamente nel database di servizio), il passo seguente prevedeva la replica, attraverso il sistema dedicato di SQL Server, di *aws*. È stato questo l'ambito in cui si sono incontrati i problemi più grossi, e che ha assorbito la rimanente parte dell'attività di implementazione di questa tesi.

Problema delle chiavi primarie Il primo problema in cui ci siamo imbattuti, e da cui sono nati gli altri, è dipeso dai requisiti necessari per la replica di una tabella. SQL Server, infatti, permette la replica solo di tabelle dotate di una chiave primaria esplicitamente dichiarata. Ciò significa che indici "unique", o chiavi costruite implicitamente utilizzando contatori unidirezionali per generare degli ID unici, non vengono riconosciuti come validi

al fine della replica, se non quando sono dichiarati con una clausola "CREATE CONSTRAINT PRIMARY KEY". In più, la chiave primaria (i campi che la compongono), devono sempre essere presenti in qualsiasi articolo si costruisca, questo per permettere al sistema di identificare correttamente le tuple da replicare o da aggiornare via replica.
Questi requisiti, in genere soddisfatti da qualsiasi database costruito a "regola d'arte", non lo sono da *aws*, le cui tabelle sono tutte prive di una chiave primaria esplicita. In questo database, per motivi pratici di implementazione, l'integrità dei dati viene mantenuta in altri modi. Per le chiavi primarie sono utilizzati degli **indici unique**, per quelle esterne dei trigger opportuni. Inoltre, tutti gli ID (di processo, di istanza, di ruolo, etc.), vengono generati in modo univoco mediante il meccanismo della chiave surrogata: apposite tabelle tengono memoria dell'ultimo valore utilizzato come ID; al momento di generarne uno nuovo, semplicemente si incrementa di una unità quel valore. In questo modo, pur non riempiendo i buchi in caso di cancellazioni, si garantisce che gli ID vengano sempre generati in modo corretto. È importante notare ancora che la scelta di non inserire le chiavi primarie nelle tabelle, ma usare indici unici, risponde ad esigenze implementative. Alcune tabelle, infatti, hanno indici composti anche da campi che ammettono valori "NULL": questi non potrebbero far parte di una chiave, ma servono ugualmente al sistema per identificare una tupla. Non tutte le tabelle hanno queste caratteristiche, ma probabilmente la scelta di Action Technology è andata verso l'uniformità dei costrutti del database. Notiamo infine che esistono alcune tabelle (cinque in tutto), formate solo da campi che ammettono i valori "NULL": queste non potrebbero ammettere nessun tipo di chiave primaria.

Le soluzioni L'idea di base è di dotare tutte le tabelle di *aws* di una chiave primaria che permetta di configurare la replica, ma non dia problemi di funzionamento al programma.
La prima soluzione al problema delle chiavi è stata provata non appena ci si è accorti della mancanza delle stesse nelle tabelle, senza aver ancora investigato a fondo le caratteristiche di tutte le tabelle di *aws*.
La soluzione più semplice ed immediata in un caso come questo, è quella di generare delle potenziali *super-chiavi*, creando chiavi che comprendano tutti i campi delle tabelle in questione, o almeno tutti i campi che non ammettono i valori "NULL".
Per operare in questo modo, si è scelto innanzitutto di sfruttare le nuove funzionalità del DOM Builder, di lettura della struttura delle tabelle di un database e di generazione di trigger per modificarla. In particolare, si è

aggiunta al programma la capacità di produrre script di modifica delle tabelle, per generare le chiavi richieste in modo semplice, sicuro e rapido (ricordiamo che le tabelle di *aws* sono 129). Da notare che le chiavi sono state definite come "UNCLUSTERED", in quanto gli indici presenti nelle tabelle sono di tipo "CLUSTERED", e ogni tabella ammette un solo indice di questo tipo.
Questa soluzione ha messo subito in evidenza il problema delle tabelle prive di campi "NOT NULL", che tuttavia in prima istanza è stato trascurato. Si è scelto di provare, infatti, almeno all'inizio, ad osservare il funzionamento del programma con le modifiche effettuate, configurando anche una prima forma di replica unidirezionale fra due server, rimandando la soluzione dell'altro problema.
Il programma ha funzionato correttamente, e con esso la replica, per le prime operazioni di configurazione: inserimento di ruoli e di identità. Le operazioni effettuate sul server publisher, infatti, sono state replicate sul subscriber, col risultato che entrambi i sistemi di workflow erano sincronizzati e riconoscevano gli stessi dati inseriti. Quando si è provato ad installare delle definizioni di processo (mappe), però, il sistema ha abortito l'operazione per violazione di chiave su una tabella. Studiando la struttura della stessa, si è evidenziata la differenza fra l'indice unique presente e la chiave da noi impostata: il primo aveva un campo in più, che però non era entrato a far parte della chiave perchè ammetteva i NULL. Per verificare l'entità del problema, si è provato a modificare opportunamente la tabella "incriminata", ma il sistema si è bloccato su una tabella seguente, per problemi analoghi. Da ciò la scelta di procedere in modo diverso, anche perchè le modifiche alle tabelle potevano avere impatto negativo sul sistema (si noti che per correggere la tabella di cui sopra, si è portato un campo da "nullable" a "not nullable")
L'idea nuova è stata di suddividere le tabelle di *aws* in gruppi, da trattare diversamente a seconda delle caratteristiche. In particolare, è nuovo il modo di gestire le chiavi per le tabelle che hanno dato problemi (comprese quelle che hanno tutti i campi "NULL"). La nuova soluzione proposta, di complessità crescente, ha previsto ulteriori modifiche al programma *Wgen*, (che ricordiamo essere l'eseguibile del DOM Builder), per renderlo in grado di identificare, via ODBC, gli indici presenti in una tabella.
Vediamo in breve in cosa consiste la soluzione, che è fondamentalmente quella definitiva adottata nel sistema.

1. Si leggono da programma (DOM Builder), tutte le tabelle di *aws*, memorizzando le informazioni relative a indici e "nullabilità" dei campi.

    - Per tutte le tabelle che hanno indice unique composto di soli campi che non ammettono i "NULL", si genera una chiave primaria di tipo "UNCLUSTERED" con gli stessi campi.

- Per tutte le tabelle prive di indici unique e composte di soli campi che non ammettono i "NULL", si genera una chiave primaria di tipo "UNCLUSTERED" con tutti i campi della tabella stessa (super-chiave).
- Per tutte le tabelle dotate di indici unique, ma con campi che ammettono i "NULL" all'interno di questi, non si genera in prima istanza nessuna chiave primaria.
- Per tutte le tabelle composte di soli campi che ammettono i "NULL", non si genera in prima istanza nessuna chiave primaria.
- Per tutte le tabelle che non rientrano nei casi considerati (e per le quali sarebbe problematico creare chiavi primarie adeguate), non si genera in prima istanza nessuna chiave primaria.

Al termine di questa prima fase avremo uno script che contiene gli statement di modifica delle tabelle per cui creiamo una chiave primaria, che risultano essere circa la metà del totale delle tabelle presenti in *aws*. Per queste tabelle, una volta modificate con l'aggiunta della primary key, è possibile configurare la replica direttamente verso l'altro *aws* sul server subscriber.

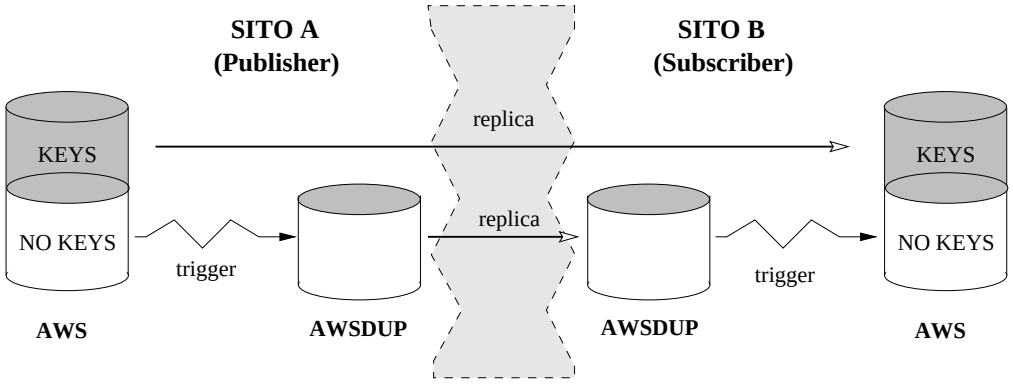


Figura 6.2: Replica unidirezionale di *aws*

2. Per le tabelle rimanenti si opera creando in tutti i siti un database di copia, chiamato *awsdup*, nel quale si inseriscono le stesse tabelle con l'aggiunta, per ognuna, di un campo chiave chiamato *ID_NOMETABELLA*. Questo campo è un "integer" per il quale adottiamo la tecnica della chiave surrogata già vista ed utilizzata in *aws*. Per l'implementazione di questa caratteristica si utilizza una tabella aggiuntiva chiamata "SURR_KEY" con un campo per ogni tabella presente in *awsdup*. Per mezzo di trigger da inserire sulle tabelle di *aws*, poi, si invia qualsiasi

inserimento, modifica, cancellazione di dati alle corrispondenti tabelle del database *awsdup* posto sullo stesso sito. In esso si configura la replica verso il suo gemello (pensiamo ancora alla situazione semplice di solo due server e, per ora, di replica unidirezionale), sull'altro sito. In questo caso (subscriber), i trigger vengono posti su *awsdup*, per portare i dati che arrivano dalla replica verso *aws*. La situazione descritta è illustrata in Fig 6.2. Notiamo che gli script per la generazione delle tabelle modificate, dei trigger adeguati, della tabella per la gestione delle chiavi, vengono tutti creati in automatico da programma (esempi degli stessi sono visibili in Appendice C).

La soluzione proposta, pur funzionando, presenta alcuni problemi.

- L'inserimento dei trigger sull'*aws* del publisher comporta la modifica sensibile dell'intero database. Questa modifica, però, non può essere effettuata in un'unica soluzione, eseguendo cioè un unico script di creazione dei trigger, per la presenza di trigger proprietari su alcune tabelle. Ricordiamo che ogni tabella ammette tre tipi di trigger, su insert, delete, update, ma solo un trigger per ogni tipo. Ciò comporta che i nostri trigger, se creati da un unico script incondizionatamente, andrebbero a sovrascrivere quelli già presenti, alterando in modo non desiderabile il comportamento del database e del programma che su di esso si basa.

- La replica così configurata, non è ancora adatta al sistema che vogliamo implementare, in quanto in esso si richiede la replica bidirezionale fra il server centrale (Master), e quelli periferici. Vedremo in seguito i problemi che questa estensione comporta al meccanismo utilizzato per permettere la replica di *aws*.

Problema dei trigger Il primo problema, purtroppo, è risolubile solo in parte all'interno del nostro sistema. In particolare, è possibile "a mano", effettuare un copia-incolla "intelligente" dei nostri trigger su quelli già presenti in *aws*: in questo modo il sistema prima esegue i controlli che gli sono propri, poi, in caso di inserimento, modifica, cancellazione corrette, invia i dati ad *awsdup*. Tuttavia, non è possibile, almeno via ODBC, nè sapere se una tabella presenta dei trigger, nè tantomeno intervenire sul codice degli stessi. Per questo motivo si è scelto di generare un file per ogni tabella con i relativi trigger; l'utente, al momento dell'esecuzione, si incaricherà di controllare se possono essere eseguiti così o se necessitano di un intervento più complesso.

Problema della replica bidirezionale Il secondo problema è ben più grave, perchè inserendo i trigger così come sono su entrambi i database *aws* e sulle loro copie parziali *awsdup*, nascono dei conflitti di trasferimento dei dati. Il trigger su *aws*, infatti, è progettato in modo da inoltrare qualsiasi dato inserito o modificato o cancellato, all'*awsdup* presente sullo stesso sito; quello su *awsdup*, per contro, esegue le stesse funzioni ma verso *aws*. In questa situazione, dunque, un dato inserito in *aws* al sito A, verrebbe inviato dal trigger all'*awsdup* sullo stesso sito, che però provvederebbe a rimandarlo indietro a causa del proprio trigger su insert. Ciò che accade, allora, è un "palleggio" di dati fra i due database che porta a generare degli errori (come verificato nella pratica).
Altro problema che nasce in questo caso, è sulla gestione delle chiavi primarie nelle tabelle duplicate: se prima queste erano generate in un solo sito, ora entrambe gli *awsdup* contengono una propria tabella "SURR_KEY". Ciò comporta la possibiltà non remota di duplicazione di chiavi dovuta alla replica dei dati, e al conseguente scatenarsi di errori che bloccano il processo. Vediamo come si è riusciti a risolvere entrambi i problemi.
Il primo problema si riconduce alla modifica dei trigger perchè possano accorgersi dell'origine dei dati che li hanno fatti scattare. Consideriamo per semplicità l'esempio dell'inserimento di dati. In particolare, i trigger su *aws* devono distinguere se l'insert avviene da programma, e in tal caso devono inoltrare i dati alle tabelle di *awsdup*, oppure da replica, e in tal caso non devono fare nulla. Analogamente, i trigger su *awsdup* devono distinguere gli insert da replica, per i quali devono copiare i dati sull'*aws* associato, da quelli che arrivano da *aws*, per i quali non devono fare nulla (ci pensa il meccanismo di replica ad inoltrarli all'altro *awsdup* nell'altro sito). La configurazione della replica bidirezionale di *aws* è mostrata in Fig 6.3.

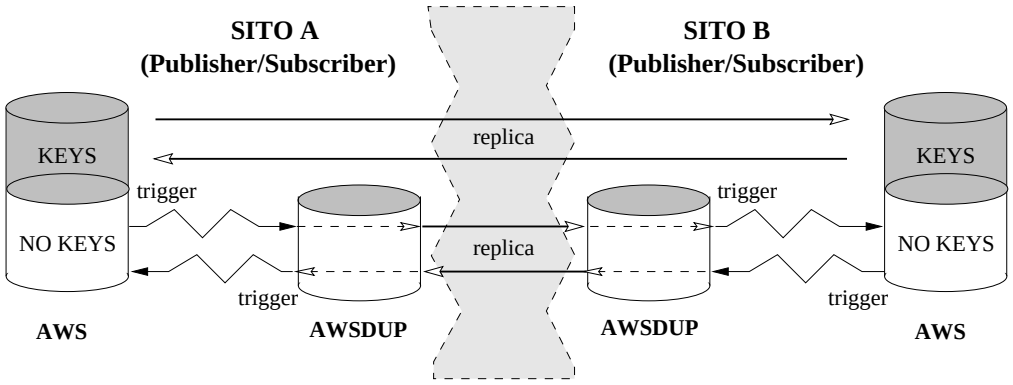


Figura 6.3: Replica bidirezionale di *aws*

La prima ipotesi di soluzione prevedeva l'utilizzo di un "semaforo", sotto

forma di flag in una tabella particolare dei due *awsdup*, che, opportunamente controllato e modificato, avrebbe dovuto regolare il traffico di dati nel modo voluto. Se in linea teorica tale soluzione funziona, nella pratica, poichè modifiche ad un dato in una tabella all'interno di più transazioni innestate non vengono viste, il meccanismo fallisce.
Per risolvere il problema, quindi, si è utilizzata una variabile globale di SQL Server, chiamata "@@NESTLEVEL", che conta il numero di transazioni innestate, e, partendo dal default "0", viene incrementata di una unità ogni volta che una nuova transazione inizia all'interno di un'altra. Nel nostro caso specifico, la conoscenza del livello di "nesting" delle transazioni è sufficiente a comprendere la direzione dei dati e ad implementare quindi correttamente la replica di *aws*.
Si supponga, per esemplificare la soluzione, di porre sui due *aws* un trigger (consideriamo il caso dell'insert), con questa struttura:

```
create trigger my_trig_aws on <tabella> for insert as
begin
  if @@NESTLEVEL = 1
  begin
    <sposta dati su awsdup>
  end
end
```

Sui due *awsdup*, analogamente, poniamo un trigger di questo tipo:

```
create trigger my_trig_awsdup on <tabella> for insert as
begin
  if @@NESTLEVEL = 1
  begin
    <sposta dati su aws>
  end
end
```

Supponiamo ora che avvenga un inserimento di un dato al sito A da programma (ad esempio in seguito all'esecuzione di un atto di un'istanza di processo), e vediamo cosa accade (per una migliore comprensione si veda anche la Fig 6.4). Questo inserimento è una transazione e il suo livello di "nesting" è per default uguale a zero. L'insert, però, fa scattare il trigger sulla tabella interessata di *aws*, e poichè il trigger è una transazione che inizia all'interno di un'altra, la variabile "@@NESTLEVEL" passa al valore "1". Se il trigger su insert di *aws* è del tipo descritto sopra, otteniamo il comportamento voluto: i dati vengono spostati su *awsdup* per essere poi trasferiti sull'altro sito.

