UNIVERSITÀ DEGLI STUDI
DI
MODENA E REGGIO EMILIA

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Integrazione fra sistemi di workflow e gestione di dati distribuiti mediante DDBMS: progetto e implementazione

Relatore:

Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di:

Daniele Scorcioni

Correlatori:

Dott. Ing. Maurizio Vincini
Dott. Ing. Domenico Beneventano

Anno Accademico 1997 - 98

Parole chiave:
Workflow
Sistema distribuito
Replica di dati
Dynamic Ownership
Distribuited Database Management System commerciale

Alla mia famiglia

RINGRAZIAMENTI

Ringrazio la Professoressa Sonia Bergamaschi per l'aiuto fornito alla realizzazione della presente tesi.
Un ringraziamento particolare va inoltre all'Ing. Maurizio Vincini per la preziosa collaborazione e la costante disponibilità.

Indice

Introduzione **1**

1 Il Workflow Management **5**
1.1 Introduzione 5
1.2 Il concetto di workflow 7
1.2.1 Definizioni 7
1.2.2 Tipologie di flussi di lavoro 7
1.3 Modellazione dei processi aziendali 11
1.3.1 Metodologie per la modellazione dei processi 13
1.3.2 L'approccio della WfMC 16
1.3.3 La ricerca: lo schema di workflow 18
1.4 Il workflow e la logica di processo 21
1.4.1 Workflow e Business Process Reengineering 23
1.5 Lo Standard 24
1.5.1 WfMC standard 24
1.5.2 MAPI Workflow Framework 31

2 I sistemi informatici per la gestione del Workflow **33**
2.1 Background 33
2.2 I sistemi per il workflow 34
2.3 Le caratteristiche dei prodotti attuali 35
2.3.1 Principali carenze dei sistemi commerciali 38
2.4 Tendenze del mercato 40
2.4.1 Interfacce aperte ed estendibili 42
2.4.2 Gli ambienti di sviluppo dei processi 43
2.4.3 Wide Area Workflow 44
2.5 La ricerca 44
2.5.1 Transactional Workflow 44
2.5.2 Dynamic Workflow 47

3 Il Workflow in ambiente distribuito 49
3.1 Architetture di rete . . . 49
3.1.1 Modello Client-Server . . . 49
3.1.2 Modello Time Sharing e Resource Sharing . . . 49
3.1.3 Architettura distribuita . . . 51
3.1.4 Vantaggi e problemi del Distributed Computing . . . 51
3.1.5 Modelli di comunicazione nei sistemi distribuiti . . . 52
3.2 Il Workflow in ambiente distribuito . . . 55
3.3 Action Workflow System . . . 57
3.3.1 Modellazione dei processi secondo ATI . . . 63
3.3.2 Note critiche su Action Workflow . . . 66

4 SQL Server 6.5: funzionalità di Distributed Computing 69
4.1 Caratteristiche generali . . . 69
4.2 Replica dei dati . . . 70
4.2.1 Componenti del sistema di replica . . . 71
4.2.2 Ruoli dei server nella replica . . . 73
4.2.3 Processo di sincronizzazione . . . 75
4.2.4 Processo di replica delle pubblicazioni . . . 77
4.2.5 Modalità di subscription: PUSH e PULL . . . 78
4.2.6 Tipologie di replica . . . 79
4.2.7 Replica di dati di tipo text ed image . . . 81
4.2.8 Replica e ODBC . . . 81
4.2.9 L'ambiente software di implementazione . . . 82
4.2.10 Transact-SQL . . . 83
4.2.11 Stored Procedures utilizzabili nella Replica . . . 85
4.2.12 Tabelle di sistema utilizzate nella Replica . . . 89
4.3 Microsoft Distributed Transaction Coordinator . . . 92
4.3.1 Scenari d'esecuzione - configurazione locale . . . 93
4.3.2 Transazioni distribuite . . . 98
4.3.3 Recovery dell'MS DTC . . . 101
4.4 Integrità dei dati e controllo della concorrenza in SQL Server 6.5 . . . 103
4.4.1 High Speed Locking . . . 103
4.4.2 Automatica prevenzione, individuazione e correzione dei deadlock . . . 104
4.4.3 Locking a livello di riga e di pagina . . . 105
4.4.4 Controllo della concorrenza . . . 105
4.4.5 Tipi d'integrità dei dati . . . 106
4.5 Caratteristiche di rete di un ambiente SQL Server . . . 107
4.5.1 Protocolli di rete e dimensione dei pacchetti . . . 107

4.5.2 Protocolli di rete: Named Pipes, NWLink IPX/SPX e TCP/IP 108

5 Progetto di distribuzione di Action Workflow 113
5.1 Tipologie di distribuzione di un WFMS 113
5.2 Action Workflow: da sistema client/server a sistema distribuito 114
5.2.1 Replica del database *aws* 117
5.2.2 Problemi della replica bidirezionale 121
5.2.3 Modifiche alla configurazione di replica 122
5.3 Integrazione ad Action Workflow di una gestione di dati distribuiti 126
5.3.1 Dynamic Ownership Model 126
5.3.2 Il sistema risultante 128
5.3.3 Setup del sistema 129

Conclusioni 132

A Esempi di script T-SQL per SQL Server 6.5 135
A.1 Script di modifica del database *aws* 135
A.1.1 Aggiunta delle chiavi primarie 135
A.1.2 Generazione tabelle *awsdup* 136
A.1.3 Tabelle di supporto alla replica 140
A.2 Installazione Trigger in *aws* 143
A.3 Script d'installazione della replica di *aws* e *awsdup* 147

B Esempio di progetto di un database distribuito 151
B.1 Descrizione della realtà da modellare 151
B.2 Distribution Design 152
B.2.1 Query e volume dei dati 154
B.2.2 Transazioni in SQL 155
B.2.3 Frammentazione orizzontale 156
B.2.4 Allocazione dei frammenti 156
B.2.5 Costo delle transazioni 157
B.2.6 Allocazione non ridondante 159
B.2.7 Copie multiple 159
B.2.8 Schema di allocazione finale 162
B.3 Implementazione del database su SQL Server 6.5 162

Bibliografia 167

Elenco delle figure

1.1 Un tipico processo strutturato: un workflow 6
1.2 Un tipico processo non strutturato: un Workgroup 6
1.3 Ad hoc workflow : revisione di una relazione 8
1.4 Administrative workflow : revisione di una relazione 9
1.5 Production workflow : reclamo pagamenti in un'assicurazione . 10
1.6 Caratteristiche del workflow 10
1.7 L'azienda nella sua visione verticale (per funzioni) 11
1.8 L'azienda nella sua visione orizzontale (per processi) 11
1.9 Esplosione di un workflow nelle sue componenti (task) e norme (regole di interrelazione). 12
1.10 La comunicazione per Action Model 14
1.11 Il Workflow per la fornitura di materiali. 15
1.12 Il Workflow per la fornitura di materiali. 15
1.13 Un esempio di Activity Net 16
1.14 Alternanza di attività: OR-Split e OR-Join 17
1.15 Attività in parallelo: AND-Split e AND-Join 17
1.16 Iterazione di attività 17
1.17 Il task di workflow 18
1.18 I simboli grafici usati nella rappresentazione del workflow . . . 19
1.19 Supertask e multitask 20
1.20 Struttura di un generico prodotto di Workflow (fonte WfMC) . 23
1.21 Workflow Reference Model - Componenti e interfacce 25
1.22 Invoked Application Interface. 28
1.23 Workflow interoperability interface. 29
1.24 Systems Administration & Monitoring Interface. 30

2.1 Le tecnologie rispetto alle attività umane e automatizzate . . . 36
2.2 Spesa mondiale (milioni di $) in prodotti e servizi di workflow. 41
2.3 Spesa mondiale (milioni di $) in prodotti di workflow per area geografica. 41

3.1 Programmazione centralizzata 53

3.2 Schema d'esempio di una *RPC* 53
3.3 Schema di una chiamata ad una procedura remota 54
3.4 I componenti funzionali di ActionWorkflow 58
3.5 Schermata dell'Action Workflow Administrator 60
3.6 Architettura di Action Workflow System 61
3.7 La mappa della Call Tracking Application 62
3.8 Workflow Loopsecondo ATI 63