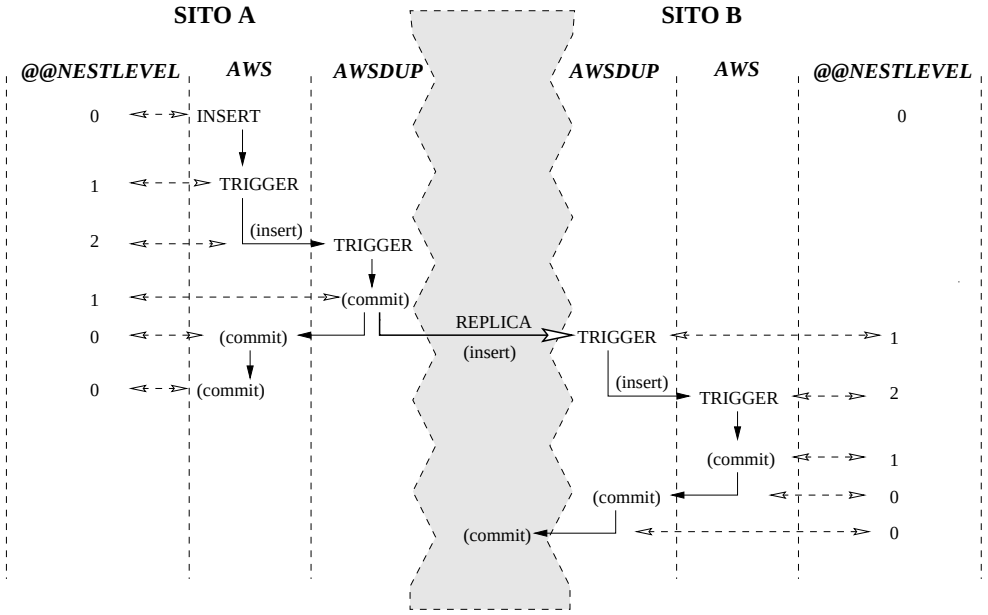


Figura 6.4: Il funzionamento dei trigger per la replica di *aws*

Su *awsdup*, l'insert dovuto ai dati che arrivano da *aws*, scatena il trigger, che però, iniziando all'interno del precedente, incrementa "@@NESTLEVEL" ponendola pari a "2". Il controllo allora fallisce, e il trigger su *awsdup* non fa nulla. Anche in questo caso il comportamento è corretto, perchè i dati che arrivano da *aws* vengono inseriti su *awsdup* e andranno poi spediti via replica all'altro *awsdup*.
Supponiamo poi, come detto sopra, di impostare gli stessi controlli anche sul sito B e vediamo cosa accade. Con il trigger su *awsdup* del sito A, si conclude la transazione iniziata con l'inserimento dei dati da programma e la variabile globale viene reinizializzata. A questo punto, avendo configurato la replica per scattare ad ogni transazione, parte il processo di replica dei dati da A verso B. Questa è una nuova transazione, non innestata[1], quindi quando i dati arrivano in *awsdup* del sito B, il trigger è il primo livello di "nesting" e inoltra correttamente i dati verso *aws*. Qui scatta ancora il trigger su insert, ma trovando il livello di "nesting" maggiore di "1", non fa nulla e i dati vengono correttamente inseriti su *aws* del sito B sincronizzando i due database.
La soluzione così implementata sembra funzionare, anche se mancano prove approfondite, soprattutto su inserimenti concorrenti da entrambi i siti.

[1] Osserviamo che, se anche la transazione di replica fosse innestata nelle precedenti, il funzionamento del sistema dimostra che per il sito B essa viene vista come una nuova transazione, quindi il livello di "nesting" locale in B è comunque "0".

In realtà, comunque, il controllo su livello di "nesting" delle transazioni dovrebbe garantire un corretto funzionamento anche in quel caso.
Rimane da considerare ancora il problema della duplicazione delle chiavi nelle tabelle dei due database *awsdup*. Per risolverlo, si è pensato di aggiornare il valore nel campo opportuno della tabella delle chiavi surrogate, non solo, come avveniva prima, quando un dato viene inserito da *aws* in *awsdup*, ma anche quando i dati arrivano in quest'ultimo da replica. In questo modo, si è garantita la correttezza degli insert anche a questo riguardo.
Una nota importante, poi, riguarda i cursori utilizzati nei trigger. Poichè i trigger stessi sono innestati, si ha la situazione in cui più cursori dichiarati da trigger diversi, sono contemporaneamente aperti. Questa situazione può generare problemi nell'esecuzione delle transazioni, portandole a fallire, se i cursori sono dichiarati con gli stessi nomi. Nella generazione da programma degli script contenenti i trigger, quindi, è necessario caratterizzare i cursori stessi con nomi univoci, facendo riferimento ad esempio al nome della tabella, e, per sicurezza, al nome del server su cui risiedono.
Per concludere la trattazione, dobbiamo dire che il lavoro di ricerca si è fermato a questo punto, lasciando aperti i problemi relativi ai passi che nel Capitolo 5 seguivano la configurazione della replica di *aws*. In particolare, è importante tracciare qui la strada da seguire per gli sviluppi futuri del progetto.

6.2.2 Sviluppi futuri

L'implementazione del progetto, sul piano pratico, non si è potuta spingere al completamento delle operazioni necessarie a fornire un sistema completo e funzionante. La configurazione della replica di *aws*, però, è una conquista importante all'interno del lavoro, perchè pone delle basi indispensabili per quanto si dovrà fare in seguito. In particolare, negli sviluppi futuri del lavoro, i punti importanti da trattare saranno due:

1. Configurazione delle partizioni delle tabelle di *aws*;

2. Specificazione e implementazione in automatico del collegamento fra Process Builder e DOM Builder ("mapping routine").

Configurazione delle partizioni delle tabelle di *aws* Il primo punto è strettamente collegato a quanto fatto in questa tesi, in quanto è il passo da intraprendere subito dopo aver abilitato alla replica il database di servizio di ActionWorkflow. In particolare, in esso si dovranno affrontare due problemi base:

- abilitazione dei soli ruoli corretti sui vari siti;

- creazione degli articoli per i server in base ai ruoli abilitati su di essi.

Per il primo punto si propone di operare nel modo seguente. Si inseriscono tutti i ruoli dal Master Server e li si replica verso tutti i server periferici. In questo modo il sistema genera degli ID di ruolo univoci per tutta la rete, e ciò evita conflitti sulle repliche seguenti. D'altra parte, così facendo, si portano tutti i ruoli su tutti i siti e ciò contrasta con l'idea di mappaggio ruolo-sito definita nel DOM. A questo punto, dunque, si eliminano dai siti locali tutti i ruoli che non vengono associati ad essi, lasciandovi solo quelli che, secondo la scelte di progetto, devono risiedervi. Quest'operazione può essere fatta da programma (Process Manager locali), o eventualmente automatizzata in modo che sia il DOM Builder stesso ad occuparsene, in accordo alle indicazioni fornite dall'utente. L'importante, però, è effettuare quest'operazione in modo che abbia effetto solo in locale, quindi quando ancora non è abilitata la replica dalla periferia verso il centro della rete.

Su questo problema si innesta poi la definizione degli utenti veri e propri del sistema, che vanno dichiarati come *identità* in ActionWorkflow. L'introduzione di nuove identità dal Process Manager comporta, in locale l'aggiunta dei corrispondenti login ad SQL Server, mentre questo non accade in remoto, perchè la replica riguarda solo i dati. Introducendo quindi le identità tutte dal Master Server, esse verranno replicate su tutti i siti, ma quelle riconosciute dai vari sistemi ActionWorkflow nei siti periferici non avranno un corrispettivo in forma di login nel server locale. In questo modo, come prima, si garantisce l'univocità degli ID delle identità, evitando a monte confusioni nella replica dei dati. Sempre come prima, poi, anche in questo caso si possono eliminare le identità relative a ruoli non residenti sul server, mentre per quelle effettivamente abilitate ad operare in locale risulterà necessaria l'aggiunta del login corretto ad SQL Server (da effettuarsi ancora manualmente o da DOM Builder). Si noti che, per motivi di coerenza e integrità referenziale fra le tabelle di *aws*, è necessario cancellare prima le identità superflue, poi i ruoli non abilitati in locale.

Una volta configurata questa parte del sistema, ci si può occupare della partizione delle tabelle di *aws*. Il problema che si presenta in questo caso, è complesso, soprattutto per il numero elevato di tabelle presenti nel database. In linea teorica, comunque, la partizione, che sarà di tipo orizzontale, si basa sui ruoli definiti nei vari siti, così come nel DOM. Per una trattazione precisa, però, è fondamentale analizzare a fondo le singole tabelle di *aws*, studiandone il "comportamento" sotto l'azione di un'istanza di processo.

Collegamento fra Process Builder e DOM Builder Al momento attuale, il collegamento fra i due sistemi è gestito in manuale, nel senso che la configurazione effettuata sul Process Builder deve essere ripresa e reimmessa nel DOM Builder, all'atto della costruzione del DST. In particolare, deve essere ripetuta la definizione dei ruoli e la creazione del DST deve essere effettuata "a mano". Nel futuro, l'idea è di automatizzare il più possibile il collegamento, facilitando il compito dell'utente nella configurazione di un processo di workflow.

Conclusioni

In questa tesi è stato progettato, e in parte realizzato, un supporto alla gestione di dati distribuiti per applicazioni di workflow mediante tecnologia DDBMS. In particolare, nel progetto si è cercato di potenziare le funzionalità di un sistema di workflow client-server, rendendolo virtualmente distribuito e affiancandolo ad un modello di replica che si appoggia ad un DDBMS, per ottenere una più corretta e potente gestione dei flussi di dati.

Il lavoro svolto si è articolato in diverse fasi prima di giungere al progetto vero e proprio. All'inizio, infatti, si è cercato di entrare nel mondo del workflow comprendendone vantaggi e limiti e prendendo anche visione diretta di un prodotto completo. In particolare, poi, ci si è concentrati sulla fase di modellazione dei processi aziendali, mostrando approcci alternativi ed evidenziando pregi e difetti di ognuno. Si è poi studiato il modello di replica Dynamic Ownership alla luce delle conoscenze acquisite riguardo i sistemi di workflow. Questa parte del lavoro ha portato a proporre delle modifiche al modello e una serie di osservazioni e confronti fra i due approcci.

Da queste osservazioni è nata l'idea su cui si è sviluppato poi il progetto presentato in questa tesi: integrazione fra sistemi di workflow e tecnologia DDBMS come mezzo per sfruttare al meglio le caratteristiche di entrambi. Dei sistemi di workflow si è utilizzata la capacità di modellare e gestire i processi, dei DDBMS, e in particolare del modello di replica DOM, si sono sfruttati l'approccio data-oriented e le caratteristiche distribuite. Mettendoli assieme, si è cercato di creare un sistema di workflow adatto ad ambienti distribuiti e ad applicazioni in cui i dati sono il centro dei processi. Lo sforzo fatto rientra in un filone di ricerca ampio e dinamico riguardante il workflow, che cerca di fornire soluzioni ed evoluzioni per i problemi e le carenze dei sistemi attuali.

L'implementazione del sistema non è stata completata, ma si è realizzato un primo passo decisivo per il proseguimento del lavoro, mostrando la possibilità per ActionWorkflow, di essere utilizzato come sistema di workflow

virtualmente distribuito. In particolare, il DOM Builder è stato rivisto e corretto, garantendo migliori funzionalità per quanto riguarda il DOM, e aggiungendo poi nuove caratteristiche per gestire i primi passi di configurazione del sistema progettato.

In futuro, per il completamento dell'implementazione, è necessario concentrarsi sulla configurazione delle pubblicazioni del database *aws*, perchè si accordino con la struttura organizzativa del sistema e con le specifiche della definizione di processo. A questo punto, con adeguate interfacce per le applicazioni client, il sistema potrebbe già essere operativo, e si potrebbero studiare sul campo le sue prestazioni. In un secondo tempo, poi, è auspicabile la realizzazione di una migliore integrazione fra DOM Builder e Process Builder, per facilitare il lavoro del progettista automatizzando il più possibile l'interoperabilità fra i due sistemi.

Appendice A

Glossario dei termini del workflow

Questo testo è la traduzione del glossario fornito dalla Coalition [17] come riferimento per la terminologia utilizzata nell'ambito del workflow. In esso sono elencati e spiegati i termini di uso più comune, con una serie di sinonimi e alcuni esempi del loro uso. Come si potrà notare, in alcuni casi ho preferito mantenere il termine originale, più espressivo, piuttosto che sostituirlo con una traduzione poco felice. Per una visione sintetica delle relazioni fra i termini di base vedi Fig A.1.

- **WORKFLOW** (Flusso di Lavoro)
  - *Definizione:*
    L'automazione per intero o in parte, di un processo aziendale (Business Process) durante il quale vengono trasferiti fra più partecipanti documenti, informazioni o azioni da svolgere (task) in accordo ad un insieme di regole procedurali.
  - *Uso:*
    L'automazione di un processo e' definita all'interno di una Process Definition (vedi), che identifica le varie attività, le regole procedurali e i dati associati necessari al controllo e alla gestione del workflow durante l'esecuzione del processo.
    Durante la messa in atto del processo, possono essere attive molte istanze del processo stesso (dette anche workflow Case), ognuna con un proprio insieme di dati.
  - *Sinonimi*
    * workflow Management
    * workflow Computing

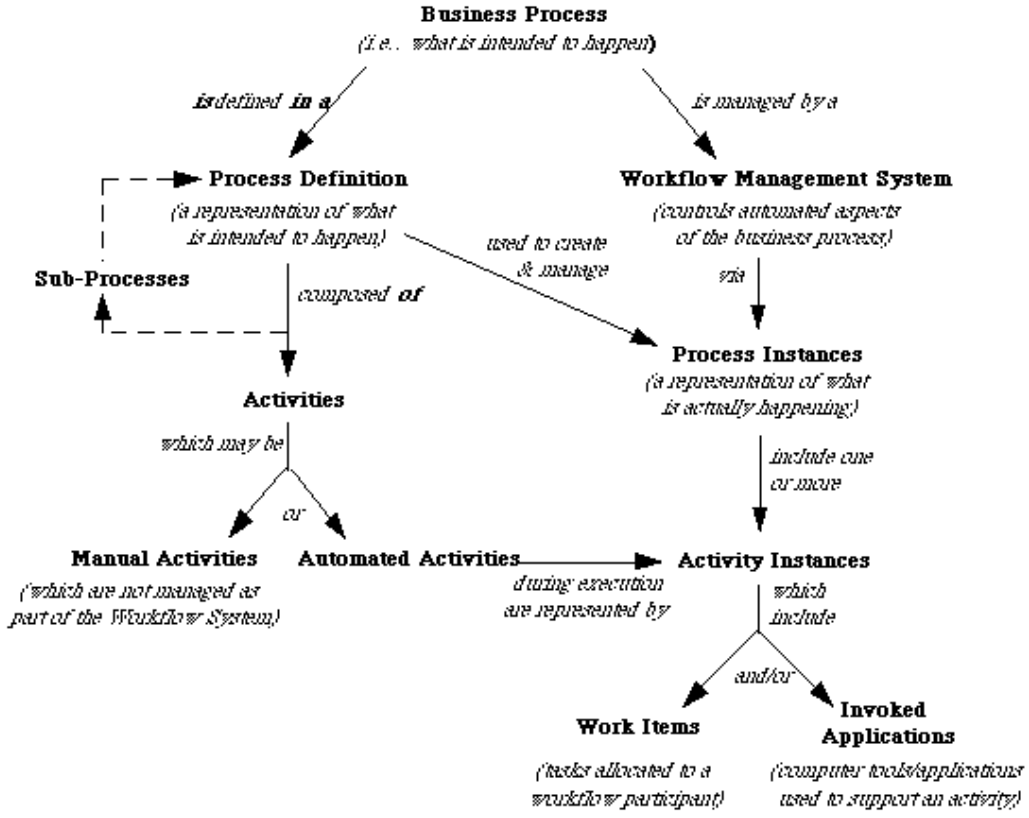


Figura A.1: I termini del workflow e le loro relazioni.

* Case Management

- **WORKFLOW MANAGEMENT SYSTEM** (Sistema di gestione di workflow abbrev. WfMS)

  – *Definizione:*

  Un sistema che definisce, crea e gestisce l'esecuzione di workflow attraverso l'uso di software capace di interpretare la definizione del processo, di interagire con i partecipanti al workflow e, dove richiesto, di invocare l'uso di applicazioni IT.

  – *Uso:*

  Un WfMS consiste di componenti software per memorizzare e interpretare le definizioni dei processi, creare e gestire istanze di workflow interagire con utenti e applicazioni.

Questi sistemi forniscono in genere anche funzioni amministrative e di supervisione, più l'audit dell'intero sitema o della singola funzione.

- *Sinonimi:*
  * Wf Automation
  * Wf Manager
  * Wf Computing System
  * Case Management

- **BUSINESS PROCESS** (Processo aziendale, abbrev. BP)
  - *Definizione:*

    Insieme di una o più procedure o attività collegate fra loro che collettivamente realizzano un obiettivo aziendale, di solito nel contesto di una struttura organizzativa in cui sono definiti ruoli funzionali e relazioni fra essi.
  - *Uso:*

    Un BP è associato tipicamente ad obiettivi operazionali e a relazioni d'affari e puo essere sviluppato all'interno di un'unica organizzazione o fra più organizzazioni come nel caso di una relazione fornitore-consumatore.

    Un BP ha delle condizioni definite per l'inizio di sue nuove istanze e outputs specificati al termine.

    Un BP può comportare interazioni formali o informali fra i partecipanti e la sua durata può variare ampiamente.

    Un BP può essere composto di attività automatizzate e/o manuali.
  - *Sinonimi:*
    * Process (colloquiale)

- **PROCESS DEFINITION** (definizione di processo, abbrev. PD)
  - *Definizione:*

    La rappresentazione di un processo aziendale in una forma che supporti la manipolazione automatizzata, come la modellazione o l'esecuzione da parte di un WfMS. La PD consiste di una rete di attività con le loro relazioni, dei criteri per determinare l'inizio e la fine del processo, delle informazioni riguardo le singole attività (come gli utenti interessati, le applicazioni invocate, etc.).

– *Uso:*

La PD è il risultato del lavoro svolto durante la fase di Process Definition Mode (vedi), e può includere attività sia manuali che automatizzate.

La PD può contenere riferimenti a sotto-processi definiti separatamente che fanno parte dell'intera PD.

– *Sinonimi:*

* Model Definition
* Routing Definition
* Flow Diagram
* State Transition Diagram
* Flow Schematic
* Workflow Script
* Instruction Sheet Definition
* Case Type

- **ACTIVITY** (Attività)

– *Definizione:*

La descrizione di una parte di lavoro che forma un'unità logica all'interno di una definizione di processo. Può essere manuale (priva di un supporto informatico), o automatizzata (detta anche di workflow). Un'attività di workflow richiede risorse umane e/o informatiche per supportare l'esecuzione del processo; quando è richiesta una risorsa umana un'attività viene allocata ad un Workflow Participant (vedi).

– *Uso:*

Una definizione di processo consiste di solito di parecchie attività fra loro logicamente correlate in termini del loro contributo alla completa realizzazione del processo aziendale.

Un'attività è tipicamente la più piccola unità di lavoro che viene considerata da un Workflow Engine (vedi), per lo scheduling di un processo.

Attività completamente manuali possono far parte di un processo aziendale ed essere incluse nella sua definizione, ma non fanno parte del workflow automatizzato che risulta dall'esecuzione del processo eseguita tramite computer.

Una attività può perciò essere catalogata come manuale o automatizzata, ma in generale nel nostro studio faremo sempre riferimento ad attività automatizzate.

Si definisce Attività Automatizzata una attività che può essere gestita tramite un'automazione al computer usando un WfMS. Durante l'esecuzione di un processo essa è gestita dal WfMS. In generale una attività automatizzata può essere una applicazione attivata direttamente dal WfMS, uno o più Work Item (vedi), assegnati ad un utente che partecipa al workflow e svolti all'interno di esso, uno o più Work Item assegnati ad un utente che partecipa al workflow e svolti all'esterno di esso comunicando solo i risultati alla fine.

Si definisce Attività Manuale una attività che non può essere automatizzata e perciò esce dagli scopi di un WfMS. Può entrare nella definizione di un processo, ma non può far parte del workflow risultante.

– *Sinonimi:*

* Step
* Node
* Task
* Work Element
* Process Element
* Operation
* Instruction

(automatizzata)
* Workflow Activity
* Activity (colloquiale)

(manuale)
* Non-automated Activity
* Manual Step
* Human Task
* Manual Work

- **INSTANCE** (istanza)

  – *Definizione:*

La rappresentazione di una singola esecuzione di un processo o attività con inclusi i dati ad esso associati. Ogni istanza rappresenta una esecuzione che puòessere controllata indipendentemente e che avrà il proprio stato interno ed un'identità visibile all'esterno, la quale potrà essere usata come chiave per poterne registrare e recuperare dati.

– *Uso:*

Un'istanza di processo o di attività è creata e gestita da un WfMS per ogni nuova invocazione del processo o dell'attività.

- **PROCESS INSTANCE** (istanza di processo, abbrev. PI)

  – *Definizione:*

  La rappresentazione di una singola messa in atto di un processo.

  – *Uso:*

  Una PI è creata, gestita ed eventualmente terminata da un WfMS, in accordo con la definizione del processo.

  È l'unità di lavoro rispetto un intero processo aziendale.

  Ogni PI mostra uno stato interno che rappresenta il suo procedere verso il completamento e il suo stato rispetto le attività che la costituiscono (vedi Process Status).

  – *Sinonimi:*

  * Process Definition Instance
  * Case
  * Workflow Definition Instance
  * Instruction Sheet Instance

- **ACTIVITY INSTANCE** (istanza di attività, abbrev. AI)

  – *Definizione:*

  La rappresentazione di un'attività all'interno di una istanza di processo.

  – *Uso:*

  Una AI è creata e gestita un WfMS, quando richiesto dall'esecuzione di un processo e in accordo con la definizione del processo stesso.

  Ogni AI rappresenta una singola invocazione di una attività, legata ad una sola istanza di processo di cui utilizza i dati. Ad una

sigola istanza di processo possono essere associate più AI (attività parallele), ma una AI può essere legata ad una ed una sola istanza di processo.

Ogni AI è di solito passibile di controllo e audit indipendente e mostra uno stato interno (vedi Activity State).

– *Sinonimi:*
  * Step Instance
  * Node Instance
  * Task Instance
  * Work Element Instance

- **WORKFLOW PARTICIPANT** (partecipante al workflow, abbrev. WfP)
  – *Definizione:*

    Una risorsa che esegue il lavoro rappresentato da un'istanza di un' attività di workflowQuesto lavoro è normalmente rappresentato da uno o più Work Items assegnati al WfP tramite una Worklist.

  – *Uso:*

    Normalmente questo termine è usato per indicare una risorsa umana, ma ha valenza più generale, anche se una risorsa automatizzata è normalmente detta Invoked Application.

    Un WfP può essere identificato direttamente nella definizione del processo, ma di solito viene legato ad un Ruolo il quale viene poi istanziato durante l'esecuzione del processo stesso.

  – *Sinonimi:*
    * Actor
    * Agent
    * Player
    * User (da noi utilizzato in seguito per le risorse umane)
    * Role Player
    * Work Performer

- **WORK ITEM** (elemento di lavoro, abbrev. WI)
  – *Definizione:*

    La rappresentazione del lavoro da svolgere (da parte di una risorsa), nel contesto di un'attività all'interno di un'istanza di processo.

- *Uso:*

  Un'attività genera di solito uno o più WI che insieme costituiscono il Task che l'utente deve eseguire in quella attività (se l'utente è una Invoked Application non c'è l'assegnazione di WI).

  I WI sono presentati all'utente mediante una lista (Worklist) che mantiene tutti i dettagli su di essi e un gestore di lista (Worklist Handler) che interagisce con l'utente stesso.

  Per protare a termine il proprio lavoro l'utente può o meno utilizzare applicazioni invocate.

- *Sinonimi:*

  * Work (ad ex. revisione di documenti, riempimento di questionari)
  * Work Object
  * Work Queue Item
  * Element
  * Work Pool Item
  * Task (da noi usato nel seguito)

- **WORKLIST** (lista di lavoro, abbrev. WL)

  - *Definizione:*

    Una lista di tasks associati ad un determinato utente o ad un gruppo di utenti.

  - *Uso:*

    Generalmente un gestore di WL richiede tasks ad un "motore" di workflow per creare tale lista; altre volte è il motore stesso a porre nella lista i tasks e solo dopo entra in gioco il gestore di lista.

  - *Sinonimi:*

    * Work Queue
    * In-Tray
    * To-Do List

- **WORKLIST HANDLER** (gestore di worklist, abbrev. WlH)

  - *Definizione:*

    Componente software che gestisce l'interazione fra l'utente e le worklist mantenute dal motore di workflowṘende possibile il passaggio dei tasks dal WfMS verso gli utenti e delle condizioni o stato di esecuzione degli stessi dagli utenti verso il WfMS.

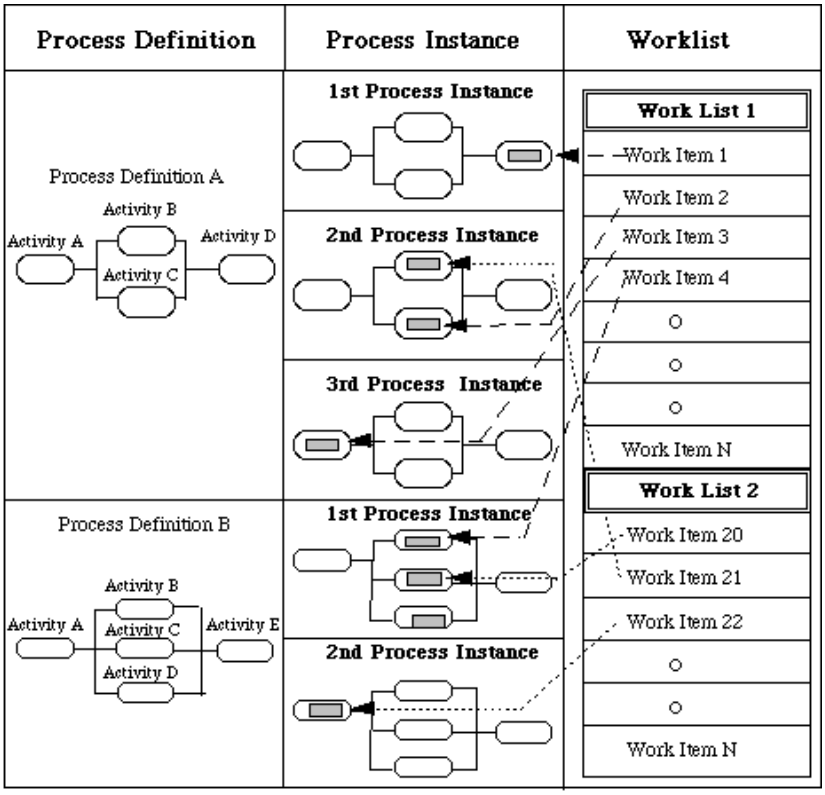


Figura A.2: Dalla definizione di processo alle worklist.

– *Uso:*

  Il WlH può far parte del pacchetto software di workflow o essere un'applicazione a parte. Può comunicare con diversi workflow per portare tutti i tasks su un'unica lista per l'utente.

  Possibili funzioni di un WlH: selezionare un task, riassegnare un task, notificare il completamento (terminazione) di un task, chiamare una applicazione come parte dell'esecuzione del task.

– *Sinonimi:*

  * WFM Front End
  * WFM Application
  * Workflow To-Do List Application
  * Task Manager
  * Active Work Performer

Per un esemplificazione grafica degli ultimi termini definiti vedi Fig A.2.

- **WORKFLOW ENGINE** (motore di workflow, abbrev. WfE)
  - *Definizione:*

    Un software che fornisce l'ambiente di esecuzione per una istanza di processo.
  - *Uso:*

    Un WfE fornisce diverse funzioni fra cui ricordiamo: interpretazione della definizione del processo;
    creazione di istanze di processi e gestione della loro esecuzione;
    navigazione attraverso le varie attività e creazione dei tasks più appropriati per eseguirle;
    funzioni di supervisione e gestione.
  - *Sinonimi:*

    * Case Processor
    * Workflow Enactment Service

- **PROCESS DEFINITION MODE**
  - *Definizione:*

    Il periodo di tempo in cui la descrizione manuale o automatizzata di un processo di workflow sono definite e/o modificate.
  - *Sinonimi:*

    * Process Modelling
    * Build Time

- **PROCESS** (processo)
  - *Definizione:*

    La visione formalizzata di un processo aziendale, rappresentato come un insieme coordinato (parallelo e/o sequenziale), di attività connesse fra loro per un fine comune.
  - *Uso:*

    Esempio di processo con cinque attività (vedi Fig A.3)
  - *Sinonimi:*

    * Activity Network
    * Directed Graph
    * Petri Net (Rete di Petri)

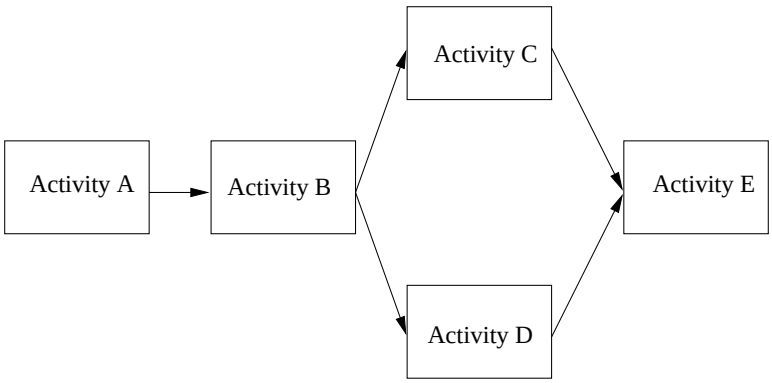


Figura A.3: Rapresentazione grafica di un processo.

* Model
* Instruction Sheet

- **SUB PROCESS** (sotto-processo)
  - *Definizione:*

    Un processo eseguito o chiamato da un altro processo del quale forma una parte. Possono essere supportati molti livelli di sotto-processi.
  - *Uso:*

    Un sotto-processo è utile per definire componenti riusabili.

    Un sotto-processo ha una sua definizione di processo.

    Più sotto-processi possono interagire in diversi modi durante l'esecuzione di un workflow (concatenati, innestati ...).
  - *Sinonimi:*

    * Subflow
    * Sub workflow

- **ACTIVITY BLOCK** (blocco di attività)
  - *Definizione:*

    Un insieme di attività all'interno di una definizione di processo, che condividono una o più proprietà comuni che fanno sì che certe azioni del sistema vengano intraprese nei confronti del blocco considerato come una unità sola.

  - *Sinonimi:*
    * Activity Set

- **DEADLINE** (capolinea)
  - *Definizione:*
    Un vincolo di tempo che richiede che una certa attività termini entro un ben determinato tempo (la deadline appunto).
  - *Sinonimi:*
    * Completion Time

- **PARALLEL ROUTING**
  - *Definizione:*
    Un segmento di un'istanza di processo in cui due o più attività sono in esecuzione parallelamente.
  - *Uso:*
    Vedi AND-Split
  - *Sinonimi:*
    * Concurrent Processing

- **SEQUENTIAL ROUTING**
  - *Definizione:*
    Un segmento di un'istanza di processo in cui più attività sono mandate in esecuzione in sequenza.
  - *Sinonimi:*
    * Serial Routing

- **AND-Split**
  - *Definizione:*
    Un punto del Wf in cui un singolo flusso di controllo si divide in due o più attività parallele.
  - *Uso:*
    Vedi Fig A.4.

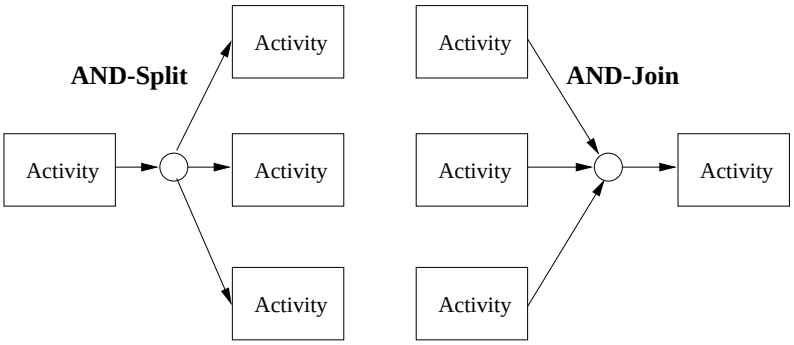


Figura A.4: Rapresentazione grafica dell'AND-Split e dell'AND-Join.

  - *Sinonimi:*
    * Split

- **AND-Join**

  - *Definizione:*

    Un punto del workflow in cui due o più attività parallele convergono in un unico flusso di controllo. La attività che segue il Join può iniziare solo quando tutte le attività parallele sono terminate.

  - *Uso:*

    Vedi Fig A.4.

  - *Sinonimi:*
    * Join
    * Rendezvous
    * Synchronisation point

- **OR-Split**

  - *Definizione:*

    Un punto del workflow in cui un singolo flusso di controllo si divide in due o più attività alternative. Il sistema decide quale strada (una sola), seguire, in base a condizioni poste sui vari rami.

  - *Uso*

    Vedi Fig A.5.

  - *Sinonimi:*

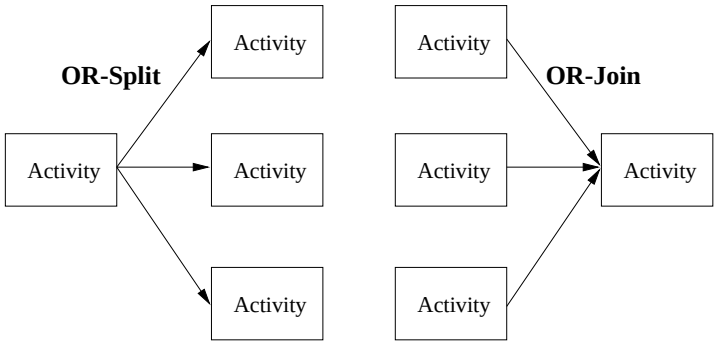


Figura A.5: Rapresentazione grafica dell'OR-Split e dell'OR-Join.

  * Conditional Branching
  * Switch

- **OR-Join**
  - *Definizione:*
    Un punto del workflow in cui due o più attività alternative riconvergono verso una singola attività comune. Non è richiesta sincronizzazione fra le attività, quindi quella che segue il join può iniziare non appena una delle precedenti è terminata.
  - *Uso:*
    Vedi Fig A.5
  - *Sinonimi:*
    * Join
    * A-Synchronous-Join

- **ITERATION**
  - *Definizione:*
    Un ciclo che coinvolge una o più attività e che termina quando viene verificata una condizione.
  - *Uso:*
    Vedi Fig A.6
  - *Sinonimi:*
    * Workflow Loop
    * While-Loop

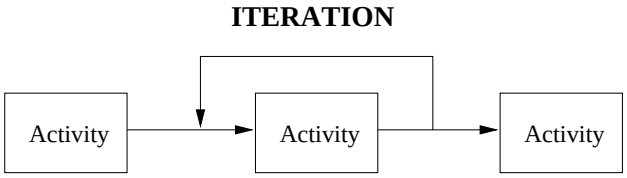


Figura A.6: Rappresentazione grafica di un ciclo.