4.1 Replica dei dati unidirezionale 71
4.2 I componenti del sistema di replica di SQL Server 72
4.3 Partizione verticale 74
4.4 Partizione orizzontale 74
4.5 Partizione mista 75
4.6 Tipologia central publisher 79
4.7 Tipologia Publishing Subscriber 80
4.8 Tipologia central subscriber 81
4.9 Tipologia Publisher/Subcriber 82
4.10 Tipologia multiple publisher of one table 83
4.11 Diagramma delle tabelle di sistema 91
4.12 Esempio di sistema misto: distribuito e client/server 92
4.13 Implementazione del two-phase commit protocol nel MS-DTC 93
4.14 Ruolo delle applicazioni 94
4.15 Ruolo delle applicazioni: *Commit* e *Abort* 94
4.16 Ruolo di un Resource Manager 95
4.17 Ruolo di un Resource Manager: esempio di `insert`, `delete` e `update` di record 96
4.18 Ruolo di un Resource Manager: **Commit** 96
4.19 Ruolo di un Resource Manager: **Commit** 97
4.20 **Commit**: fasi d'interazione tra Resource e Transaction manager 97
4.21 Transaction Manager: **Commit** e scrittura su log 98
4.22 MS DTC in ambiente distribuito 99
4.23 MS DTC in ambiente distribuito 100
4.24 MS DTC in ambiente distribuito: Restart del sistema 102
4.25 Recovery nel MS-DTC 103
4.26 Prestazioni SQL Server; fonte Compaq Laboratories 108
4.27 Prestazioni SQL Server; fonte Compaq Laboratories 109

5.1 Action Workflow: configurazione Client/Server 115
5.2 Action Workflow in configurazione distribuita 116
5.3 Replica unidirezionale di *aws* 118
5.4 Replica bidirezionale di *aws* 119

5.5 Il funzionamento dei trigger per la replica di *aws* 120
5.6 Replica bidirezionale con tabelle aventi partizioni orizzontali . 121
5.7 Replica bidirezionale: schema di chiamata a procedura remota 122
5.8 DST di una pratica legale attraverso i vari organi legislativi . . 128

B.1 Configurazione della rete di server 153
B.2 Schema di allocazione finale 162

Introduzione

Al giorno d'oggi le aziende devono confrontarsi con un mercato sempre più competitivo e globalizzato, cercando di ridurre i costi, sviluppare rapidamente nuovi servizi e prodotti e gestire problemi di flessibilità ed efficienza delle proprie strutture organizzative: una funzionale gestione dei processi e delle attività aziendali, è uno dei temi che suscita da sempre un crescente interesse nelle imprese che cercano soluzioni a queste problematiche.

Da rappresentazioni matematico/statistiche dei numerosi modelli delle attività lavorative (ad esempio il PERT o il CPM), si è giunti in questi ultimi anni ad una soluzione operativa mediante il controllo stretto dei flussi informativi che sono accessori od oggetto primo delle stesse attività: le principali soluzioni sono il *Business Process Re-Engineering* (che, sinteticamente, effettua un'analisi per verificare la corretta finalizzazione delle attività rispetto all'obiettivo aziendale, solitamente il *business*) e il *Groupware* nelle sue due componenti di *Workgroup* e *Workflow*.

In particolare il *Workflow Management* è una tecnologia "giovane", ma in forte sviluppo, la cui caratteristica principale è l'automazione di processi, coinvolgendo una combinazione tra attività umane e automatizzate, particolarmente quelle riguardanti applicazioni e strumenti informatici.

I principali benefici di questa tecnologia sono:

- organizzazione, schedulazione, controllo e monitoraggio dei processi;
- un valido aiuto nel capire/migliorare i processi (analisi, simulazione e reingenierizzazione);
- riduzione del lavoro su carta;
- standardizzazione delle procedure aziendali.

I primi sistemi per la gestione del workflow erano applicazioni "monolitiche" riguardanti una specifica area applicativa, come ad esempio la gestione delle immagini o dei documenti. La novità principale della seconda generazione di WFMS (Workflow Management System) è l'aggiunta di un generico

"*workflow engine*", cioè di un "motore informatico" che fornisce una robusta e stabile infrastruttura alla gestione delle attività e processi; il progetto e la definizione dei flussi di lavoro viene invece realizzata separatamente, tipicamente attraverso interfacce grafiche.

Denominatore comune della maggioranza dei sistemi di Workflow Management, rimane però quella di aver implementato il controllo del flusso dei processi, trascurando gli aspetti riguardanti il flusso dei dati; questa carenza appare evidente soprattutto in applicazioni di workflow che elaborano grandi quantità di informazioni (applicazioni *data-intensive*), dove diventa necessaria una loro gestione efficiente e corretta.

Un'altra caratteristica comune di questi sistemi, è l'essere basati sul paradigma client/server; questo può essere un limite, in quanto le imprese tendono sempre più, sia a decentralizzarsi in unità produttive dislocate in luoghi diversi, sia a decentralizzare determinate funzioni favorendo l'elaborazione locale; in questi casi le applicazioni distribuite sono molto utilizzate nei loro sistemi informativi: un esempio sono i DBMS distribuiti, in cui la replica dei dati in tutte le filiali, favorisce un'elaborazione locale delle informazioni.

Il lavoro svolto in questa tesi si propone quindi di integrare la tecnologia dei DBMS distribuiti a quella dei WFMS, allo scopo di dare una soluzione alle carenze appena descritte, che in sintesi possono elencarsi in:

- limiti imposti dal paradigma client/server, quali la stretta dipendenza dal server centrale e l'assenza di elaborazione locale delle informazioni;

- limitata o inesistente gestione e definizione del flusso dei dati dell'applicazione.

Il progetto è stato realizzato utilizzando due sistemi commerciali: il DBMS *Microsoft SQL Server 6.5* e il WFMS *Action Workflow System* della Action Technology.

Il primo problema è stato superato rendendo Action Workflow un sistema distribuito: grazie al sistema di replica di SQL Server, il database che Action Workflow utilizza per memorizzare sia le definizioni che le azioni eseguite sulle istanze dei processi, è stato duplicato e sincronizzato su più server, e il *workflow engine* è stato installato su tutte le macchine interessate alla distribuzione. In questo modo ogni azione compiuta dall'utente su un'istanza di processo, viene elaborata solo in locale; sarà poi il sistema di replica di SQL Server a far pervenire anche agli altri siti remoti tutte le modifiche locali.

Per dare al programma anche le funzionalità di gestione dei dati, si è pensato d'integrare al sistema le funzionalità di *data-handling* di SQL Server. In particolare, per la gestione di dati distribuiti, si è utilizzato un modello di

replica sviluppato e implementato su SQL Server in questa università, il *Dynamic Ownership Model* (DOM): in sintesi prevede che solo un determinato soggetto ("ruolo") in un certo istante sia il "propietario" del dato, e quindi solo lui lo possa modificare o cancellare, mentre tutti gli altri soggetti lo possano solo leggere. Questo meccanismo automatico per la modifica dinamica del soggetto possessore del dato, si presta molto bene ad integrare un WFMS, dove per definizione i dati sono processati dinamicamente da molte stazioni di lavoro.

Il sistema risultante utilizza quindi le funzionalità di gestione dei processi dei WFMS, e le capacità di *data-handling* dei DDBMS.

Il contenuto della tesi si articola in questo modo. Nel primo capitolo vengono presentati i concetti generali sul workflow: definizioni, standard e tipologie di modellazione dei processi. Il secondo capitolo descrive le caratteristiche generali dei sistemi in commercio per il workflow management e si conclude con una panoramica sugli argomenti di ricerca in questo campo. Il terzo capitolo tratta del workflow in ambiente distribuito e offre una descrizione tecnica su Action Workflow. Nel quarto capitolo viene presentato il DBMS Microsoft SQL Server 6.5, dalle funzionalità di *distributed computing*, al supporto per l'integrità dei dati e controllo della concorrenza. La tesi si conclude con il capitolo che descrive il progetto di supporto della tecnologia dei DMBS distributi ai sistemi per il workflow, dalle fasi di configurazione, all'installazione del sistema risultante.

Capitolo 1

Il Workflow Management

1.1 Introduzione

Il Workflow Management (WFM) rappresenta il coordinamento, controllo e comunicazione automatizzata di processi, procedure o flussi di lavoro che regolano la condivisione fra più "attori" di documenti, compiti da svolgere e informazioni. Di Workflow Management si parla da anni e, seppure sotto spoglie diverse, sistemi per la gestione dell'operatività sono pressoché sempre esistiti nell'infinito magazzino dell'IT. Infatti con gli inizi del periodo industriale la ricerca del miglior coordinamento è divenuta l'attività specifica di taluni individui che, sotto le vesti di filosofi o semplici tecnici, cercavano di scoprire il ritmo giusto per dirigere l'orchestra degli sforzi. La complessità delle situazioni reali aveva portato allo sviluppo di modelli applicabili tanto semplificati da essere validi solo per situazioni locali di ridotte dimensioni come gli uffici o per ambiti tipicamente manifatturieri. Inzialmente i processi erano eseguiti interamente da persone che manipolavano oggetti fisici. Con l'introduzione dell'Information Technology, i processi sul posto di lavoro furono totalmente o parzialmente automatizzati dai sistemi informatici.

In origine i sistemi per la gestione dei workflow erano embrioni di funzionalità sparsi e non organizzati che, in alcune applicazioni specifiche, consentivano semplicemente di coordinare o di fornire indicazioni per il coordinamento di operatori diversi coinvolti in aspetti distinti del medesimo obiettivo lavorativo.

A partire dallo scorso decennio le prime funzionalità di Workflow Management sono state analizzate, perfezionate, ampliate e, soprattutto, aggregate in prodotti dai connotati ben specifici.
In questo periodo si è iniziato a parlare di Workflow Management sia in termini di esigenza, sia come una nuova classe di prodotti, esplicitando dun-

que in modo forte la volontà di imporre alle attività una *causalità temporale deterministica*.

Grazie alla capillare diffusione degli strumenti informatici nel periodo tra gli anni '80 e gli anni '90, e sfruttando l'incredibile sviluppo delle tecnologie di interconnetività e comunicazione, i sistemi di Workflow Management hanno assunto una connotazione sempre più definita, non tanto in termini di omogeneizzazione delle funzionalità quanto di differenziazione da altri strumenti affini, quali ad esempio le applicazioni di "Workgroup" con le quali spesso, ancora oggi, i sistemi di Workflow vengono confusi.

Dunque, se si cerca di dare uno sguardo appena sotto la superficie di quei prodotti che ci sembra aver individuato come "utili per il coordinamento delle componenti eterogenee di attività complesse e spesso non ben identificate nei propri confini ", scopriamo un insieme di applicazioni talmente vasto e diversificato che sembra impossibile essere stato unificato sotto lo stesso termine. Se poi estendiamo questo insieme anche ai prodotti che presentano funzionalità "orientate al Workflow", pur senza vantarne un utilizzo così specifico, i labili confini svaniscono del tutto.

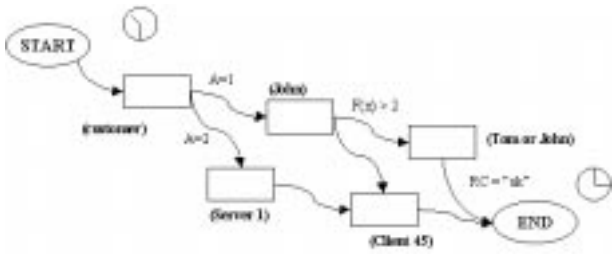


Figura 1.1: Un tipico processo strutturato: un workflow

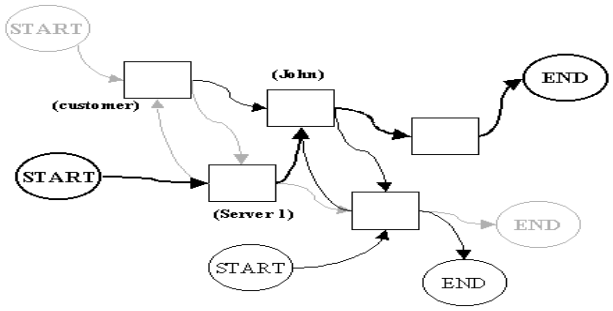


Figura 1.2: Un tipico processo non strutturato: un Workgroup

1.2 Il concetto di workflow

1.2.1 Definizioni

Non vi è molto accordo su cosa sia il workflow e quali caratteristiche un *Workflow Management System* (WFMS) debba avere. Con il termine "workflow", che spesso è usato inopportunamente, ci si può riferire a un processo aziendale, o alle specifiche di un processo generico, a un software che implementi e automatizzi un processo, oppure a un software che semplicemente supporti il coordinamento e la collaborazione delle persone che creano il processo stesso.

Le aziende produttrici di software per il Workflow Management e le organizzazioni che ricercano in questo campo propongono tutte una propria definizione; il workflow :

- consiste nel definire ed eseguire una serie di task mentre è in esecuzione un business process. [Concordium97]
- è il processo tramite il quale task individuali concorrono per completare una transazione (un business process ben definito) all'interno di un'azienda. [Action Technology95]
- è un meccanismo grazie al quale si possono attivare le procedure di business process reengineering. [PeopleSoft Inc.]
- va oltre l'instradamento (ad es. muovendo informazioni tra utenti e sistemi informatici) integrando informazioni da una varietà di sorgenti. [Wang Laboratories]
- è 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. [Workflow Management Coalition]

1.2.2 Tipologie di flussi di lavoro

Accanto alle definizioni di workflow appena elencate, abbiamo (cfr. [27]) descritti tre tipi di flussi di lavoro : *ad hoc*, *administrative*, e *production*.
I termini di distinzione sono :

- la ripetitività e la predicibilità dei workflow e delle attività;
- come il workflow è inizializzato e controllato (ad esempio se controllato da persone o automatizzato);

- i requisiti per le funzionalità dei WFMS.

Il workflow *ad hoc* è riferito alle procedure d'ufficio, come produrre documenti o proposte di vendita, dove non vi sia un modello prestabilito di passaggio delle informazioni. Le attività tipicamente riguardano coordinazione, collaborazione e co-decisione delle persone e non sono quindi automatizzate. Tipicamente questi tipi di flussi di lavoro coinvolgono ristretti gruppi di professionisti e hanno come fine progetti a breve termine. WFMS che supportano tali workflow devono funzionalmente facilitare la coordinazione, la collaborazione e la co-decisione tra le persone; questi sistemi per il workflow non sono "mission-critical", cioè possibili errori da parte di questi sistemi non interferiscono significativamente con il complessivo processo aziendale. L'infrastruttura informatica che ospita questi sistemi varia tra sistemi di posta elettronica avanzati, sistemi di calendaring e di videoconferenza; per memorizzare informazioni si sfruttano solitamente database locali.

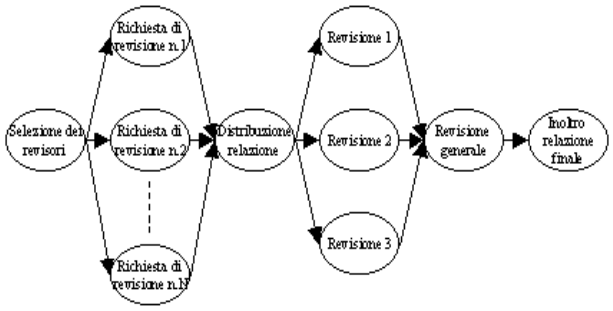


Figura 1.3: Ad hoc workflow : revisione di una relazione

La Fig. 1.3 illustra un workflow ad hoc che descrive la revisione di una relazione aziendale; il processo si articola nelle seguenti fasi: selezione dei revisori, distribuzione della relazione ai revisori, produzione delle revisioni e collaborazione nel produrre la relazione finale da inoltrare poi agli autori iniziali. Abbiamo qunidi tutte le caratteristiche di un workflow ad hoc : (1) negoziazione per la selezione dei revisori e (2) collaborazione per produrre il documento finale. Un'ultima caratteristica di questo flusso di lavoro, e in genere di tutti i workflow ad hoc, è il settare procedure per produrre ed elaborare attività di un solo tipo. Gli *administrative* workflows riguardano processi ripetitivi e predicibili con una semplice coordinazione tra le attività, come l'inoltrameno di un rapporto spese in azienda attraverso un processo di autorizzazione. L'ordine e la collaborazione fra i task può essere automatizzato. WFMS che supportano tali processi trattano informazioni

"semplici", che non necessitano di instradamenti complessi. Anche gli administrative WFMS sono non mission-critical e l'infrastruttura tecnologica correntemente utilizzata è basata prevalentemente su e-mail. Considerando ancora il processo precedente, i revisori ora sono noti in anticipo (ad es. tutti i revisori sono utilizzati per tutti i tipi di relazioni), essi non collaborano tra loro ma producono modifiche individuali che sono elaborate dall'autore che prende una decisione finale.