- **CONDITION**
  - *Definizione:*

    Una espressione logica che può essere valutata da un motore di Wf per decidere su azioni alternative da intraprendere. In particolare si hanno:
    - *PRE-CONDITION*: per iniziare o meno un processo o un'attività.
    - *POST-CONDITION*: per terminare o controllare la terminazione di un processo o di un'attività.
    - *TRANSITION-CONDITION*: per decidere che strada seguire nell'esecuzione di un processo.
  - *Uso:*

    Le condizioni vengono definite all'interno della definizione del processo.

    A volte le condizioni sulle transizioni son espresse come insieme di Pre e Post condizioni.

    Le condizioni possono essere poste su eventi esterni, su variabili definite nel workflow, su dati correlati (ad ex di un Database), su variabili di sistema come data e ora.
  - *Sinonimi:*

    (Pre-Condition)
    * Entry Criteria
    * Activity Start Rules

    (Post-Condition)
    * Exit Criteria
    * Activity Completion Rules

    (Transition-Condition)
    * Navigation Rules
    * Transition Rules

  * Routing Condition

- **PROCESS STATE** (stato del processo)
  - *Definizione:*

    La rappresentazione delle condizioni interne che definiscono lo stato di una istanza di processo in un particolare momento. Queste informazioni fanno parte dei dati di controllo del WfMS.
  - *Uso:*

    L'esecuzione di un'istanza di processo comporta una serie di transizioni fra i vari stati in cui questa può trovarsi. L'insieme completo degli stati dà una descrizione/definizione del comportamento delle istanze del processo.

    La WfMC ha identificato un certo numero di stati che possono comporre un insieme completo (vedi per un'esemplificazione grafica la Fig A.7):
    * *INITIATED* l'istanza del processo è stata creata ma non è ancora in esecuzione.
    * *RUNNING* l'istanza ha iniziato l'esecuzione e una o più attività possono essere fatte partire.
    * *ACTIVE* una o più attività sono iniziate ed esistono già le loro istanze.
    * *SUSPENDED* l'istanza del processo è in attesa; nessuna ulteriore attività viene iniziata fino ad un "risveglio" della istanza.
    * *COMPLETED* l'istanza del processo ha raggiunto le condizioni di terminazione e sono in corso attività posteriori come il log dei dati.
    * *TERMINATED* l'esecuzione del processo è stata fermata in modo anomalo per errori o per una richiesta dell'utente.
    * *ARCHIVED* l'istanza del processo è stata posta in uno stato indefinito di archiviazione, ma può essere ripresa se si presenta la necessità.
  - *Sinonimi:*
    * Workflow State
    * Model State

- **ACTIVITY STATE** (stato dell'attività)

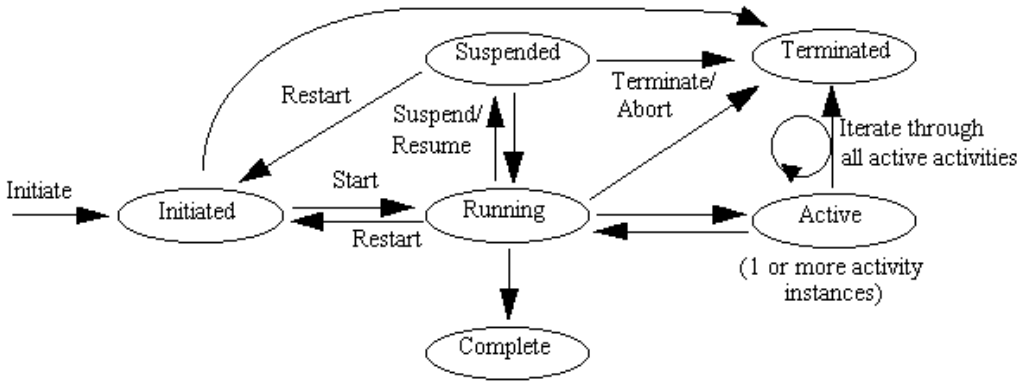


Figura A.7: Gli stati possibili dell'istanza di un processo

– *Definizione:*

  La rappresentazione delle condizioni interne che definiscono lo stato di una istanza di un'attività in un particolare momento. Queste informazioni fanno parte dei dati di controllo del WfMS.

– *Uso:*

  La WfMC ha identificato un certo numero di stati che possono comporre un insieme completo per le istanze di un'attività (vedi Fig A.8):

  * *INACTIVE* l'istanza è stata creata ma non è ancora in esecuzione e non esistono ancora tasks per essa.
  * *ACTIVE* sono stati creati ed assegnati uno o più tasks.
  * *SUSPENDED* l'istanza è in attesa e non possono essere creati ulteriori tasks finchè essa non viene riattivata.
  * *COMPLETED* l'istanza ha raggiunto le condizioni di terminazione e sono in atto le attività posteriori come il log dei dati.

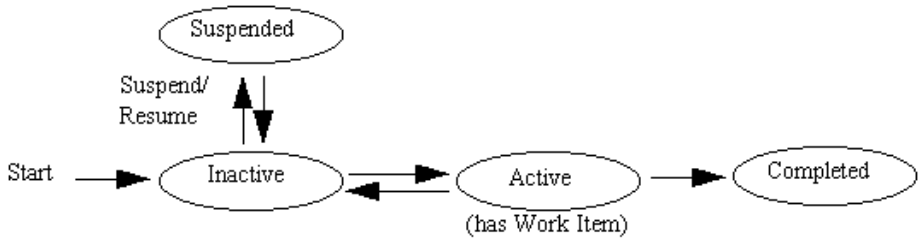


Figura A.8: Gli stati possibili dell'istanza di una attività.

- *Sinonimi:*
  * Step State

- **STATE TRANSITION** (transizione di stato)
  - *Definizione:*

    Un movimento da uno stato interno (di un'istanza di processo o di attività), ad un altro, che riflette un cambiamento di stato del workflow. Può avvenire in risposta ad un evento esterno, ad una decisione del motore di workflow ...
  - *Uso:*

    Una serie di transizioni di stato è ciò che avviene mano a mano che il workflow procede nella sua esecuzione. Tali transizioni possono essere registrate per poi verificare a posteriori come si è svolta l'esecuzione del processo.

- **ORGANISATIONAL ROLE** (Ruolo Organizzativo)
  - *Definizione:*

    Un gruppo di utenti caratterizzati da uno specifico insieme di attributi, caratteristiche, abilità, qualifiche.
  - *Uso:*

    Tipicamente un qualsiasi utente che ricopre un certo ruolo può eseguire un'attività che richieda una risorsa (umana in questo caso), con quell'insieme di attributi.
  - *Sinonimi:*
    * Role
    * User Groups
    * Organisational Groups

- **ORGANISATIONAL MODEL** (modello organizzativo)
  - *Definizione:*

    Un modello che rappresenta le entità di un'organizzazione e le loro relazioni; può anche contenere un isieme di attributi associati alle entità. Un tale modello dovrebbe essere realizzato tramite un database.

- *Uso:*

  Qusto modello comprende di solito concetti come gerarchia,autorità, responsabilità e attributi associati ai ruoli.

- *Sinonimi:*

  * Role Model
  * Organisational Directory

- **PROCESS ROLE**

  - *Definizione:*

    Un meccanismo con cui si associano gli utenti ad una serie di attività del Wf.

  - *Uso:*

    Il ruolo definisce il contesto nel quale l'utente partecipa ad un processo o ad una attività. Il ruolo implica concetti organizzativi come gerarchia o autorità, ma spesso riguarda altri attributi come abilità, ubicazione, data e ora, etc.

  - *Sinonimi:*

    * Role
    * Activity Group
    * Workflow Performer Definition

Appendice B

Concetti base dei DDBMS

B.1 Introduzione ai sistemi distribuiti

Un sistema distribuito è un insieme di sistemi di elaborazione (o processori), non necessariamente omogenei, interconnessi attraverso una rete, che cooperano nell'esecuzione di determinati compiti.
La distribuzione può essere relativa a:

- *Funzioni:* le diverse funzioni di un sistema possono essere delegate a diverse parti della struttura.
- *Dati:* i dati utilizzati dalle applicazioni possono essere distribuiti su diversi elaboratori appartenenti alla rete.
- *Controllo:* il controllo dell'esecuzione delle applicazioni può essere svolto da più parti della struttura e non solo da un sistema.

Una classificazione dei sistemi distribuiti può basarsi sui criteri seguenti:

- *Grado di accoppiamento:* è un indicatore del legame fra due sistemi di elaborazione e può essere misurato dal rapporto fra la quantità di dati scambiata e la quantità di elaborazione locale in un determinato task. I due livelli di accoppiamento cui si fa normalmente riferimento sono qualificati dal tipo di comunicazione fra le diverse entità della struttura:
  - *debole* se la comunicazione avviene attraverso una rete di computer;
  - *forte* se la comunicazione avviene tramite componenti condivisi (memoria o supporti magnetici).

- *Struttura di interconnessione:* è la "forma" in cui è realizzato il collegamento fra le unità elaborative e può essere di due tipi:
  - *point-to-point* se i sistemi sono collegati direttamente (ad esempio con una rete ad anello);
  - *canale di comunicazione comune* se i sistemi sono tutti collegati con un unico mezzo di comunicazione (ad esempio le reti a bus).
- *Indipendenza dei componenti:* le diverse unità elaborative possono essere fra loro indipendenti, o avere legami più o meno stretti. Nel primo caso, l'unico modo per scambiare informazioni è attraverso messaggi all'inizio e alla fine delle elaborazioni (rispettivamente contenenti i dati e i risultati).
- *Sincronizzazione fra i processi:* i processi possono essere sincroni o asincroni. La valutazione di questa caratteristica è fortemente legata ad altri fattori (processi sincroni implicano sistemi *dipendenti* e probabilmente accoppiati in modo *forte*).

B.2 Sistemi di gestione di basi di dati distribuite

Possiamo definire un *Distributed Database System* (DDBS) come un insieme di più database logicamente intercorrelati e distribuiti attraverso una rete di computer. Di conseguenza un *Distributed Database Management System* (DDBMS) è definito come uno strumento software che permette la gestione di un DDBS e *rende la distribuzione trasparente all' utente.*
Quindi il DDBS non è semplicemente un insieme di file, ma occorre che ci sia una struttura (gestita attraverso il DDBMS) che lega logicamente questi files. Anche il concetto di rete di computer va esteso: non si indica solo l' interconnessione di diversi computer geograficamente distanti tra loro (LAN o WAN), ma ogni sistema in cui lo scambio di dati non avviene attraverso una memoria condivisa ma attraverso una rete (eterogenea) sulla quale passano unicamente messaggi.
Tra i siti della rete non ne esiste uno che contiene tutto il database, infatti se così fosse la struttura sarebbe client/server: la gestione è centralizzata e riservata al server che contiene il database (figura B.1), al quale arrivano tutte le richieste dai siti periferici. Nell'ambiente distribuito ogni sito della rete possiede e mantiene una porzione del DDBS (figura B.2) che può rendere disponibile ad un altro sito che ne faccia richiesta.

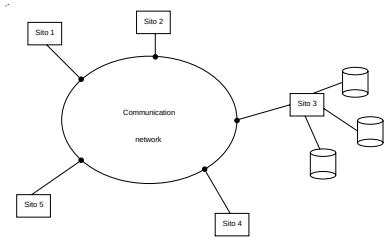


Figura B.1: Database centralizzato

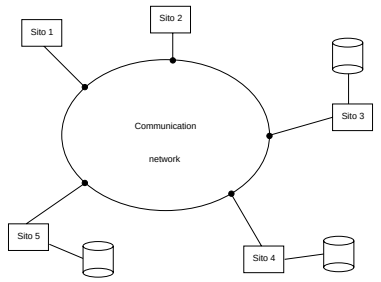


Figura B.2: Distributed Database System

Elenchiamo i vantaggi che la distribuzione dei dati e delle applicazioni possono dare:

Autonomia locale. La struttura distribuita permette al sito proprietario dei dati un controllo maggiore su di essi.

Performance. I dati più frequentemente utilizzati da un utente sono fisicamente più vicini ad esso, questo porta ad un aumento nelle prestazioni, inoltre ogni sito si occupa della gestione delle informazioni di sua pertinenza, perciò l' utilizzo delle risorse viene anch' esso distribuito tra i vari siti.

Robustezza. I dati sono replicati in più di un sito (tutti quelli che ne fanno richiesta), pertanto in caso di crash esiste sempre una copia dei dati che permette di ricostruire il database. In modo analogo nel caso di problemi sulla linea di comunicazione, è possibile trovare un percorso alternativo per recuperare i dati.

Economicità. Con i sistemi distribuiti è possibile portare i dati da elaborare sul server dell' utente, in questo modo dopo una iniziale fase di sincronizzazione dei database, le comunicazioni possono ridursi notevolmente.

Espandibilità. Espandere un database distribuito è molto semplice, infatti è sufficiente aggiungere siti alla rete, questi nuovi siti si occuperanno della gestione delle nuove funzioni aggiunte, gli altri siti non necessitano di modifiche.

Condivisibilità. La distribuzione dei dati permette una maggiore facilità nel reperimento delle informazioni, in particolare quando le distanze tra i siti sono elevate, in questi casi infatti la distribuzione rende possibile

la condivisione dei dati tra i diversi computer ad un costo minore e più semplicemente che nel caso centralizzato.

Osserviamo che i vantaggi elencati devono anche essere visti come obiettivi da raggiungere nello sviluppo di un DDBMS.

Naturalmente la distribuzione non porta solo vantaggi:

Mancanza d' esperienza. I DDBMS commerciali sono stati introdotti sul mercato solo recentemente, questo porta inevitabilmente delle difficoltà nella progettazione ed installazione di nuove applicazioni.

Complessità. Aumentando la complessità della struttura di gestione dei dati, aumenta anche il numero dei problemi nuovi da affrontare, molti dei quali ancora irrisolti.

Costi. A fronte dell' economicità nelle telecomunicazioni vi è un aumento dei costi per le risorse hardware (in parte compensato dalla continua diminuzione dei prezzi) e per il software di gestione sia delle reti che delle applicazioni distribuite.

Sicurezza. Con i sistemi distribuiti il controllo della sicurezza deve essere fatto in ogni sito, inoltre occorre rendere sicura anche la rete. Questo coinvolge quindi più aspetti del sistema, che con un sistema centralizzato erano svolti interamente dal DMBS centrale.

Difficoltà di cambiamento. Il passaggio da un sistema centralizzato (eventualmente già ben collaudato) ad un sistema distribuito, è molto dispendioso sia in termini di adeguamento delle risorse hardware e software che in risorse umane (gli amministratori di sistema). Attualmente non esistono metodi che aiutano a convertire un sistema centralizzato in uno distribuito.

Ci sono però altri aspetti che possono portare a malfunzionamenti, soprattutto legati al meccanismo di replica (che descriveremo nel dettaglio in seguito): le informazioni replicate su un sito, non sono aggiornate istantaneamente quando vengono modificate dal server che le possiede, questo provoca difficoltà nella gestione della sincronizzazione delle transazioni, inoltre le reti di comunicazione diventano un aspetto critico: se qualche malfunzionamento provoca l' interruzione delle comunicazioni è possibile che i siti che possiedono repliche dei dati non abbiano i dati validi. Questi sono problemi che devono essere risolti dal DDBMS e non si presentano nei sistemi centralizzati.

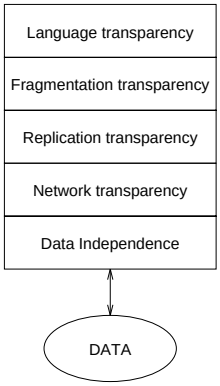


Figura B.3: Livelli di trasparenza

B.3 Livelli di trasparenza nei DDBMS

Uno degli obiettivi dei sistemi distribuiti è la trasparenza, obiettivo comune anche ai database tradizionali, ma che nel caso distribuito assume primaria importanza. E' indispensabile infatti nascondere all' utente tutti i dettagli dell' implementazione del sistema, che possono presentare una complessità di gestione notevole. L' utente deve poter utilizzare tutti i dati e le applicazioni necessarie alla propria funzione, indipendentemente dalla localizzazione fisica all' interno della rete delle informazioni.

In figura B.3 è rappresentato uno schema dei vari livelli di trasparenza, il nucleo centrale è rappresentato ovviamente dai dati, attorno a questo devono essere forniti gli adeguati supporti fino a permettere l' elaborazione dei dati attraverso linguaggi ad alto livello.

Data independence L' indipendenza dei dati può essere di due tipi:

- Logical data independence. Riguarda la possibilità per l' applicazione utente di rimanere immune ai cambiamenti nella struttura logica del database. Il subset di attributi delle relazioni utilizzate dall' applicazione deve rimanere disponibile anche dopo eventuali aggiunte di nuovi attributi.

- Physical data independence. La memorizzazione fisica dei dati (ad esempio il tipo di supporto utilizzato) non deve riguardare l' applicazione, questa deve continuare a funzionare anche nel caso di modifiche fisiche (ad esempio il trasferimento di un database da dischi magnetici a dischi ottici).

L' applicazione utente deve essere modificata solo se ci sono variazioni nelle operazioni da eseguire sui dati.

Network transparency Si deve garantire l' indipendenza dalla distribuzione dei dati, cioè l' applicazione non deve essere modificata se i dati non sono più centralizzati ma diventano distribuiti su una rete. Questo implica l' unicità dei nomi utilizzati, infatti le applicazioni li utilizzano come se fossero su un unico database.

Replication transparency Per ottenere miglioramenti nelle prestazioni, i dati remoti utilizzati frequentemente (in sola lettura) devono essere resi disponibili sul sito locale. Questa operazione viene definita *replica dei dati*, e su questo argomento torneremo diffusamente in seguito.