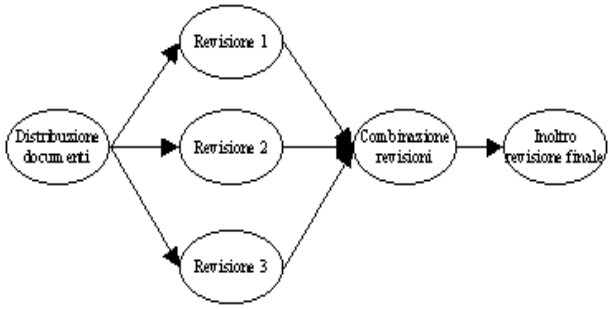


Figura 1.4: Administrative workflow : revisione di una relazione

In base a queste ipotesi l'esempio di Fig. 1.4 rappresenta un administrative workflow. In questi tipi di workflow gli utenti sono attivamente invitati a eseguire le proprie attività. Mentre nel caso precedente i revisori dovevano accedere al WFMS per determinare se il lavoro era completato, in questa situazione essi ricevono un e-mail con le istruzioni di revisione del documento e i commenti degli altri revisori. Quando il lavoro è completato, esso viene automaticamente inoltrato all'autore il quale è avvisato quando tutte le revisioni sono terminate.

I *production worflows* si riferiscono a ripetitivi e predicibili processi aziendali come richieste di prestito nelle banche o reclami alle assicurazioni dove, a differenza degli administrative workflow, è richiesta una comlessa gestione delle informazioni, da reperire in molteplici e complessi sistemi informativi. L'ordine e il coordinamento dei task è automatizzato come nel caso precedente ma in modo più complicato causa la complessità dei processi e l'accesso a molteplici sistemi informatici per accedere ai dati necessari al supporto delle decisioni. In queta situazione i WFMS devono facilitare la definizione delle dipendenze tra i task e controllarne l'esecuzione con un minimo intervento umano; essi sono spesso vitali per l'intero processo (mission-critical systems).

Un esempio di production workflow potrebbe essere quello in Fig. 1.5 dove è rappresentato il processo di valutazione di una richiesta di pagamento in un'assicurazione : il modulo di richiesta è prima digitalizzato (ad esempio con scanner) e memorizzato manualmente nel database delle richieste;

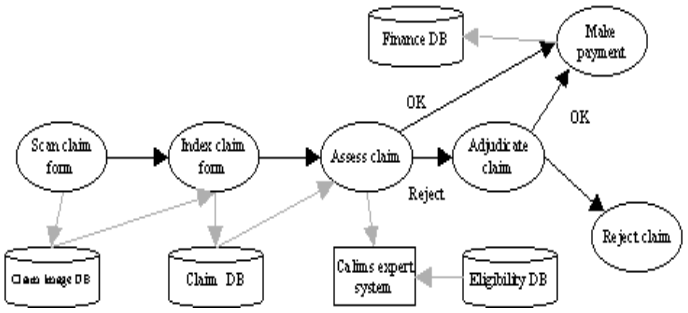


Figura 1.5: Production workflow : reclamo pagamenti in un'assicurazione

quindi viene indicizzato in un database relazionale; l'informazione che se ne ricava è successivamente analizzata da un processo automatico di valutazione ("Assess Claim"), che grazie a un sistema esperto che sfrutta un database "di idoneità" ("eligibility database"), determina se il pagamento deve essere eseguito. Se la richiesta è respinta, si discute ancora con il cliente se eseguire i pagamenti o rigettare del tutto il reclamo. Se il pagamento viene accettato, il processo "make payment" accede al "finance database" e registra il tutto. Le differenze significative tra questi tre tipi di workflow sono : (a) l'interazione dei sistemi informatici con i *business process*, e (b) l'uso di elaborazione dei processi automatizzata. La relazione tra ad hoc, admninistrative e production workflow è rappresentata in Fig. 1.6, dove la struttura dei task è messa in relazione con la loro complessità.

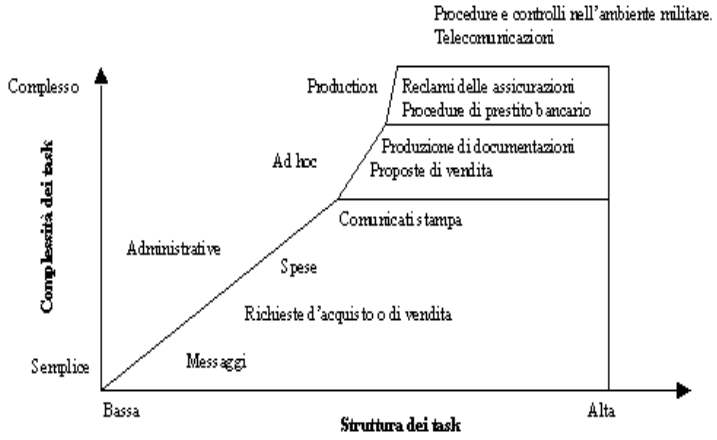


Figura 1.6: Caratteristiche del workflow

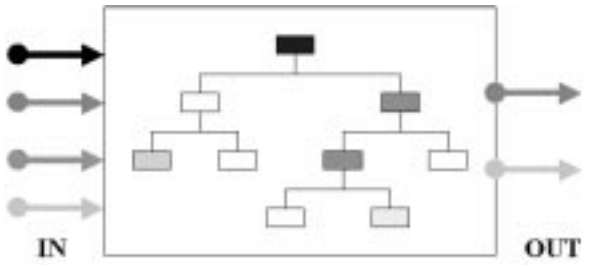


Figura 1.7: L'azienda nella sua visione verticale (per funzioni)

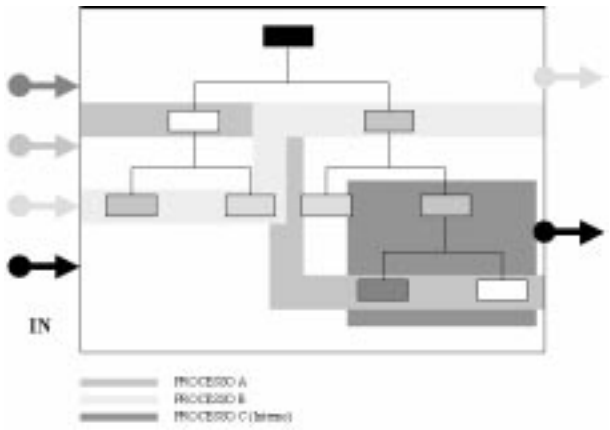


Figura 1.8: L'azienda nella sua visione orizzontale (per processi)

1.3 Modellazione dei processi aziendali

Per la maggior parte dei prodotti di Workflow l'ente macroscopico è il "processo" quale insieme di "attività". A sua volta il processo può essere componente di un processo più esteso (in generale allora si definisce "sottoproceso") fino ad iterare un numero di volte qualsiasi questo annidamento.

Le attività (talvolta indicate col nome di "task" o "job unit") sono le unità minime di lavoro all'interno di un processo. Un'attività può essere la memorizzazione di un dato, lo spostamento di un file, la redazione di un documento, la generazione di un codice seriale, un calcolo, una decisione; in generale, dunque, un compito non frazionabile se non scendendo ad un dettaglio che porti in conto l'operatività minuta (il *come*) e, quindi, funzionalità di programmazione informatica o elaborazione umana.

Una volta delineato il processo, questo viene modellizzato mediante un tool accessorio del sistema e rappresentato in forma grafica per consentirne

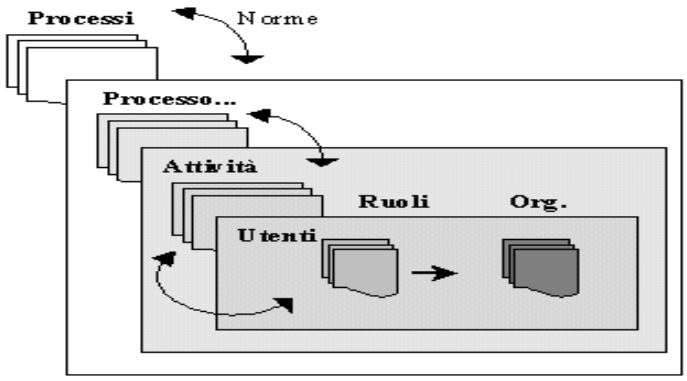


Figura 1.9: Esplosione di un workflow nelle sue componenti (task) e norme (regole di interrelazione).

una visione rapidamente interpretabile e condivisibile anche da i non addetti ai lavori. Il termine di questa fase è stato definito il *cosa*.

Identificando il *cosa* implicitamente viene stabilito anche il *quando*, e cioè tutto l'insieme di propedeuticità che le varie attività mantengono le une rispetto alle altre (ad esempio l'attività X non può iniziare se non sono terminate la Y e la Z), rispetto ai termini temporali assoluti (ad es. il giorno della settimana) e anche i termini temporali relativi od ad eventi asincroni esterni (ad esempio altri processi).

In questa fase vengono *portati* dentro il modello del processo anche gli altri componenti che risulterebbero esterni in termini funzionali (in pratica, le integrazioni con altri sistemi o procedure). È proprio in questa fase che la mancanza di cultura, nella visione orizzontale dei flussi, si rivela in errori macroscopici che talvolta inficiano i risultati finali; in pratica l'errore più frequente consiste nel replicare meramente le strutture funzionali e le consuetudini operative in assenza di una sufficiente astrazione della rappresentazione della realtà attuativa.