La replica non avviene in modo completamente trasparente all' utente, infatti sarebbe molto complesso per il DDBMS gestire completamente le repliche, si è scelto di lasciare all' utente la responsabilità di decidere se avere copie o no. Questo però implica una riduzione della indipendenza dei dati.

I DDBMS più recenti vanno nella direzione di aumentare la trasparenza anche della replica, questo va a scapito di una maggior complessità del setup iniziale del database.

Fragmentation transparency Ogni relazione di un database può essere suddivisa in più parti; questo permette di migliorare la gestione aumentando le prestazioni e facilitando la replica. Questa suddivisione implica un cambiamento nelle query immesse dall' utente, la trasparenza a questo livello permette di utilizzare tabelle frammentate con query globali, il passaggio da query globale a query frammentata è eseguita dal sistema.

Language transparency Gli utenti accedono ai dati utilizzando linguaggi ad alto livello, questi consentono di utilizzare i dati focalizzando unicamente gli oggetti di interesse e non i dettagli implementativi del database. Questo può avvenire utilizzando linguaggi 4GL, interfacce grafiche, linguaggi naturali, ...

B.4 Architettura dei DDBMS

Gli usuali DBMS relazionali sono basati sull' architettura ANSI/SPARC (figura B.4), strutturata in tre livelli di vista sui dati: *esterno*, *concettuale*, *interno*, rispettivamente orientati all' utente, all' azienda e al sistema. Per ognuna di queste viste serve l' appropriata definizione dello schema.

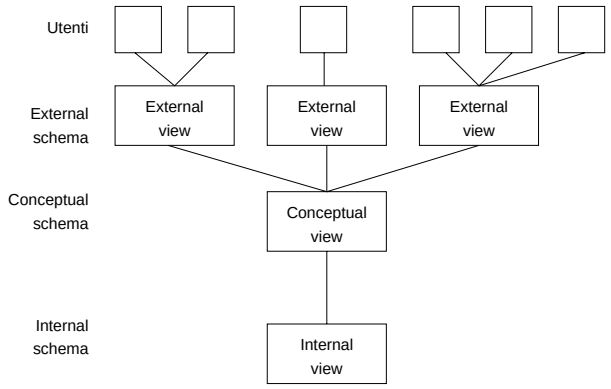


Figura B.4: Architettura ANSI/SPARC

La trasformazione tra questi livelli viene svolta attraverso un mapping che mette in relazione tra loro le definizioni dei diversi livelli.

La natura del modello ANSI/SPARC porta a ricercarne una estensione da applicare all' ambiente distribuito.

Iniziamo a descrivere l' architettura dei DDBMS dal punto di vista dell' organizzazione dei dati.

Ogni sito può avere una organizzazione fisica dei dati diversa dagli altri, questo impone di avere uno schema interno locale (*LIS*) che permette di definire in modo uniforme le diverse strutture fisiche. Per ogni sito è necessario avere anche uno schema concettuale locale (*LCS*) che permette la definizione dell' organizzazione logica dei dati presenti sul sito.

La vista globale di tutti questi schemi è detto schema concettuale globale (*GCS*), globale in quanto descrive la struttura dei dati di tutti i siti. L' applicazione utente accede ai dati attraverso uno schema esterno (*ES*). Infine il *GD/D* (global directory/dictionary) che permette di effettuare il mapping globale dei dati quando usiamo il GCS. Questa struttura è l' estensione dell' usuale concetto di dizionario dei dati definito per i database relazionali (architettura ANSI/SPARC), è un database che contiene schemi, mapping tra gli schemi, statistiche, controllo degli accessi, ecc . . . .

Questo modello (figura B.5) ha il pregio di soddisfare i requisiti di trasparenza descritti in precedenza, in particolare gli schemi concettuali globali e locali permettono di avere la trasparenza della replica e la frammentazione dei dati.

Possiamo ora esaminare i componenti di un generico sistema DDBMS. E' composto da due moduli (figura B.6):

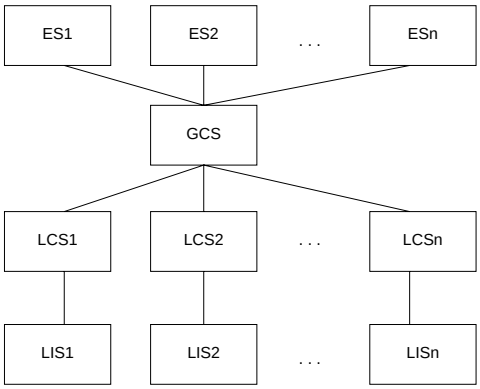


Figura B.5: DDBS reference

B.4.1 User processor

L' *user processor* si occupa dell' interazione con l' utente, analizziamo le sue procedure:

- *User interface handler.* E' l' interfaccia con l'utente: interpreta i comandi impartiti e formatta l' output.
- *Semantic data controller.* Applica i vincoli di integrità e controlla le autorizzazioni definite nel GCS per decidere se la query può essere eseguita.
- *Global query optimizer and decomposer.* Determina la strategia di esecuzione a minor costo e traduce le query globali in query locali utilizzando i GCS ed LCS eseguendo il mapping globale attraverso la GD/D.
- *Distributed execution monitor.* Coordina l' esecuzione distribuita dei comandi dell' utente.

B.4.2 Data processor

Il *data processor* si occupa della gestione e del controllo della memorizzazione e del recupero delle informazioni. E' composto da 3 componenti:

- *Local query optimizer* E' l'ottimizzatore locale della query, sceglie il miglior tipo di accesso (tramite indici) ai dati locali.
- *Local recovery manager* Si occupa di garantire che il database locale rimanga consistente anche in caso di guasto a qualche componente.

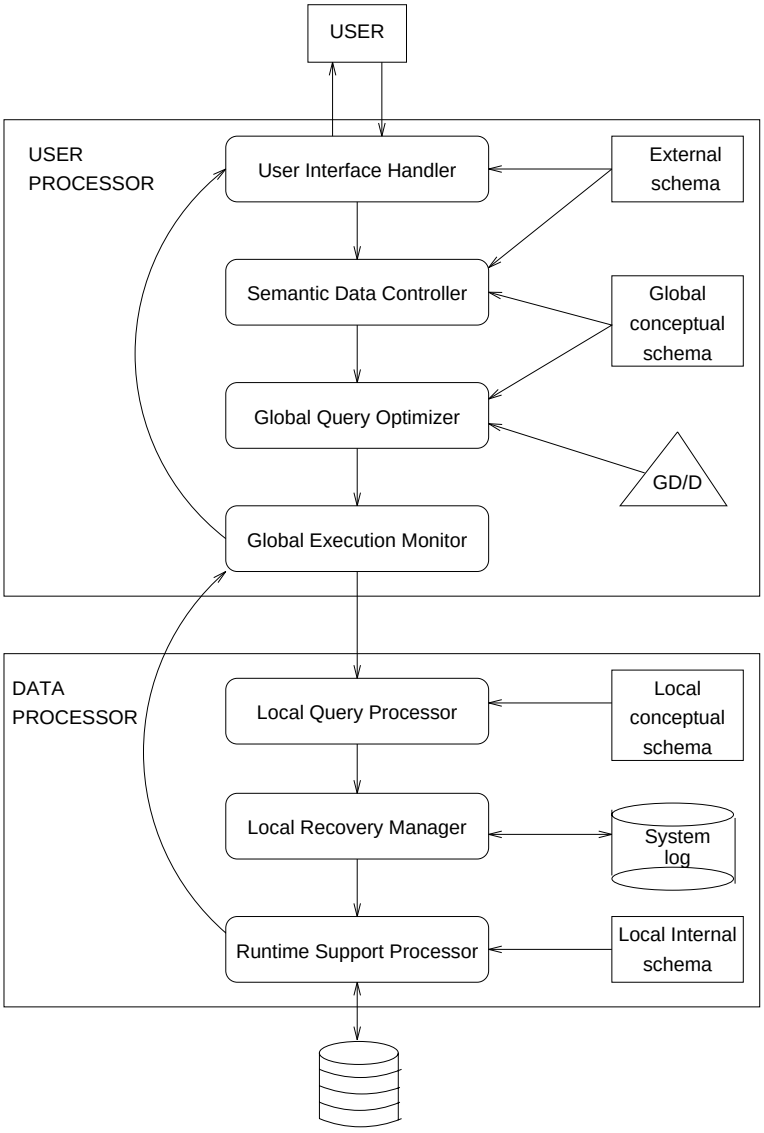


Figura B.6: Componenti di un DDBMS

- *Run-time support processor* Accede fisicamente al database.

B.5 Frammentazione e replica dei dati

Vediamo ora i motivi che fanno preferire la decomposizione (partizionamento) delle tabelle in piccole viste nella progettazione di un database distribuito.

Nei database distribuiti la partizione dei dati è necessaria per migliorare il processo di replica, infatti scegliendo di replicare solo alcune viste il peso della replica sulla rete diminuisce considerevolmente, analogamente sui siti periferici possono essere duplicate solo le informazioni effettivamente utilizzate e non tutte. Inoltre la frammentazione consente di trattare ogni partizione come unità, quindi le transazioni possono avvenire parallelamente sulle varie viste (anche le singole query globali possono essere decomposte in sub-query da eseguire concorrentemente *intra-query concurrency*).

Gli svantaggi maggiori della frammentazione sono dati dal fatto che non tutte le applicazioni concordano sulle viste da utilizzare, quindi è possibile che le viste ideali per una applicazione non lo siano per un' altra, e questa debba accedere a più viste diverse degradando le prestazioni. La frammentazione deve essere fatta in modo tale da ridurre al minimo tali problemi.

B.5.1 Tipi di partizione

Vediamo ora i diversi tipi di partizione, per farlo introduciamo una tabella da utilizzare come esempio: la tabella ARTICOLI.

Cod_art	Descr.	Prezzo	Categ.	Q.tà
1	Articolo 1	10000	C1	23
2	Articolo 2	8500	C1	3
3	Articolo 3	14500	C3	32
4	Articolo 4	3600	C4	4
5	Articolo 5	12500	C3	20

La frammentazione può avvenire in tre diverse modalità:

- *Partizione orizzontale.* Le tabelle vengono scomposte per gruppi di righe secondo particolari caratteristiche degli attributi. Ad esempio supponiamo di frammentare la tabella ARTICOLI in modo tale da avere solo gli articoli con quantità minore di 5, il risultato è il seguente:

Cod_art	Descr.	Prezzo	Categ.	Q.tà
2	Articolo 2	8500	C1	3
4	Articolo 4	3600	C4	4

Equivale quindi alla seguente relazione in algebra relazionale:

$$\delta_{\text{Q.tà}<5}(\text{Articoli})$$

.

In questo modo il destinatario della partizione (ad esempio il reparto acquisti) ha solo le righe che lo interessano con gli articoli che stanno per esaurirsi.

- *Partizione verticale.* Le tabelle vengono scomposte per gruppi di colonne (quindi solo alcuni attributi). Ad esempio partizionando verticalmente la tabella sugli attributi Cod_art, descr., Prezzo si ottiene la seguente:

Cod_art	Descr.	Prezzo
1	Articolo 1	10000
2	Articolo 2	8500
3	Articolo 3	14500
4	Articolo 4	3600
5	Articolo 5	12500

In algebra relazionale è:

$$\pi_{[\text{Cod\_art},\text{Descr.},\text{Prezzo}]}(\text{Articoli})$$

.

Anche in questo caso il destinatario (ad esempio il reparto vendite) ha solo le informazioni necessarie, senza quelle inutili.

- *Partizione mista.* E' l' unione dei due tipi di partizione precedenti, permette di restringere una tabella alle colonne e alle righe desiderate. Ad esempio la seguente tabella è ottenuta partizionando verticalmente rispetto alle colonne Cod_art, descr., Prezzo, e orizzontalmente alle righe con prezzo superiore a 10000:

Cod_art	Descr.	Prezzo
1	Articolo 1	10000
3	Articolo 3	14500
5	Articolo 5	12500

Il reparto vendite potrà avere un ufficio riservato alle vendite dei beni preziosi che avrà solo gli articoli di un certo valore.

B.5.2 Grado di frammentazione

Il *grado di frammentazione* è una scelta importante perchè da esso dipendono le prestazioni dell' esecuzione della query. Il minimo livello è ovviamente la tabella intera: nessuna partizione, quindi non si hanno vantaggi, si devono gestire tabelle intere che appesantiscono la distribuzione. Al contrario il livello massimo di frammentazione è la *tupla* (nel caso di partizione orrizzontale) o il singolo attributo (nel caso di partizione verticale), in questo caso aumenta l' accesso ai dati remoti creando sovraccarico inutile.

B.5.3 Alternative di allocazione

Dopo aver scelto come frammentare il database si deve decidere dove allocare i frammenti nei vari siti della rete. I dati allocati possono essere replicati oppure in singola copia. I motivi per la replica sono la robustezza e l' efficienza delle query read-only, infatti se ci sono copie multiple dei dati aumentano le possibilità di reperire le informazioni anche nel caso di guasti alla rete o al sistema propietario dei dati. Inoltre è possibile eseguire query sugli stessi dati in parallelo, accedendo a copie su server distinti degli stessi dati. I problemi nascono con le query di update, il sistema deve assicurare che tutte le copie dei dati sono modificate correttamente. La scelta di replicare i dati dipende quindi dal rapporto tra la quantità di dati acceduta in modo read-only e le query di update.

Le alternative possono essere le seguenti:

- *database partizionato*: non ci sono dati replicati, ogni frammento del database risiede su un solo server della rete.
- *database replicati parzialmente*: i frammenti del database sono replicati su tutti i server che hanno necessità di accedervi.
- *database replicati interamente*: tutto il database viene replicato sui server della rete.

La tabella seguente riassume un confronto tra le alternative viste:

	Full replication	Partial replication	Partitioning
Query processing	Easy	Same difficulty	
Directory management	Easy or none	Same difficulty	
Concurrency control	Moderate	Difficult	Easy
Reliability	Very high	High	Low
Reality	Possible application	Realistic	Possible application

Problemi di allocazione

Consideriamo un insieme di frammenti $F = \{F_1, F_2, \ldots, F_n\}$ e un insieme di server $S = \{S_1, S_2, \ldots, S_m\}$ sui quali sono eseguiti un insieme di applicazioni (query) $Q = \{q_1, q_2, \ldots, q_q\}$.

L' allocazione dei dati implica il problema di trovare la distribuzione "ottima" dei frammenti F sui server S. L' ottimo è definito rispetto due criteri:

- Costo minimo: comprende il costo di memorizzazione di ogni F_i sul server S_j, il costo della query q_i sul sito S_j, il costo di modifica di F_i su tutti i siti che lo contengono e infine il costo di comunicazione. L' allocazione deve essere fatta in modo da minimizzare il costo globale di questi componenti.

- Performance: l' allocazione deve mantenere le prestazioni, misurate in termini di:

  - Tempi di risposta: devono essere minimizzati.
  - Throughput: deve essere massimo in ogni sito.

Purtroppo un modello che permette di ottenere questi requisiti è complesso, e non è stato ancora sviluppato. Sono stati proposti schemi di allocazione più semplici tutti NP-completi, quindi la realizzazione di uno schema ottimo nel caso di un numero elevato di frammenti non è computazionalmente fattibile. Inoltre gli schemi semplici sviluppati non si possono applicare con successo ai database distribuiti in quanto:

- I frammenti non possono essere trattati come singoli file che possono essere allocati uno alla volta, ma ogni frammento coinvolge nella decisione sull' allocazione anche gli altri che sono utilizzati insieme (ad esempio nel caso di join distribuiti). Quindi la relazione tra i frammenti deve essere tenuta in considerazione.

- L' accesso ai dati da parte delle applicazioni non può essere modellato semplicemente come: richesta - risposta (ad esempio il semplice accesso remoto ai file), nei database distribuiti l' utilizzo dei dati è più complicato.

- Soddisfare i vincoli di integrità nel caso di frammenti è molto costoso, e questi costi non sono introdotti nei modelli semplificati.

- In modo analogo i costi del controllo della concorrenza.

Una ulteriore difficoltà nella definizione dello schema di allocazione ottimo è dovuta alla centralità che occupa l' allocazione dei dati rispetto i vari componenti del DDBMS, quindi deve essere nota la sua relazione con questi componenti.

E' necessario quindi separare il problema dell' allocazione dei file (*file allocation problem*, FAP) dall' allocazione dei frammenti nel database distribuito (*database allocation problem*, DAP). Essendo il FAP un problema NP-completo, possiamo aspettarci che anche il DAP lo sia, e di solito è vero, gli algoritmi risolutivi sono di tipo euristico, ad esempio branch-and-bound (problemi tipo "knapsack problem").

E' tuttavia possibile semplificare il problema utilizzando diverse strategie:

1. Assumere che le partizioni siano determinate contemporaneamente con i loro costi e benefici in termini di query processing.

2. Ignorare inizialmente la replica, trovare una soluzione ottimale per il problema non replicato, poi introdurre la replica e applicare un algoritmo che estenda la soluzione precedente al nuovo problema.

B.5.4 Replica dei dati

I concetti di partizionamento e frammentazione dei dati rappresentano le basi per la replica dei dati.

La replica dei dati porta all' esistenza di copie multiple di una stessa porzione di dati su diversi siti. Il problema della loro gestione è quindi molto importante per garantire la consistenza del database.

Tutti le repliche di uno stesso frammento devono essere identiche (*mutual consistency*), questo può essere garantito utilizzando una tecnica denominata *one copy serializable* che indica la possibilità di serializzare le repliche da parte dello scheduler, deve rispettare le seguenti condizioni:

- ogni scheduler locale deve essere serializzabile;

- due operazioni in conflitto devono essere nello stesso ordine in tutti gli scheduler.

Un protocollo di controllo della replica che segue queste indicazioni è il *read-once/write-all* (ROWA). Consideriamo una partizione di dati x (definita *logical data*) e un insieme di copie $x_1, x_2, \ldots, x_n$ (definiti *physical data*). Le c' è la trasparenza della replica allora le transazioni dell' utente vanno a modificare x. Il processo di replica deve mappare ogni operazione di lettura "read(x)" in una operazione "read(x_j)", cioè la lettura avviene solo sulla replica del dato originale. Ogni scrittura "write(x)" però deve essere eseguita

solo sulla partizione originale x poi replicata, questo significa che l' operazione di modifica di un dato viene trasformata in una operazione di modifica in ogni sito contenente dati replicati.

Il protocollo ROWA ha il difetto di diminuire la disponibilità dei dati in caso di guasti alla rete o ad un sito, infatti in questi casi l' operazione di scrittura in ogni sito contenente una replica può non completarsi. La soluzione di questo problema non è semplice, sono stati proposti molti algoritmi per consentire una gestione sempre consistente e sicura dei malfunzionamenti, tutti partono dal presupposto di ignorare il guasto su un sito e modificare i dati di tutti gli altri, il disallineamento del database del sito non funzionante dovrà essere recuperato dopo la riparazione del guasto attraverso una operazione di sincronizzazione dei dati.

La metafora publishing/subscribing

Una possibile terminologia dei modelli di replica è basata sulla metafora publishing/subscribing:

- Quando il dato è reso disponibile per la replica si dice che viene pubblicato (*published*). Il server che contiene il database sorgente e rende disponibile il dato è detto *publication server* o *publisher*.

- L' unità base di replica è chiamata articolo (*article*). L' articolo contiene i dati relativi ad una sola tabella e deve appartenere ad una publication.

- Una *publication* è una collezione di articoli (almeno uno). Una o più publication possono essere create da ciascun utente di un publisher.

- Quando un server richiede una pubblicazione diventa un *subscriber* (o *subscription server*). Un subscriber può richiedere la replica di alcune pubblicazioni (anche tutte) e di specifici articoli (anche tutti) all' interno di una pubblicazione.

Richiameremo in seguito questa terminologia, in quanto è quella utilizzata anche dal sistema SQL Server, da noi utilizzato.

Modalità di replica

In generale, la modalità di replica dei dati in applicazioni distribuite può essere suddivisa in due categorie:

- *Tight consistency*: In questo tipo di modello viene mantenuta, in ogni istante, la consistenza dei dati replicati rispetto all' originale. Per rispondere ad una esigenza cosìstringente, il sistema deve essere dotato di una rete ad alta velocità (tipicamente una LAN), prevedere l' indisponibilità di accesso ai dati durante tutti i periodi di aggiornamento (ridotta disponibilità) ed un protocollo *two phase commit* (2PC) per garantire l' integrità e la consistenza dei dati.

- *Loose consistency*: è un modello di replica che ammette un periodo di latenza tra il momento in cui i dati originali sono modificati ed il momento in cui tutte le copie vengono aggiornate. Naturalmente non viene garantita, in ogni istante, l' uguaglianza dei dati replicati rispetto all' originale.

Il vantaggio del modello loose consistency è quello di supportare anche reti LAN o WAN piuttosto lente ed eventualmente non collegate in modo permanente. Inoltre garantisce una maggiore disponibilità d' uso del database ed una maggiore scalabilità rispetto al tight consistency.

L' interesse delle applicazioni che verranno considerate in seguito, ricade nel modello loose consistency, poichè si ipotizza una rete di DBMS debolmente connessi, tipocamente collegati da reti a velocità medio/bassa (quindi LAN, WAN), o collegamenti non permanenti.

B.6 Sicurezza dei dati

Il DBMS deve garantire la sicurezza dei dati, distinguiamo quindi due aspetti:

- *Data protection*: impedire all' utente la visione fisica dei dati, questo è un compito svolto abitualmente dai file systems dei sistemi operativi distribuiti. La tecnica principale è la crittografia: i dati crittografati possono essere decrittografati solo dagli utenti che possiedono le chiavi corrette. I sistemi più utilizzati sono il DES *Data Encription Standard* e gli algoritmi a chiave pubblica (alcuni sono RSA, RC4 e IDEA).

- *Authorization control* deve garantire che solo gli utenti autorizzati possano eseguire determinate operazioni sul database. Questo controllo viene effettuato in parte dal sistema operativo (l' accesso fisico al sistema), e in parte dai DBMS distribuiti o centralizzati, i quali possono permettere o negare l' accesso anche a sottoinsiemi dei dati, permettendo un accesso differenziato agli utenti con previlegi diversi.

Il secondo aspetto è maggiormente interessante, in particolare per le funzioni svolte dai DBMS. Infatti rispetto ai sistemi operativi che controllano le autorizzazioni, i DBMS consentono una maggiore differenziazione in modo tale che utenti diversi hanno diritti diversi sulla stessa porzione di dati, questo implica che un DBMS deve distinguere gli utenti non solo attraverso il nome e la password, ma anche con il ruolo, la funzione e il gruppo di appartenenza. Queste funzionalità verranno sfruttate nel sistena di replica che proponiamo nel seguito.

B.7 Esecuzione delle query

L' esecuzione delle query è svolta da un modulo denominato *query processor*, che consente di esprimere le interrogazioni in linguaggio ad alto livello (anche naturale) in modo tale da nascondere completamente i dettagli implementativi. Questo modulo deve anche occuparsi di ottimizzare le query introdotte dall' utente, il quale cosìnon deve preoccuparsi della "qualità" delle interrogazioni introdotte.

Il query processor è un punto critico dell' intero DBMS, le performance globali del sistema dipendono in modo considerevole dal metodo scelto per eseguire le query, di conseguenza la scelta di un buon algoritmo è molto importante ma ed è difficile il confronto di più metodi diversi, sono stati introdotti quindi dei parametri per la valutazione:

Linguaggio. Il query processor deve eseguire un mapping tra il linguaggio di input (quello utilizzato dall' utente) e il linguaggio di output (di solito in algebra relazionale con primitive di comunicazione).

Tipo di ottimizzazione. La ricerca del miglior metodo di esecuzione implica un costo notevole per l' ottimizzatore, un overhead che deve essere minimo. Non è sufficiente quindi esaminare il costo delle query ottimizzate (che in ogni caso deve essere minimo), ma si deve tenere in considerazione anche il costo di ottimizzazione del DBMS.

Tempo di ottimizzazione. Una query può essere ottimizzata staticamente: cio'e prima dell' esecuzione; oppure dinamicamente: durante l' esecuzione. I due metodi hanno vantaggi e svantaggi:

- Statico: il vantaggio è che l' ottimizzazione viene fatta una volta sola durante la compilazione, poi riutilizzata anche per le query successive, in modo tale che il costo di ottimizzazione viene in parte ammortizzato. Il metodo utilizzato è di tipo statistico,

questo è il punto debole: infatti nel caso di dati errati tutta l' ottimizzazione è inutile.

- Dinamico: l' ottimizzazione viene effettuata durante l' esecuzione, in ogni momento è possibile scegliere il metodo migliore per la singola operazione da eseguire. Non sono richieste le statistiche sul database (ad eccezione della prima query), i dati vengono ricavati run-time, quindi la propbabilità di commettere errori è molto inferiore al metodo statico. Il più grosso svantaggio è dato dal costo che viene moltiplicato per ogni operazione da eseguire.

Sono possibili metodi misti: si segue l' approccio statico fino a quando i dati statistici si dimostrano errati, a questo punto si effettua l' ottimizzazione dinamica con la correzione delle statistiche (per le esecuzioni successive).

Statistiche. Le statistiche racchiudono informazioni sullo stato del database, il numero e la cardinalità dei frammenti, propietà degli attributi. L' utilizzo delle statistiche è fondamentale nel caso statico, marginale in quello dinamico (dove si usa solo per la prima query), è importante però che siano sempre aggiornate e corrette, per questo vengono fatti update periodici, che hanno un costo notevole ma sono indispensabili. Dopo ogni modifica alle statistiche alcune query ottimizzate in modo statico possono essere riottimizzate per migliorarne l' efficienza.

Sito decisionale. Nel caso di ottimizzazione statica, uno o più siti possono essere coinvolti nella decisione della strategia. Se il sistema decisionale è centralizzato allora un singolo sito si occupa dell' ottimizzazione, e sarà l'unico contenente il database delle statistiche. Viceversa, se la decisione è distribuita tra tutti i server del sistema allora le query sono ottimizzate utilizzando dati locali. Sono possibili approcci ibridi, in cui un sito è prevalente e si occupa della maggior parte delle ottimizzazioni, gli altri siti effettuano decisioni locali.

Topologia di rete. E' importante la conoscenza della topologia della rete in quanto il peso del costo di comunicazione cambia. Nel caso delle reti geografiche il costo da minimizzare può essere ristretto al solo costo di comunicazione trascurando le altre componenti. Nel caso di reti locali il costo di comunicazione può essere considerato uguale al costo di I/O, quindi diventa ragionevole aumentare il parallelismo dell' esecuzione delle query.

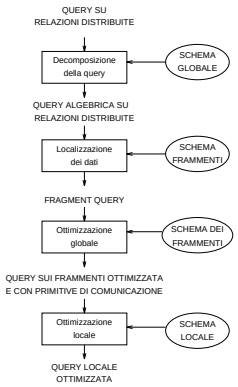


Figura B.7: Schema di esecuzione di una query su DDBS

Replica dei frammenti. E' utile avere frammenti del database replicati su diversi siti, l' ottimizzatore deve conoscere quindi il luogo in cui i dati vengono acceduti: se su una replica locale, oppure sul sito proprietario.

Utilizzo dei semijoins. Il semijoin ha la propietà di ridurre la dimensione delle relazioni, di consenguenza quando il costo di comunicazione è importante, conviene utilizzare dei semijoin che riducono le quantità di dati da scambiare.

L' esecuzione di una query in un ambiente distribuito viene scomposta in quattro diversi sotto-problemi, rappresentati a livelli nella figura B.7. L' input è una query su dati distribuiti espressa mediante gli operatori relazionali, è costituita da relazioni globali (distribuite), questo significa che la distribuzione viene nascosta all' utente. L' obiettivo è trasformare la query globale in una sequenza di operazioni locali sul database locale.

I primi tre livelli:

- Query decomposition,
- Data localization,
- Global query optimization,

sono eseguiti da un sito centrale ed utilizzano informazioni globali, il quarto:

- Local query optimization

è eseguito localmente.

Query decomposition Il primo livello decompone una query globale distribuita in una query sulle relazioni distribuite. Questo obiettivo viene raggiunto mediante una sequenza di passi successivi:

Normalizzazione La query in input può essere arbitrariamente complessa, questo dipende dalle caratteristiche del linguaggio usato per scriverla. L' obiettivo della normalizzazione è quello di trasformare la query nella forma normale, in modo tale da facilitare le operazioni successive.

Nel caso di linguaggi relazionali come l' SQL la trasformazione più importante è eseguita sulla clausola `WHERE`. Ci sono due possibili forme per i predicati, in funzione alla precedenza dell' operatore AND o dell' OR.

Analisi semantica Nel secondo passo viene eseguita l' analisi semantica della query, in modo tale da rilevare gli errori e in questo caso non procedere ulteriormente con l' esecuzione, ma rifiutare la query e visualizzare un messaggio di errore.

Eliminazione della ridondanza Il terzo passo è la semplificazione della query in modo tale da eliminare ogni ridondanza. In particolare ad una query su una view può essere arricchita con dei predicati per la corrispondenza tra la vista e la relazione assicurando l' integrità semantica, questi predicati possono essere ridondanti rispetto quelli già presenti nella query. Occorre quindi eliminare i predicati ridondanti utilizzando le propietà di idempotenza degli operatori logici.

Ricostruzione Infine, ultimo passo è la ricostruzione della query: la sua riscrittura utilizzando le regole dell' algebra relazionale. Abbiamo due operazioni da compiere:

- trasformazione della query dal calcolo relazionale all' algebra relazionale.
- ristrutturare la query in algebra relazionale per ottimizzare le performance.

Localizzazione dei dati Il secondo livello dello schema dell' esecuzione della query distribuita (figura B.7) riguarda la localizzazione dei dati utilizzati dalla query (ottenuta dal livello precedente). Questo livello determina quali frammenti sono necessari, e trasforma la query distribuita in una query di frammento (*fragment query*) utilizzando lo schema di frammentazione.

Questa trasformazione viene eseguita applicando delle regole di ricostruzione alla relazione globale, si ottiene un programma relazionale (*localization program*) che ha come operandi i frammenti.

Ottimizzazione globale Questo livello riceve in ingresso le query sui frammenti generate al livello precedente, in uscita le fornisce ottimizzate e con le operazioni per le comunicazioni. I livelli precedenti hanno già ottimizzato in parte la query, però in modo indipendente dalle caratterisctiche del frammento.

L' ottimizzatore globale deve trovare la strategia migliore per l' esecuzione della query, cioè un insieme di operazioni relazionali e di primitive di comunicazione "send" e "receive" tra i siti. Permutando l' ordine delle operazioni su un frammento è possibile individuare delle query equivalenti, ciascuna di costo diverso, l' ottimizzatore deve scegliere le query a costo minimo. Per semplificare, molti DBMS trascurano i diversi costi legati all' elaborazione locale per evidenziare solo i costi di comunicazione, questo può essere vero solo nel caso di reti metropolitane e geografiche con banda limitata.

Ottimizzazione locale L' ultimo livello dello schema di esecuzione delle query distribuite è l' ottimizzazione locale. Riceve in ingresso le query sui frammenti già ottimizzate dal livello precedente e fornisce in uscita query ottimizzate sul database locale.

L' ottimizzazione è fatta seguendo gli algoritmi tradizionali, utilizzati sui DBMS centralizzati.

B.8 Gestione delle transazioni distribuite

Le transazioni devono avere precise caratteristiche:

- Atomicità: se una transazione si interrompe (per qualunque motivo) i risultati parziali ottenuti fino a quel momento devono essere annullati. Questo equivale a dire che le transazioni devono essere viste come una unica istruzione, anche se racchiudono più operazioni, se anche solo una di queste fallisce, tutta la transazione deve fallire.

- Consistenza: una transazione è una trasformazione dello stato del sistema in un altro, ma deve mantenere tutte le invarianti predefinite.

- Isolamento: una transazione non completa non può fornire un risultato ad un' altra transazione concorrente prima del commit.

- Persistenza: quando una transazione termina (commit) il sistema deve garantire che il risultato non venga perso, indipendentemente dai guasti che possono accadere dopo il commit.

Una transazione è di solito parte di una applicazione che non ha queste caratteristiche, è possibile utilizzare la primitiva `begin transaction` per definire l' inizio di una sequenza di istruzioni che diverranno parte di una stessa transazione, e quindi con le caratteristiche descritte, che termina quando incontra un comando `commit` o *abort*.

Le transazioni distribuite eseguono processi che si trovano su differenti siti, uno di questi è denominato "root" e si trova sul sito di origine della transazione. In questo sito viene iniziata la transazione (con il `begin transaction`) e viene conclusa, inoltre si occupa di creare i processi da eseguire sui siti remoti.

Possiamo ora notare come il processo "root" abbia il controllo su tutta l' esecuzione della transazione, dopo averla iniziata se si presenta un errore la termina, altrimenti crea il processo da eseguire sul sito remoto e gli passa i parametri necessari, quindi esegue il commit. Sul sito remoto abbiamo procedure con i propri parametri che eseguono la parte di transazione che interessa quel sito, poi terminano.

B.8.1 Transaction manager

L' esecuzione di una transazione distribuita avviene tramite un modulo del DDBMS denominato "distributed execution monitor" (figura B.8), il quale è composto a sua volta da due componenti: *transaction manager* che coordina l' esecuzione delle operazioni sul database (le applicazioni) e uno *scheduler* che controlla la concorrenza dell' esecuzione delle transazioni per sincronizzare l' accesso al database. Può essere presente anche un modulo di recovery, che si occupa della gestione delle condizioni di errore e di ripristino.

Ogni transazione inizia in un sito che chiameremo "sito origine", il transaction manager di questo sito ne coordina l' esecuzione.

Trascuriamo per ora il caso di transazioni concorrenti, l' interfaccia tra il transaction manager e l' applicazione utente è composta da cinque comandi:

1. `begin transaction`. Questo comando indica al transaction manager che sta per iniziare una nuova transazione, in questo istante vengono raccolte alcune informazioni come il nome dell' applicazione, il nome della transazione, etc . . . .

2. `read`. Se un dato è memorizzato localmente il suo valore è letto e

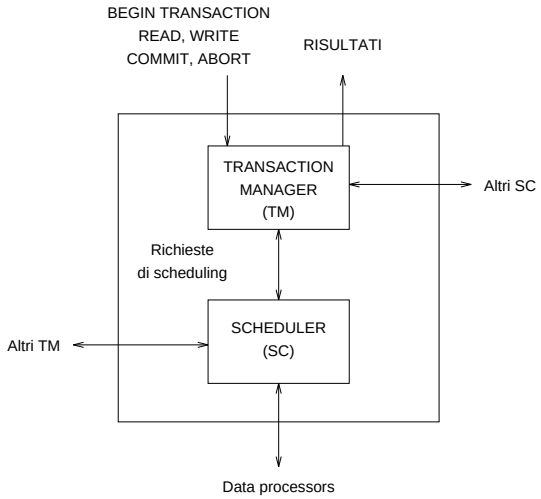


Figura B.8: Schema del distributed execution monitor

ritornato all' applicazione. Altrimenti viene selezionata una copia del dato e ritornato il suo valore.