Per non avere pericolose perdite di consistenza del modello deve essere rappresentato tutto ciò che partecipa al processo, pena anche il decadimento delle potenzialità di controllo e monitoraggio complessivo sul processo stesso.

Un sistema di workflow correttamente realizzato è in ogni momento in grado di fornire una vista sullo stato corrente e mostrare la sequenza degli eventi che hanno portato a quello stato. Questa prerogativa, in molti casi, ha rappresentato una spinta all'adozione di un sistema piuttosto che un altro, all'inevitabile atto della scelta.

Al termine della fase di definizione delle regole è evidente e definito il

come. Il passo successivo consiste nell'identificazione del *chi*. Ossia : *chi deve svolgere l'attività X se quelle Y e Z sono terminate?*

Per rispondere a queste domande il modellatore scopre di dover definire delle classi di utenti caratterizzandole in base ad attributi comuni. Queste classi vengono definite usualmente "ruoli". La gran parte degli attuali strumenti di workflow è flessibile al punto da lasciare definire dinamicamente i ruoli in base al processo, nel senso che chi per il processo "*A*" svolge un determinato "ruolo" potrà anche svolgerne un altro per lo stesso o per altri processi. È possibile vedere il "ruolo" come una sorta di mansionario legato al singolo specifico processo. Il ruolo è un oggetto della visione orizzontale, appartiene infatti al dominio del processo e non a quello della funzione aziendale. Parallelamente ai "ruoli" è frequente poter definire una ulteriore classificazione degli utenti in base alla loro appartenenza a strutture funzionali dell'ambito reale; queste seconde classi prendono il nome di "organizzazioni" o "gruppi di lavoro". È evidente che la visione verticale tipica della struttura funzionale dell'azienda si riflette nelle "organizzazioni".

Sia i ruoli sia le organizzazioni possono essere gerarchizzati, nel senso che è possibile sotto-ruoli e sotto-organizzazioni. Per tutti possono poi essere esplicitate figure priviligiate come i coordinatori, i responsabili, i leader, ecc. . Le classificazioni viste determinano gli attributi in base ai quali gli utenti si troveranno assegnatari o responsabili delle attività del workflow.

1.3.1 Metodologie per la modellazione dei processi

Ci sono due categorie base per la modellazione dei processi: la metodologia basata sulla comunicazione (*communication-based*) e quella basata sulle attività (*activity-based*).

Metodologia *communication-based*

La metodologia *communication-based* è trattata da Winograd/Flores (cfr. [13]). Questa metodologia assume che lo scopo della Business process re-engineering è incrementare la soddisfazione del cliente. Vi sono quattro fasi di ogni azione in un processo di workflow (illustrate in Fig. 1.10), basate sulla comunicazione tra "cliente" (*customer*) ed "esecutore" (*performer*):

1. *preparazione* - un cliente richiede l'esecuzione di un'azione o un esecutore propone di eseguirne alcune;

2. *negoziazione* - sia il cliente che l'esecutore accettano di eseguire una determinata azione e definiscono i termini di soddisfazione;

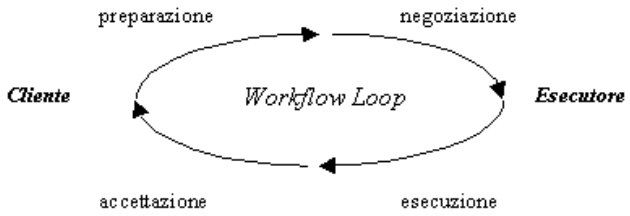


Figura 1.10: La comunicazione per Action Model

3. *esecuzione* - l'azione viene eseguita in accordo ai termini prestabiliti;

4. *accettazione* - il cliente riporta la propria soddisfazione (o disapprovazione) riguardo all'azione eseguita.

Ogni ciclo di workflow (*Workflow loop*) tra un cliente ed un esecutore può essere unito da altri cicli per completare un processo aziendale. L'esecutore in un ciclo può essere esecutore in un altro. Il business process risultante rivela la "rete sociale" in cui un gruppo di persone, occupando vari ruoli, mira ad eseguire un certo processo.

L'esempio in Fig. 1.11 mostra un business process per il rifornimento di materiali. Il workflow loop principale (procurare materiale) richiede alcuni loop secondari durante la fase d'esecuzione (verica stato, ottenere un'offerta, piazzare l'ordine). In particolare, colui che cerca i materiali (l'investigatore), richiede dei servizi dall'ufficio apposito (ufficio forniture). Nell'esecuzione della fornitura, l'ufficio interroga i propri conti per verificare quelli dell'acquirente. L'ufficio poi contatta i venditori per le offerte, e alla fine sceglie un venditore per piazzare l'ordine. Il Workflow è completato quando l'ufficio forniture conferma all'investigatore la fornitura del materiale. Si può notare che l'esecutore nel loop principale è il cliente in quello secondario e che le specifiche di workflow, usando questa metodologia, non indicano quali attività possono essere eseguite in parallelo, o se esse sono azioni condizionali o alternative. Poiché queta metodologia assume che l'obiettivo della BPR è incrementare la soddisfazione del cliente, l'attenzione è focalizzata sul cliente. Comunque ci possono essere processi aziendali dove la soddisfazione del cliente non è importante, quali i processi per la riduzione dei costi di un sistema informativo o dello spreco di materiali; per questi tipi di processo quindi la metodologia appena descritta non è molto adatta.

Lo strumento "ActionWorkflow Analyst" da Action Technologies, da noi utilizzato, è basato proprio sul modello di Winograd/Flores appena descritto.

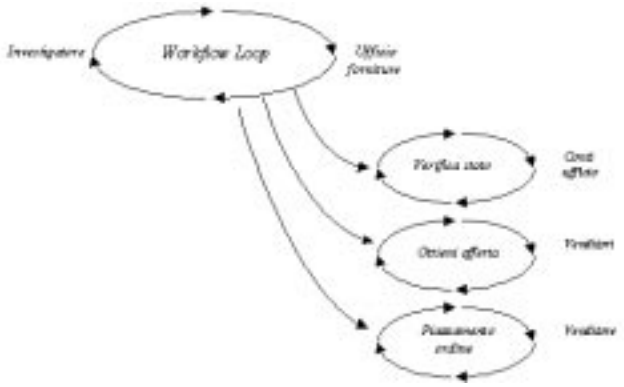


Figura 1.11: Il Workflow per la fornitura di materiali.

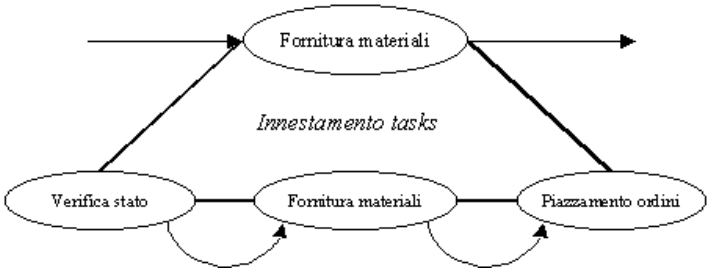


Figura 1.12: Il Workflow per la fornitura di materiali.

Metodologia *activity-based*

Le metodologie basate sulle attività focalizzano l'attenzione sulla modellazione del lavoro piuttosto che sugli sforzi delle persone. Per esempio, si consideri il workflow in Fig. 1.12 dove la "fornitura di materiali" è composta da diversi task. Le freccie indicano la natura della sequenza della mappa del processo; questo workflow può essere a sua volta un task in un altro processo e via via i processi possono essere innestati come task fino ad una qualsiasi profondità. A differenza del modello basato sulla comunicazione, in questo caso non è data importanza all'incremento della soddisfazione del cliente.

I due modelli possono essere combinati quando i corrispondenti obiettivi di process re-engineering sono compatibili (ad esempio soddisfare il cliente minimizzando i task di workflow e i ruoli).

Le metodologie Object-oriented, quali quelle proposte da Rumbaugh e Jacobson (cfr. [15] e [16]) possono essere utili nel definire le specifiche di workflow (da cui derivare l'implementazione). Per esempio, Jacobson in [16] descrive come (1) identificare gli oggetti che corrispondono agli "attori" (i

"ruoli" del workflow), (2) identificare le dipendenze tra questi oggetti, (3) usare tecniche quali l'ereditarietà per organizzare le specifiche degli oggetti, e (4) descrivere "casi pratici", che sono essenzialmente una sequenza di task necessari per completare qualche business process. Comunque, l'approccio ad oggetti non fornisce un esplicito supporto (come un modello di workflow) per la modellazione dei processi.