3. `write`. Il transaction manager si occupa di coordinare l' update del dato in tutti i siti in cui risiede (originale e copie replicate).

4. `commit`. Il transaction manager effettua la modifica fisica del database confermando tutti i dati alterati dalla transazione.

5. `abort`. Le modifiche ai dati vengono ignorate, il database fisico rimane inalterato.

Durante l' esecuzione di queste istruzioni il transaction manager può interagire con gli scheduler di diversi siti, il coordinamento rimane del sito di origine della transazione.

B.9 Controllo della concorrenza distribuita

In un sistema distribuito riveste fondamentale importanza la gestione della concorrenza delle transazioni, deve garantire che la consistenza del database sia mantenuta anche nell' ambiente distribuito e multiutente. Se le transazioni sono internamente consistenti, il metodo più semplice è quello di eseguirle una dopo l' altra sequenzialmente (*serializzabilità*), questo metodo assicura la consistenza di tutta l' applicazione, ma il troughput che si ottiene è il minimo. Il numero di transazioni concorrenti è uno dei parametri più importanti

dei DDBMS, il meccanismo che controlla la concorrenza deve assicurare un buon equilibrio tra il numero di transazioni concorrenti e la consistenza del database. Gli algoritmi per il controllo della concorrenza possono essere di due tipi:

- *Pessimistici*: partono dal presupposto che molte transazioni andranno in conflitto, quindi sincronizzano l' esecuzione delle transazioni concorrenti subito dopo il loro inizio.
- *Ottimistici*: non molte transizioni andranno in conflitto, allora possono essere sincronizzate alla fine della loro esecuzione.

Il criterio di classificazione è la primitiva di sincronizzazione utilizzata: *locking* cioè le tabelle sono accedute in modo mutuamente esclusivo, oppure *timestamp ordering* in cui l' ordine di esecuzione delle transazioni viene riordinato secondo criteri specifici.
Con l' approccio locking based la sincronizzazione delle transazioni è fatta da meccanismi logici o fisici che bloccano alcune porzioni del database, in base alla posizione in cui l' algoritmo agisce possiamo avere un locking:

- Centralizzato: un sito della rete blocca le tabelle per l' intero database
- Copia principale: una delle copie (realizzate mediante replica) dei dati bloccati è la principale, tutte le transazioni che interessano quella porzione di dati devono essere autorizzate dal sito che contiene la principale. Se il database non è replicato il controllo delle tabelle viene distribuito tra i vari siti.
- Distribuito: il controllo del locking è condiviso tra tutti i siti della rete, se un sito deve utilizzare una porzione di dati deve ottenere il permesso da tutti i possessori delle tabelle.

L' approccio timestamp-ordering coinvolge l' ordine di esecuzione delle transazioni, in modo da riorganizzarle per evitare conflitti. Gli algoritmi timestamp-based non utilizzano la mutua esclusione come i precedenti, ma si basano sulla selezione a priori di un ordine di esecuzione favorevole. Il transaction manager assegna ad ogni transazione durante l' inizializzazione T_i un timestamp univoco $ts(T_i)$. Ordinando i timestamp possiamo serializzare le transazioni. Lo scheduler deve ripetere l' ordinamento dei timestamp ad ogni nuova transazione, in modo tale da evitare i conflitti, se la nuova operazione appartiene ad una transazione più recente di tutte quelle in conflitto è accettata, altrimenti viene rifiutata e tutta la transazione riparte con un nuovo timestamp.

B.10 Reliability

L' attendibilità dei dati memorizzati è importante, e deve essere garantita anche se il sistema diventa inaffidabile, cioè anche se un componente del sistema distribuito si guasta un DDBMS affidabile deve poter continuare a soddisfare le richieste dell' utente senza violare la consistenza del database.

Ogni deviazione del sistema dal comportamento descritto nelle specifiche è da considerarsi come un guasto, può essere di due tipi: *permanente* quando è irreversibile, e richiede un intervento per ripararlo, *temporaneo* (o intermittente) se è dovuto a instabilità hardware o software o dipendente dalle condizioni ambientali (ad esempio l' aumento della temperatura), può essere riparato dopo aver individuato la causa del problema, nel caso del fattore ambientale invece può essere impossibile l' annullamento della condizione che provoca il malfunzionamento.

Il termine reliability si riferisce alla probabilità $R(t)$ che il sistema non presenti nessun problema durante un intervallo di tempo specificato $[0, t]$. Viene di solito utilizzato per descrivere sistemi che non possono essere riparati o che eseguono attività critiche cioè non è possibile interromperle per eseguire la riparazione. Per quantificare l' affidabilità di un sistema si utilizzano due misure: *mean time between failures* MTBF e *mean time to repair* MTTR, indicano rispettivamente il tempo medio tra due guasti successivi e il tempo medio di riparazione, a volte è utile misurare anche il *mean time to fail* MTTF cioè il tempo medio entro cui avviene il primo guasto, tramite la relazione: $MTBF = MTTF + MTTR$ è possibile legarlo agli altri valori. Nei prodotti commerciali sono valori forniti dal produttore.
Analizziamo ora i guasti specifici dei sistemi distribuiti di basi di dati, il sistema di recovery del DDBMS deve gestire quattro tipi di problemi:

1. Transaction failure: dipendono da diverse cause: dati errati in input, deadlock, controllo della concorrenza, lock, ... , e ogni ulteriore caso in cui la transazione non può terminare con un commit. L' approccio da utilizzare per risolvere questi problemi è terminare la transazione con un abort, quindi resettare il database ad uno stato consistente precedente all' inizio della transazione.

2. System failure: causati da problemi hardware (guasti fisici agli elaboratori, mancanza di energia elettrica) o software (bug nel programma o nel DBMS o nel sistema operativo). Il database si suppone integro, ad eccezione eventualmente del risultato delle transazioni in esecuzione al momento del guasto contenute in un buffer nella memoria centrale. Se il sistema è distribuito il sito in cui è avvenuto il problema diventa ina-

cessibile dall' esterno, e in funzione della topologia della rete provocare una interruzione che determina due sottoreti isolate tra loro.

3. Media failure: la memoria di massa contentente di dati si danneggia, può avvenire sia per cause fisiche (danni alle testine di lettura/scrittura dei dispositivi, o guasti di natura magnetica) che per cause software (bug nel sistema operativo). Backup periodici e copie multiple dei dati (tecnologia RAID) permettono di recuperare in parte o completamente i dati. Di solito è possibile trascurare i guasti ai supporti, in quanto sono meno probabili degli altri tipi.

4. Communication failure: a differenza degli altri tre tipi che sono comuni anche ai sistemi centralizzati, i problemi relativi alle comunicazioni sono caratteristica dei sistemi distribuiti. La perdita di messaggi, la sequenza di arrivo errata o guasti alle linee di trasmissione sono i casi che si presentano più frequentemente. Alcuni problemi non sono di competenza del DDBMS, infatti i livelli inferiori delle architetture di rete di dovrebbero già occupare della verifica della correttezza della comunicazione. Per quanto riguarda i messaggi persi dovrà essere il DDBMS ad accorgersi delle incongruenze tra la situazione prima e dopo il guasto, e rilevare cosìla perdita di informazioni.

Ogni malfunzionamento in un sistema distribuito deve essere gestito sempre sotto due aspetti: sul sito locale (come un sistema centralizzato) e durante la comunicazione. Dovranno esserci quindi degli algoritmi che permettono di conservare l' integrità dei dati locale e dei protocolli di comunicazione che permettono di eseguire transazioni distribuite affidabili.
In locale, un componente dedicato, il *Local Recovery Manager*, si occupa della gestione dei malfunzionamenti.
Anche per le transazioni distrbuite, però, bisogna garantire le caratteristiche di atomicità e persistenza: opportuni protocolli devono controllare la transazione anche durante le comunicazioni tra i siti, in modo tale da poter sempre garantire che il database sia in uno stato consistente. Il protocollo più noto, che qui citiamo come esempio, è il *Two Phase Commit*, o 2PC.

2PC protocol Il protocollo 2PC estende gli effetti del commit locale alle transazioni distribuite, viene richiesta ai siti coinvolti nell' esecuzione la disponibilità ad effettuare un commit prima che gli effetti della transazione diventino permanenti. La sincronizzazione tra i siti è necessaria in quanto alcuni scheduler possono non essere pronti a terminare la transazione col commit e di conseguenza per evitare il commit a cascata anche di transazioni

valide occorre evitare che un dato modificato da una transazione che fallisce sia utilizzato da un' altra operazione prima del commit. Un' altra ragione per cui un sito partecipante può rifiutare la richiesta di commit è dovuta alla presenza di un deadlock che porta all' abort della transazione, in questo caso il sito può effettuare l' abort anche senza consultare gli altri siti, questo è detto *unilateral abort*.

Il 2PC viene eseguito con i seguenti passi (figura B.9):

- Il coordinatore scrive il "begin commit" nel suo log.
- Quindi invia un messaggio "prepare" a tutti i partecipanti, informandoli dell' inizio del commit, quindi entra in uno stato di wait.
- Quando un partecipante riceve il messaggio controlla se può terminare la transazione con successo. Ci sono quindi i due casi:
  1. Si (commit), invia un messaggio "vote-commit" al coordinatore ed entra in uno stato di attesa.
  2. No (abort), invia un messaggio "vote-abort" al coordinatore ed entra in uno stato di attesa. In questo caso il sito localmente può dimenticare la transazione, in quanto effettua un abort unilaterale.
- Se la decisione è di terminare con abort (è sufficiente un solo sito), allora il coordinatore invia un abort globale a tutti i partecipanti ed entra in uno stato "commit"
- I partecipanti possono eseguire un commit o un abort coerentemente con quanto inviato dal coordinatore, e ritornano un messaggio di conferma.
- Il coordinatore quando riceve tutte le risposte dei partecipanti termina la transazione.

La scelta sul modo di terminare la transazione è presa dal coordinatore seguendo due regole di base (che insieme vengono chiamate *global commit rule*):

1. Se anche un solo partecipante vota per l' abort, il coordinatore decide per l' abort globale.
2. Se tutti i partecipanti votano per il commit, il coordinatore decide per il commit globale.

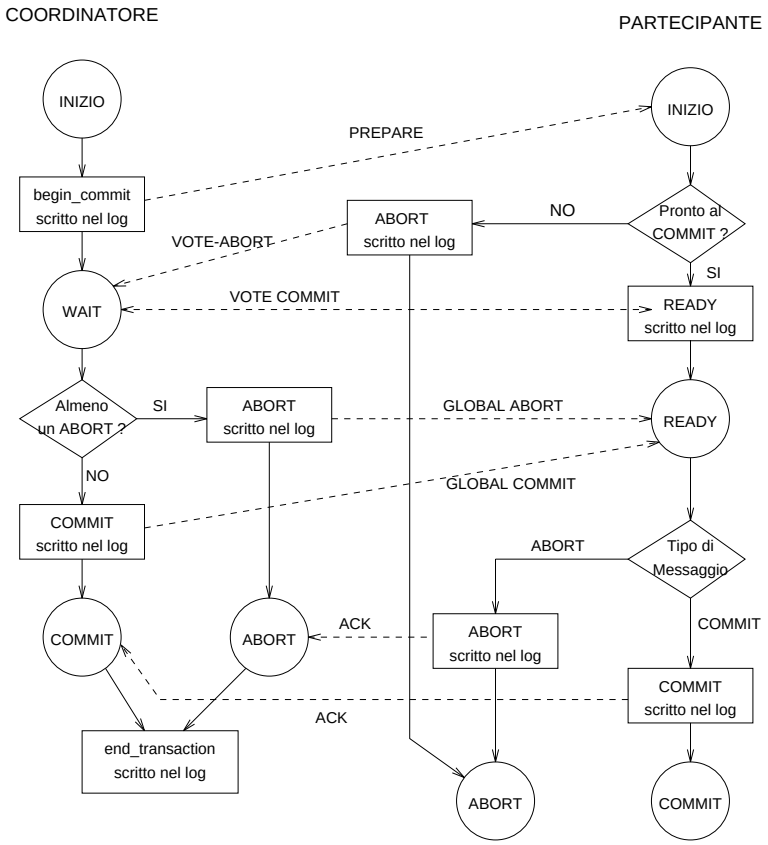


Figura B.9: Funzionamento del two-phase commit protocol

Vediamo alcune osservazioni: la votazione è irrevocabile, non sono possibili cambiamenti nelle decisioni prese. Inoltre tutti gli stati di attesa sono controllati anche da dei timer, per evitare effetti indesiderati vengono imposti dei timeout da rispettare, entro cui la risposta deve arrivare.
E' possibile implementare il 2PC seguendo diverse topologie, quello descritto fino ad ora è il *centralized two-phase commit*, in quanto la comunicazione avviene solo tra un sito centrale che svolge la funzione di coordinatore e gli altri siti del sistema.
Una alternativa è il *linear 2PC* in cui la comunicazione avviene da un sito ad un altro, seguendo un ordine prestabilito. Lo svantaggio della struttura lineare è che non è possibile alcun tipo di parallelismo, quindi è un metodo naturalmente lento. Può trovare tuttavia impiego nelle reti che non dispongono della possibilità del broadcast.
Un' altra struttura è il *distributed 2PC* che a differenza del precedente coinvolge tutti i siti nella prima fase, cosìche la decisione viene presa indipendentemente. Non servè più la seconda fase, in quanto i partecipanti hanno già preso una decisione.
Sono state proposte diverse varianti ai protocolli base del 2PC per incrementare le prestazioni. Questo può essere fatto in due modi:

1. Riducendo il numero di messaggi trasmessi tra coordinatore e partecipanti (*presumed abort*).

2. Riducendo il numero di scritture sui log (*presumed commit*).

La prima variante parte dal presupposto che quando un partecipante interroga il coordinatore per conoscere i risultati della transazione e il coordinatore non ha questi dati in memoria virtuale, la risposta sarà sempre l' abort. Questo funziona fino a quando il coordinatore rimuove le informazioni della transazione in seguito ad un commit, in ogni caso questo non succede finché tutti i partecipanti non spediscono l' acknowledge. Questa tecnica permette di risparmiare sul numero di messaggi da scambiare quando una transazione deve terminare con abort.

Il secondo metodo parte dal presupposto che non ci sono informazioni sulla transazione e quindi deve essere considerata commit. Non è il duale della precedente metodologia, in quanto il commit non è forzato immediatamente, inoltre il commit non richiede l' acknowledge. Il coordinatore invia il messaggio "prepare" aggiungendo i nomi di tutti i partecipanti e si pone in attesa, quando questi ricevono il messaggio decidono sulla fine della transazione e la comunicano al coodinatore mediante i messaggi "vote-abort" e "vote-commit". Quando il coordinatore riceve le votazioni di tutti i partecipanti decide per il "global-abort" o per il "global-commit". Nel caso del

commit dimentica la transazione e aggiorna il database. Se il risultato è abort termina la transazione e invia un acknowledge ai partecipanti i quali a loro volta terminano l' esecuzione. In generale, comunque, il 2PC è un protocollo "pesante", che rallenta notevolmente eventuali applicazioni distribuite.

Appendice C

Esempi di script

In questa appendice si introducono alcuni esempi di scripts citati nel corso della trattazione della tesi.

C.1 Stored procedure mynewdev

Questa stored procedure è stata scritta appositamente per la creazione via script di device su SQL Server. Prevede tre parametri, che sono rispettivamente il nome logico del device da creare, il nome fisico dello stesso (nome del file ".dat"), la dimensione in Mbyte. L'interesse della procedura risiede nella capacità di generare un numero di device unico, diverso dagli esistenti, problema trascurato negli script precedenti.

```
use master
go

if exists (select * from sysobjects where id = object_id('dbo.mynewdev')
           and sysstat & 0xf = 4)
   drop procedure dbo.mynewdev
go

create procedure mynewdev @logic_name varchar(30),@physical varchar(255),
                          @size int
as

declare @vdevno integer

select @vdevno = (min(low)/0x01000000)+1
        from sysdevices d1
```

```
        where low/0x01000000 between 0 and 254
        and not exists
                (select * from sysdevices d2
                        where d2.low/0x01000000 = (d1.low/0x01000000)+1)
go

disk init
        name = @logic_name,
        physname = @physical,
        vdevno = @vdevno,
        size = @size*1000
go
```

C.2 Script di modfica delle tabelle di *aws*

In questa sezione mostriamo alcuni degli script generati dal DOM Builder per la configurazione della replica di *aws*.

C.2.1 Generazione delle chiavi primarie

Questo script serve ad aggiungere le chiavi primarie alle tabelle che le ammettono, secondo i criteri esposti nel Capitolo 6. Si compone quindi di una lunga serie di statement "ALTER TABLE ADD CONSTRAINT PRIMARY KEY", intervallati da commenti per le tabelle che in questa fase non vengono modificate. Per motivi di spazio riportiamo solo una parte dello script, in cui appare chiaramente la sua struttura.

```
/* script di modifica tabella ADSDBARCHDFNAMES */
ALTER TABLE ADSDBARCHDFNAMES ADD CONSTRAINT pk_ADSDBARCHDFNAMES
PRIMARY KEY NONCLUSTERED (
        ArchProfileId,
        BPDefName
 )
go

/* script di modifica tabella ADSDBARCHIVE */
ALTER TABLE ADSDBARCHIVE ADD CONSTRAINT pk_ADSDBARCHIVE
PRIMARY KEY NONCLUSTERED (
        ArchProfileId
 )
```

```
go

/* script di modifica tabella ADSDBBPDEFBPDLLMAPNAME */
/* tabella senza indici */

/* script di modifica tabella ADSDBBPDEFBPROLEMAPPING */
ALTER TABLE ADSDBBPDEFBPROLEMAPPING ADD CONSTRAINT
pk_ADSDBBPDEFBPROLEMAPPING PRIMARY KEY NONCLUSTERED (
        BPDefId,
        BPRoleId,
        OrgRoleId
 )
go
/* script di modifica tabella ADSDBBPDEFBPROLES */
ALTER TABLE ADSDBBPDEFBPROLES ADD CONSTRAINT pk_ADSDBBPDEFBPROLES
PRIMARY KEY NONCLUSTERED (
        BPDefId,
        BPRoleId
 )

go

[..]
```

C.2.2 Generazione delle tabelle di *awsdup*

Questo script contiene tutti gli statement di "CREATE TABLE" che permettono di ricreare, sul database *awsdup*, le tabelle che in *aws* non ammettono chiavi primarie. Ad ognuna di esse è aggiunto qui un campo chiave gestito dal sistema. Data la lunghezza dello script originale, anche in questo caso se ne inserisce una parte che lo esemplifica. In coda, poi, si è posto lo script di creazione della tabella SURR_KEY, per la gestione delle chiavi surrogate in *awsdup*.

```
/* script di creazione tabelle duplicate di aws in awsdup */

USE awsdup
go

/* script di generazione tabella ADSDBBPDEFBPDLLMAPNAME */
```

```
/****** Object:  Table ADSDBBPDEFBPDLLMAPNAME ******/
if exists (select * from sysobjects where
  id = object_id('ADSDBBPDEFBPDLLMAPNAME') and sysstat & 0xf = 3)
  drop table ADSDBBPDEFBPDLLMAPNAME
go

create table ADSDBBPDEFBPDLLMAPNAME (
        ID int               NOT NULL,
        BPDefId int          NOT NULL,
        BPDefDLLMapName varchar  NULL,
        DLLRegistryId int        NULL,


  constraint pk_ADSDBBPDEFBPDLLMAPNAME primary key
  (
    ID
  )
)
go

/* script di generazione tabella CONFIGINFO */
/****** Object:  Table CONFIGINFO ******/
if exists (select * from sysobjects where
  id = object_id('CONFIGINFO') and sysstat & 0xf = 3)
  drop table CONFIGINFO
go

create table CONFIGINFO (
        ID int                   NOT NULL,
        szArchDBName varchar     NULL,
        iTMPollInterval smallint NOT NULL,
        iTMOptions smallint      NOT NULL,
        szDBVersion varchar      NULL,
        iDBVersion smallint      NOT NULL,


  constraint pk_CONFIGINFO primary key
  (
    ID
  )
)
```

```
go

[..]

/* script di generazione tabella DFBPROLE */
/****** Object:  Table DFBPROLE ******/
if exists (select * from sysobjects where
  id = object_id('DFBPROLE') and sysstat & 0xf = 3)
  drop table DFBPROLE
go

create table DFBPROLE (
        ID int               NOT NULL,
        lBPDid int               NULL,
        lBPRoleid int            NULL,
        szBPRoleName varchar     NULL,
        szCompFieldName varchar  NULL,
        szDesc varchar           NULL,


  constraint pk_DFBPROLE primary key
  (
    ID
  )
)
go

[..]

/* script di generazione tabella PWF */
/****** Object:  Table PWF ******/
if exists (select * from sysobjects where
  id = object_id('PWF') and sysstat & 0xf = 3)
  drop table PWF
go

create table PWF (
        ID int not null,
        BPID int         NOT NULL,
        WFID int         NOT NULL,
        BPDefID int      NOT NULL,
```

```
WFDefID int      NOT NULL,
GrpID int        NULL,
CExpectsComp varchar     NULL,
CRespDue varchar         NULL,
PCompDue varchar         NULL,
PRespDue varchar         NULL,
WFCustName varchar       NULL,
WFCustRole varchar       NULL,
WFFirstActID int         NULL,
WFIsOpen smallint        NULL,
WFIsClosed smallint      NULL,
WFIsInstantiated smallint        NULL,
WFLastAct varchar        NULL,
WFLastActByName varchar  NULL,
WFLastActByRole varchar  NULL,
WFLastActTime varchar    NULL,
WFLastActID int  NULL,
WFName varchar   NULL,
WFParentWFID int         NULL,
WFPendingAct varchar     NULL,
WFPendingAction varchar  NULL,
WFPendingByName varchar  NULL,
WFPendingByRole varchar  NULL,
WFPendingName varchar    NULL,
WFPendingTime varchar    NULL,
WFPendingToName varchar  NULL,
WFPendingToRole varchar  NULL,
WFPhaseName varchar      NULL,
WFPerfName varchar       NULL,
WFPerfRole varchar       NULL,
WFStartTime varchar      NULL,
WFStateName varchar      NULL,
WFStateNumber smallint   NULL,
WFStateType varchar      NULL,
WFStatus varchar         NULL,
WFTimeAcceptance varchar         NULL,
WFTimeNegotiation varchar        NULL,
WFTimePerformance varchar        NULL,
WFTimePreparation varchar        NULL,
WFType varchar   NULL,
WFWaiting smallint       NULL,
```

```
        WFIsCustVQ smallint      NULL,
        WFIsPerfVQ smallint      NULL,
        WFIsObsVQ smallint       NULL,
        WFCOS varchar    NULL,
        lDataId int      NULL,


  constraint pk_PWF primary key
  (
    ID
  )
)
go

[..]

/*script di creazione della tabella SURR_KEY */

USE awsdup
go

if exists
 (select * from sysobjects where
 id = object_id('dbo.SURR_KEY') and sysstat & 0xf = 3)
 drop table dbo.SURR_KEY

GO

 CREATE TABLE dbo.SURR_KEY (
         ID_ADSDBBPDEFBPDLLMAPNAME int not null default 1,
         ID_CONFIGINFO int not null default 1,
         ID_DFBPADA int not null default 1,
         ID_DFBPADARES int not null default 1,
         ID_DFBPADASPEC int not null default 1,
         ID_DFBPCOMPS int not null default 1,
         ID_DFBPDLL int not null default 1,
         ID_DFBPROLE int not null default 1,
         ID_DFFIELDRAL int not null default 1,
         ID_DFWFACTCNAMES int not null default 1,
         ID_DFWFACTSDIS int not null default 1,
         ID_DFWFADA int not null default 1,
```

```
ID_DFWFADARES int not null default 1,
ID_DFWFADASPEC int not null default 1,
ID_DFWFFUP int not null default 1,
ID_DFWFIDT int not null default 1,
ID_DFWFPROTOCOL int not null default 1,
ID_DFWFSTATECUSTINFO int not null default 1,
ID_IXBINDATA int not null default 1,
ID_IXWFCOS int not null default 1,
ID_IXWFMAINTIMES int not null default 1,
ID_PAACT int not null default 1,
ID_PACT int not null default 1,
ID_PBINDATA int not null default 1,
ID_PBP int not null default 1,
ID_PDEFADTABLENAMES int not null default 1,
ID_PDEFAPPCLASS int not null default 1,
ID_PDEFAPPDATA int not null default 1,
ID_PDEFBP int not null default 1,
ID_PDEFFORMFIELD int not null default 1,
ID_PDEFPERFOBS int not null default 1,
ID_PDEFRESOURCE int not null default 1,
ID_PDEFWF int not null default 1,
ID_PDEFWFNAMES int not null default 1,
ID_PDEFWFSHNAMES int not null default 1,
ID_PGLOBALS int not null default 1,
ID_PINC int not null default 1,
ID_PNDXAPPDATA int not null default 1,
ID_PWF int not null default 1,
ID_PWFOBS int not null default 1,
ID_PWFPERF int not null default 1,
ID_TXMAILINFO int not null default 1,
ID_TXPBINDATA int not null default 1,
ID_TXPBPAPPDATA int not null default 1,
ID_TXPBPIDT int not null default 1,
ID_TXPCOS int not null default 1,
ID_TXPCYCLETIME int not null default 1,
ID_TXPTRANSFERIDT int not null default 1,
ID_TXPUSERDATA int not null default 1,
ID_TXPWFAPPDATA int not null default 1,
ID_TXPWFIDT int not null default 1,
ID_TXPWFPARTICIPANTS int not null default 1,
ID_TXQADHOC int not null default 1,
```

```
        ID_TXQBINDATA int not null default 1,
        ID_TXQBPAPPDATA int not null default 1,
        ID_TXQBPIDT int not null default 1,
        ID_TXQCOS int not null default 1,
        ID_TXQCYCLETIME int not null default 1,
        ID_TXQTRANSFERIDT int not null default 1,
        ID_TXQUSERDATA int not null default 1,
        ID_TXQWFAPPDATA int not null default 1,
        ID_TXQWFIDT int not null default 1,
        ID_TXQWFPARTICIPANTS int not null default 1,
 )
 go

/* inserimento dei primi valori */
INSERT INTO SURR_KEY (ID_DFWFIDT) values(1)
go
```

C.2.3 Esempi di trigger

Presentiamo qui due esempi di trigger su insert, come vengono generati dal DOM Builder. Il primo è il trigger sulla tabella di *aws*, il secondo sulla corrispondente di *awsdup*.

```
/****** Object:  Trigger dbo.AWS_DFWFIDT_INS ******/
/****** Script Date: 16/06/98 18.33.44 ******/

create trigger AWS_DFWFIDT_INS on DFWFIDT for insert as

begin

if @@NESTLEVEL = 1
  begin
  print 'Sto spostando i dati: sono in aws'
/* sposta dati su awsdup */
/* variabili locali -> colonne di DFWFIDT + ID */

  declare @ID int
  declare @lBPDid int
  declare @lWFDid int
  declare @lWFRoleIDTId int
```

```
declare @bIsMultiValued smallint
declare @lDefBPRoleId int
declare @iSeqNo smallint
declare @lBPRoleId int

declare @fs int
select @fs = 0

declare server1_aws_DFWFIDT cursor for select * from inserted
open server1_aws_DFWFIDT

while @fs = 0
begin
fetch next from server1_aws_DFWFIDT into
                        @lBPDid,@lWFDid,@lWFRoleIDTId,
                        @bIsMultiValued,@lDefBPRoleId,
                        @iSeqNo,@lBPRoleId

select @fs = @@FETCH_STATUS
if @fs = 0
begin
  declare server1_SURR cursor for select
                    ID_DFWFIDT from awsdup.dbo.SURR_KEY
  open server1_SURR

  fetch next from server1_SURR into @ID

  insert awsdup.dbo.DFWFIDT values (@ID,@lBPDid,
  @lWFDid,@lWFRoleIDTId,@bIsMultiValued,@lDefBPRoleId,
  @iSeqNo,@lBPRoleId)

  update awsdup.dbo.SURR_KEY set ID_DFWFIDT = @ID + 1
                             where ID_DFWFIDT = @ID
  close server1_SURR
  deallocate server1_SURR

  fetch next from server1_aws_DFWFIDT into
                          @lBPDid,@lWFDid,@lWFRoleIDTId,
                          @bIsMultiValued,@lDefBPRoleId,
                          @iSeqNo,@lBPRoleId
```

```
    select @fs = @@FETCH_STATUS

    end

  end

  close server1_aws_DFWFIDT
  deallocate server1_aws_DFWFIDT

  end   /* fine sposta dati */

else

begin

 declare @str char(10)
 print 'livello nesting diverso da 1'
 select @str = convert(char(10),@@NESTLEVEL)
 print @str

end

end

GO

/* fine */


/****** Object:  Trigger dbo.AWSDUP_DFWFIDT_INS ******/
/****** Script Date: 16/06/98 18.33.10      ******/

create trigger AWSDUP_DFWFIDT_INS on DFWFIDT for insert as

begin

if @@NESTLEVEL = 1
begin

  print 'livello nesting = 1'
```

```
declare @ID int
declare @lBPDid int
declare @lWFDid int
declare @lWFRoleIDTId int
declare @bIsMultiValued smallint
declare @lDefBPRoleId int
declare @iSeqNo smallint
declare @lBPRoleId int

declare @fs int
select @fs = 0
declare server1_awsdup_DFWFIDT cursor for select * from inserted
open server1_awsdup_DFWFIDT

fetch next from server1_awsdup_DFWFIDT into
                @ID,@lBPDid,@lWFDid,@lWFRoleIDTId,
                @bIsMultiValued,@lDefBPRoleId,@iSeqNo,
                @lBPRoleId

select @fs = @@FETCH_STATUS

while @fs = 0
begin
  insert aws.dbo.DFWFIDT values
                (@lBPDid,@lWFDid,@lWFRoleIDTId,
                 @bIsMultiValued,@lDefBPRoleId,
                 @iSeqNo,@lBPRoleId)

  update SURR_KEY set ID_DFWFIDT = @ID + 1

  fetch next from server1_awsdup_DFWFIDT into
                @ID,@lBPDid,@lWFDid,@lWFRoleIDTId,
                @bIsMultiValued,@lDefBPRoleId,@iSeqNo,
                @lBPRoleId

  select @fs = @@FETCH_STATUS

end

close server1_awsdup_DFWFIDT
deallocate server1_awsdup_DFWFIDT
```

```
end

else

begin
 declare @str char(10)
 print 'livello nesting diverso da 1'
 select @str = convert(char(10),@@NESTLEVEL)
 print @str
end

end

GO
```

Bibliografia

[1] B. Reinwald and H. Wedekind, "Automation of Control and Data flow in Distributed Application System" in *Database and Expert Systems Applications (DEXA), Proc. of the Int. Conf. in Valencia, Spain*, pag. 475-481, Berlin 1992, Springer-Verlang.

[2] G. Alonso, B. Reinwald and C. Mohan, "Distributed Data Management in Workflow Environments". *Research Issues in Data Engineering (RIDE), Proc. of the Int. Work. in Birmingham, UK*, pag. 82-90, IEEE Computer Society Press, 1997.

[3] D. J. Dietterich, "DEC Data Distributor: for Data Replication and Data Warehousing", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ACM, Minneapolis, Minnesota, May 1994.

[4] A. Gorelix, Y. Wang and M. Deppe, "Sybase Replication Server", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ACM, Minneapolis, Minnesota, May 1994.

[5] J. Gray, P. Helland, P. E. O'Neil and D. Shasha, "The Dangers of Replication and a Solution", *SIGMOD Conference*, pag. 173-182, ACM 1996.

[6] Brad Hammond, "Using Referential Integrity to Easily Define Consistent Subset Replicas", *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases"*, Morgan Kaufmann, Mumbai (Bombay), India , Sept. 1996.

[7] D. Hollingsworth, "Workflow Management Coalition: The workflow reference model", *Document WFMC-TC-1003, Workflow Management Coalition, Nov. 1994*, accessible via http://www.aiim.org/wfmc/.

[8] L. Kawell, S. Beckhradt, T. Halvorsen, R. Ozzie and I. Greif, "Replicated document management in a group communicating system", in *Proc. of*

the Conf. on Cumputer Supported Cooperative Work, CSCW(Portland, Oregon), 1988.

[9] Lotus Notes, Cambridge, MA, "Lotus Notes Realease 4 Application Developer's Guide", 1995.

[10] Lotus Notes, Cambridge, MA, "Lotus Notes Realease 4 Database Manager's Guide", 1995.

[11] Microsoft Corporation, "Microsoft SQL Server Administrators Guide: Microsoft Corporation", 1996.

[12] C. Mohan and B. Lindsay, "Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions", *in 2nd SIGACT-SIGMOD Symp. on Principles of Distributed Computing*, pag. 76-88, ACM, 1983.

[13] C. Mohan, B. Lindsay and R. Obermarck, "Transaction Management in the R* Distributed Database Management System", *ACM Trans. Database System*, pag. 378-396, N. 11, S. 4, ACM, 1986.

[14] M. T. Ozsu and P. Valduriez, "Principles of Distributed Database Systems", *Prentice Hall International Editions*, New Jersey, 1991.

[15] G. Smith et al., "Oracle's Symmetric Replication Technology and Implications for Application Design", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ACM, Minneapolis, Minnesota, May 1994.

[16] H. Stark and L. Lachal, "Ovum Elatuates Workflow", *Ovum Evaluates*, Ovum Ltd., London, 1995.

[17] "The Workflow Management Coalition Specification, Workflow Management Coalition: Terminology & Glossary", *Document WFMC-TC-1011, Workflow Management Coalition, June 1996*, accessible via http://www.aiim.org/wfmc/.

[18] Action Technology Inc., "ActionWorkflow Enterprise Series 3.0: ActionWorkflow Guide", 1993-1996.

[19] F. Casati, P. Grefen, B. Pernici, G. Pozzi and G.Sanchez, "WIDE - workflow model and architecture.", *Techical report CTIT 96-19, University of Twente, 1996.*

[20] F. Casati, S. Ceri, B. Pernici and G. Pozzi, "Conceptual modeling of workflow.", in *Proc. of 14th Object-Oriented and Entity-Relationship Approach Int. Conf.*, pages 341-354, Gold Coast, Australia, 1995. Springer-Verlang Lectures Notes in Computer Science.

[21] F. Casati, S. Ceri, B. Pernici and G. Pozzi, "Semantic workflow interoperability.", in *Proc. of 5th Extending Database Technology Int. Conf.*, pages 113-162, Springer-Verlang Lectures Notes in Computer Science, 1996.

[22] A. Bisi, "Implementazione di modelli di replica di dati per sistemi di gestione di basi di dati distribuite",*Tesi di Laurea in Ingegneria Informatica, Università di Modena, Marzo 1997.*