1.3.2 L'approccio della WfMC

In base alla definizione proposta dalla Workflow Management Coaltion (vd. [700]), un processo è un insieme coordinato di attività che si susseguono in parallelo o sequenzialmente per raggiungere uno scopo comune. Le attività e la loro interazione possono essere rappresentate graficamente come qui di seguito descritto.

Activity Net. L'attività (o task) è rappresentata da un rettangolo con un'etichetta che ne rappresenta il nome. L'interazione tra le attività è rappresentata da archi orientati; l'insieme di archi e rettangoli modella l'*Activity Net* (in breve AN).

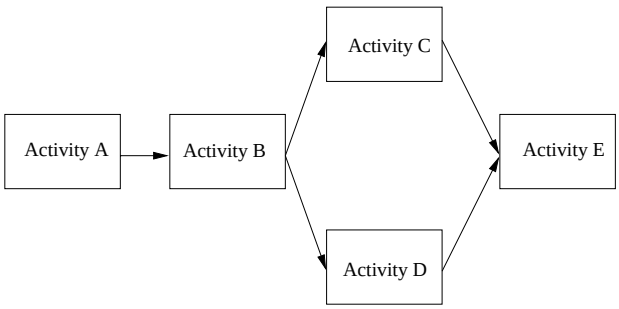


Figura 1.13: Un esempio di Activity Net

OR-Split e OR-Join. L'"OR-Split" vuole rappresentare la separazione dell'esecuzione del processo in due o più percorsi (attività) alternativi, in base a determinate condizioni da porre sugli archi. L'esecuzione potrà percorrere uno solo dei percorsi alternativi.

L'"OR-Join" modella la convergenza di due o più attività (a seconda dell'esecuzione del processo) in un'unica attività a seconda delle condizioni di Join poste sugli archi.

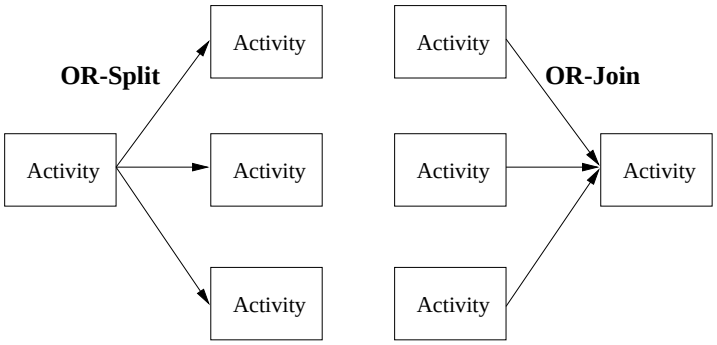


Figura 1.14: Alternanza di attività: OR-Split e OR-Join

AND-Split e AND-Join. L'"AND-Split" rappresenta la suddivisione del percorso d'esecuzione del processo in sotto-percorsi paralleli: terminata l'attività iniziale, devono essere eseguite tutte le attività seguenti.

L'"AND-Join" modella la convergenza di due o più percorsi d'esecuzione in un unico percorso: solo quando sono terminati tutti le attività paralleli precedenti, si può eseguire l'attività d'"arrivo".

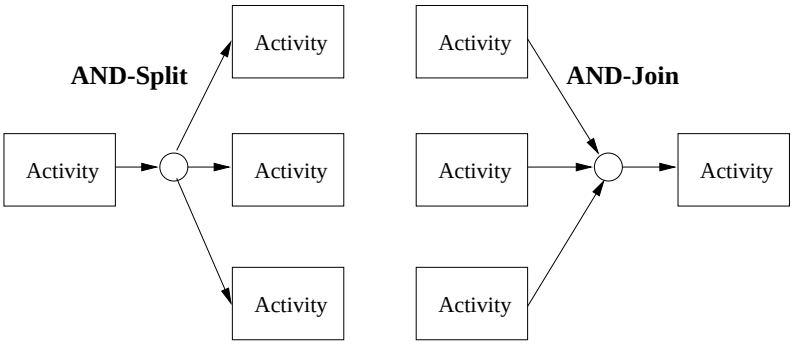


Figura 1.15: Attività in parallelo: AND-Split e AND-Join

Iterazione. L'iterazione tra più attività è rappresentata in Fig.1.16; vuole modellare l'esecuzione ciclica di certe attività fino a che certe condizioni non sono soddisfatte.

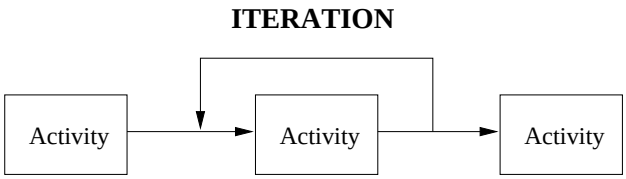


Figura 1.16: Iterazione di attività

1.3.3 La ricerca: lo schema di workflow

In questo approccio quello che era l'AN (*Activity Net*), 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 1.17. In particolare, un task è caratterizzato da:

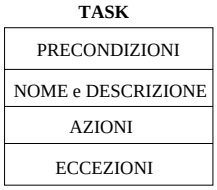


Figura 1.17: Il task di workflow

- *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;
- *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 1.18. 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*.

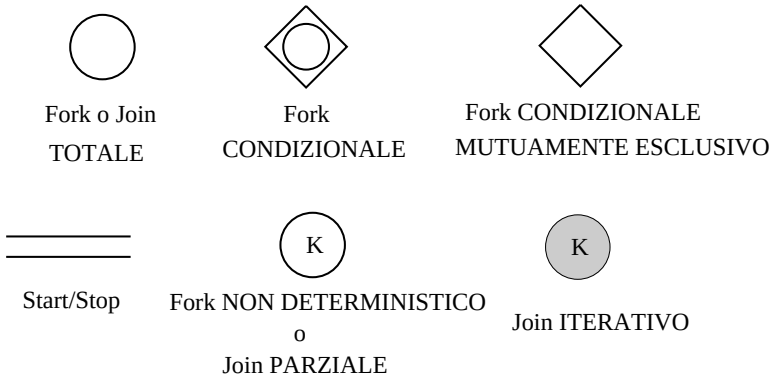


Figura 1.18: I simboli grafici usati nella rappresentazione del workflow

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 1.18.

- **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).

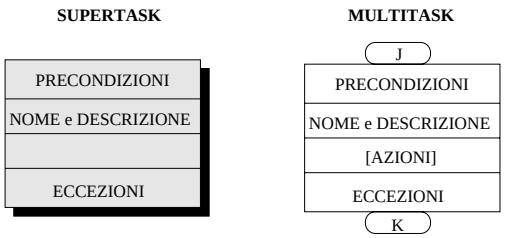


Figura 1.19: Supertask e multitask

Altre novità introdotte in questa ricerca sono rappresentate dai concetti di *supertask* e di *multitask*, mostrati in Fig. 1.19

- **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 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).

1.4 Il workflow e la logica di processo

La maggior parte delle attività aziendali per essere completate richiedono il succedersi di più fasi e la partecipazione di più attori con capacità e funzioni specifiche. Identificate le varie fasi del processo, assegnati gli esatti esecutori, bisogna ricostruire poi il processo stesso come flusso di lavoro coordinato. La logica di processo definisce proprio le regole che un processo aziendale deve seguire e un sistema di workflow fornisce il supporto informatico per il corretto rispetto di queste regole.

In generale però non tutti i processi aziendali sono automatizzabili tramite sistemi di workflow; i processi modellizzabili sono quelli che :

- hanno un inizio ben definito, causato da un evento o da un ciclo temporale ;
- hanno uno o più stadi finali;
- possono essere suddivisi in uno o più sottoprocessi;
- le transazioni tra i processi sono governate da regole ben precise.

Con riferimento a [13] possiamo suddividere i processi di un'azienda in tre categorie :

1. **material process**;
2. **information process**;
3. **business process**.

Lo scopo di un *material process* è quello di assemblare componenti fisici e generare prodotti fisici. Tali attività (o *task*) consistono nel muovere, immagazzinare, trasformare, misurare e assemblare oggetti fisici. Gli *information*

processes riguardano attività automatizzate (gestite da programmi informatici) o parzialmente automatizzate (gestite da programmi con l'interazione delle persone) che creano, processano, gestiscono, e producono informazioni. I database e la tecnologia dei sistemi distribuiti forniscono un'infrastruttura di base per il supporto degli information processes. La nozione di *business process* si colloca ad un più alto livello rispetto alle precedenti definizioni: sono descrizioni orientate al mercato delle attività e delle procedure di un'azienda, implementate attraverso information processes o material processes. Quindi un business process è realizzato per raggiungere un certo scopo aziendale (ad esempio il profitto) o per soddisfare uno specifico bisogno del cliente. Un'azienda che riesce a delineare, definire i propri obiettivi in termini di business processes, è in grado di reingenierizzare ogni procedura per migliorarla o adattarla alle continue richieste di mercato. Ciò produce un incremento della soddisfazione del cliente, dell'efficienza dell'organizzazione, della qualità dei prodotti e una diminuzione dei costi. *Business process reengineering* (BPR), come verrà più approfonditamente sottolineato nella sezione seguente,riguarda l'esplicita revisione e riprogetto dei business processes. Essa viene attuata prima che i sistemi informatici siano utilizzati per automatizzare questi processi. *Information process reengineering* è un'attività complementare del business process reengineering e mira a determinare come utilizzare i vecchi e nuovi sistemi informatici per automatizzare i reengineered business processes. Le due attività possono essere svolte in azienda iterativamente in modo da realizzare un mutuo feedback. Mentre la prima attività può esplicitamente mirare al raggiungimento della soddisfazione del cliente, la seconda può rendere il sistema informativo più efficiente, ridurre i costi e trarre vantaggio dall'evoluzione dell'IT. Il workflow (flusso di lavoro) è un concetto molto correlato al reengineering e all'automazione dei business e information processes : tramite i flussi di lavoro si possono descrivere le fasi di un business process a un livello concettuale tale da riuscire a capirlo, valutarlo e riprogettarlo. Per quanto riguarda gli information processes, i workflows possono "catturare" le fasi che li compongono in modo da descrivere i requisiti necessari per la funzionalità del sistema informativo e le esigenze delle persone che vi lavorano. La distinzione tra questi due punti di vista non è sempre ben definita, e qualche volta il termine workflow è usato per descrivere entrambe le prospettive dei business e information systems.

Il *Workflow Management*, come già descritto, è una tecnologia che supporta la reingenierizzazione dei business e information processes e comporta :

1. definire i flussi di lavoro, descrivendo quegli aspetti di un processo che sono rilevanti per controllare e coordinare l'esecuzione dei task;

2. rendere possibile una veloce (ri)progettazione e (re)implementazione

dei processi per far fronte a un cambiamento del sistema informativo o delle esigenze dell'azienda.

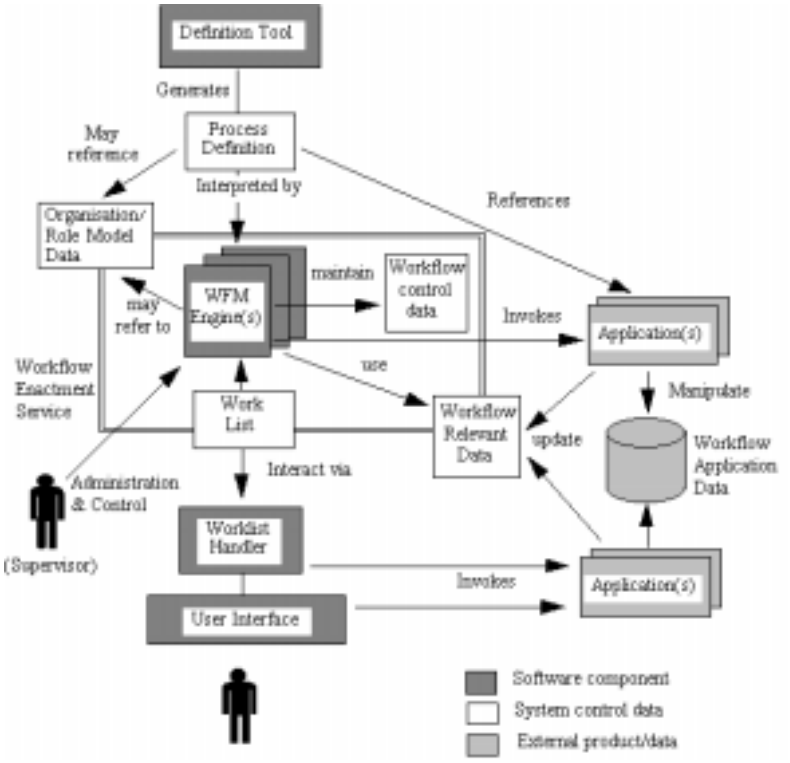


Figura 1.20: Struttura di un generico prodotto di Workflow (fonte WfMC)

1.4.1 Workflow e Business Process Reengineering

La tecnologia di supporto al workflow e il *Business Process Reengineering* (BPR) si sono sviluppate indipendentemente anche se tuttavia qualche volta vengono usati come sinonimi. In realtà non vi è una netta separazione tra i due concetti. Business process reengineering ha come obiettivo complessivo analizzare e progettare sistemi aziendali e loro parti; quindi sono analizzati i processi aziendali e sono analizzate e valutate possibili alternative e modifiche. L'attenzione è focalizzata sulla pianificazione e progettazione dei business systems (*build time*), mentre l'esecuzione dei processi (*run-time*) viene considerata solo per analizzare gli effetti di un possibile feedback della reingenierizzazione; BPR usa le metodologie della modellazione dei business process. Lo scopo della modellazione del workflow è invece centrato sull'esecuzione dei processi nel senso che l'analisi e modellazione viene direttamente

effettuata per una visione a run-time. Questa differenza porta a fondamentali conseguenze :

Caratteristiche di modellazione La modellazione del BPR risulta tipicamente più ampia (*Modeling in the width*), mirata a rispondere a quesiti del tipo "iniziare con che cosa" (ad alto livello), "perché", "e se anche ... ". La modellazione del workflow risulta più approfondita (*Modeling in the depth*), finalizzata a risolvere questioni del tipo "iniziare con che cosa" (ad un livello dettagliato), "come" e "chi".

Specifiche dei flussi di lavoro Sia la modellazione BPR che del Workflow sono centrate sulle attività (task) aziendali e sui flussi di lavoro; in più la Business Process Reengineering identifica questi task e workflows esaminando i servizi aziendali, la loro cooperazione e coordinazione.

Scopo della ricerca Per automatizzare i processi a run-time, la modellazione del Workflow analizza in dettaglio le attività aziendali come pure gli "attori" che le svolgono. Per la modellazoine BPR è invece accettabile raggiungere un'analisi efficiente sui servizi e sulle procedure aziendali, indipendentemente dai vari aspetti concernenti gli esecutori.

Caratteristiche dei modelli Nella BPR può essere utilizzata una modellazione che prevede procedure gerarchiche, in modo da produrre un processo comprendente molti livelli di astrazione. Partendo dai processi principali del sistema aziendale, fino a livelli sempre più dettagliati. La modellazione del Workflow invece produce solo un modello "piatto", esistendo solitamente un solo livello d'astrazione.

1.5 Lo Standard

Attualmente i due più importanti tentativi di standardizzazione nell'ambito della tecnologia del workflow sono quelli proposti dalla Workflow Management Coalition (WfMC) e dalla collaborazione tra Microsoft e Wang.

1.5.1 WfMC standard

La WfMC è un'associazione internazionale, fondata nel 1993, ed è ora considerata una fonte primaria per lo standard nel mercato del workflow. Essa si propone di fornire un modello di riferimento per i Workflow Management Systems, identificando le loro caratteristiche, terminologie e componenti.

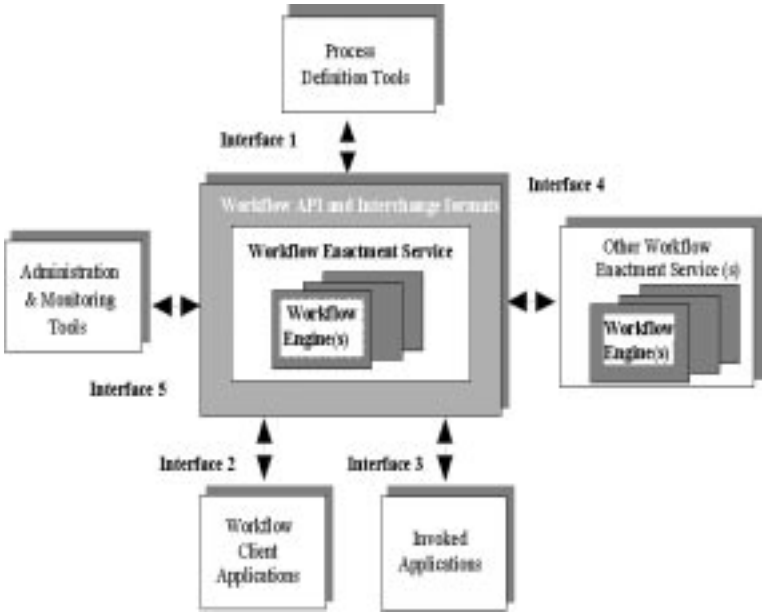


Figura 1.21: Workflow Reference Model - Componenti e interfacce

Workflow Reference Model

Il modello di riferimento specifica una struttura per i sistemi di workflow, identificando le loro caratteristiche, funzioni e interfacce. La figura 1.21 illustra i più importanti componenti di un'architettura di workflow.

L'attenzione è centrata sulla specifica di cinque interfacce di workflow (WAPIs) che circondano il workflow engine. Queste APIs provvedono a una comunicazione standard tra il workflow engine e i client (incudendo gli altri componenti di workflow, come la definizione dei processi e i tool di monitoraggio).

Workflow Enactment Services. Il *Workflow Enactment Service* fornisce l'ambiente a run-time in cui le istanze dei processi vengono eseguiti. È composto da uno o più *workflow engine*; le applicazioni si interfacciano a questo servizio tramite la *Workflow Application Programming Interface* (WAPI).

L'interazione con le risorse esterne avviene tramite una o due interfacce:

- La *Client application interface*, attraverso la quale un workflow engine interagisce con il gestore della *worklist* (lista di task associati a una o più persone). L'attivazione di strumenti applicativi deve essere eseguita sotto il controllo del gestore della worklist o dell'utente finale.

- La *Invoked application interface*, che abilita il workflow engine ad attivare direttamente uno specifico tool per eseguire una certa attività.

Il workflow engine. Un *Workflow engine* è un componente software responsabile di parte o tutto il controllo dell'ambiente run-time.

Tipicamente tale software fornisce falicitazione nel trattare:

- l'interpretazione della definizione dei processi;
- il controllo delle istanze dei processi - creazione, attivazione, sospensione, terminazione, ecc.;
- la navigazione attraverso le attività dei processi, che possono coinvolgere operazioni sequenziali o parallele, scheduling delle scadenze, interpretazione dei dati rilevanti, ecc.;
- l'ingresso o uscita di partecipanti al workflow;
- il mantenimento dei dati di controllo e rilevanti;
- un'interfaccia per invocare applicazioni esterne e collegare tutti i dati rilevanti;
- la supervisione delle azioni di controllo e amministrazione

Workflow Definition Interchange (*Interfaccia 1*). L'interfaccia tra strumenti di modellazione e definizione di processi e il software per la gestione a run-time del workflow è definita come "l'interfaccia import/export della definizione dei processi". Quest'interfaccia supporta lo scambio di informazioni, fisiche o elettroniche, tra il WFMS e i prodotti per la definizione dei processi (*Process Definition Tools*).

Il lavoro della Coalition si concentra anche su due aspetti:

1. la derivazione di un meta-modello che può essere usato per descrivere oggetti, le loro relazioni e attributi, all'interno della definizione di un processo;
2. chiamate ad API fra sistemi di workflow o tra sistemi di workflow e prodotti di definizione dei processi, fornendo un modo comune per accedere alle definizione dei processi. L'accesso può essere in lettura, in lettura/scrittura o solo in scrittura e può manipolare un set di oggeti standard all'interno del meta-modello o di un set di prodotti specifico.

Workflow Client Application Interface (*Interfaccia 2*). La Workflow Client Application Interface è l'interfaccia per accedere da applicazioni "client" esterne al workflow engine e alla worklist.

Le APIs e i suoi parametri vengono mappati in diversi meccanismi di comunicazione (ad es. via e- mail); le caratteristiche essenziali di queste API sono:

- *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.

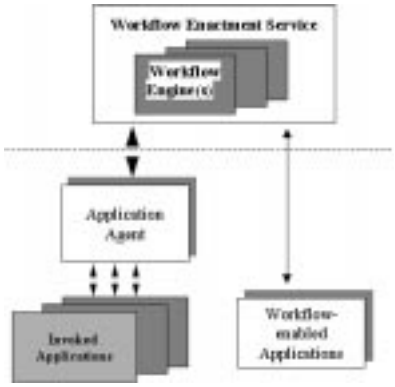


Figura 1.22: Invoked Application Interface.

Invoked Applications Interface (*Interfaccia 3*). La figura 1.22 mostra lo scopo di quest'interfaccia, progettata per essere applicabile agli "agenti" delle applicazioni e alle applicazioni "workflow enable" (adatte al workflow).

In un caso semplice, l'invocazione dell'applicazione è gestita localmente al workflow engine, usando informazioni della definizione del processo per identificare la natura delle attività, il tipo d'applicazione da invocare e tutti i requisiti sui dati. L'applicazione invocata deve essere locale al workflow engine, co-residente sulla stessa piattaforma o in un locazione separata accessibile via rete. Le funzionalità delle APIs che si interfacciano a tali applicazioni sono le seguenti :

- *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.

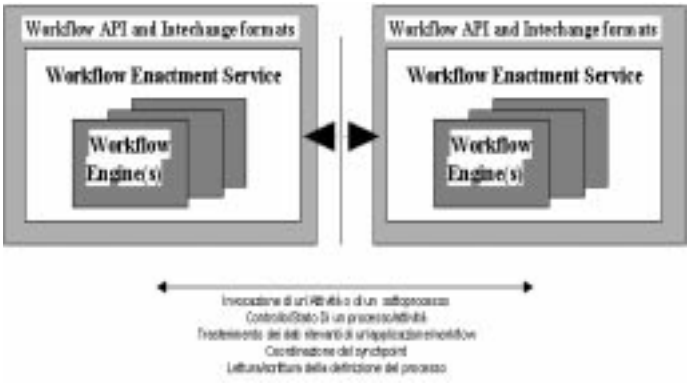


Figura 1.23: Workflow interoperability interface.

- *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*). La generica natura delle informazioni e flussi di controllo tra eterogenei sistemi di workflow è mostrata in Fig.1.23.

Vi sono due aspetti fondamentali per l'interoperabilità :

- fino a che punto è necessario estendere la comune interpretazione della definizione dei processi (o sotto-processi);
- il supporto a run-time all'interscambio dei vari tipi di controllo delle informazioni e trasferire i dati rilevanti di workflow e/o delle applicazioni tra differenti "enactment services".

I modelli d'interoperabilità finora proposti sono i seguenti quattro:

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.

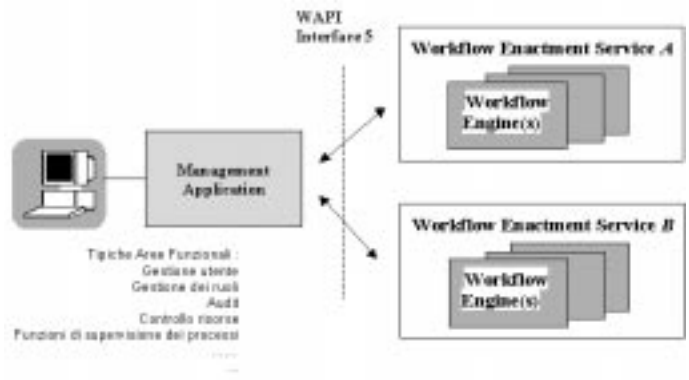


Figura 1.24: Systems Administration & Monitoring Interface.

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.

Administration and Monitoring Interface (*Interfaccia 5*). L'interfaccia 5, come illustarto in Fig. 1.24, permette a un indipendente applicazione

di gestione di interagire con differenti domini di workflow; lo scopo è permettere una completa visione dello stato del lavoro dell'organizzazione e offrire un completo set di funzioni a scopo di amministrazione, incudendo specifiche considerazioni di sicurezza, controllo e autorizzazioni.

I dettagli di quest'interfaccia sono ancora oggetto di studio, ma si può prevedere che includerà i seguenti tipi di operazioni (alcune delle quali sono comuni alle aree delle altre interfacce) :

- **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).

1.5.2 MAPI Workflow Framework

MAPI è uno standard basato sui messaggi API promosso dalla collaborazione tra Microsoft e Wang e MAPI Workflow Framework (MAPI-WF) è un'iniziativa Microsoft per la Coalition. L'idea è combinare le funzionalità dei sistemi per il workflow e la flessibilità dei sistemi basati sui messaggi; esso mira all'interoperabilità tra i sistemi basati sui messaggi (*messaging systems*) e sistemi per il workflow. In un ambiente basato sui messaggi, una richiesta di workflow (ad esempio, dell'interfaccia 4) può essere "incapsulata" all'interno di parti di un messaggio. MAPI-WF fornisce un set di parti di messaggio e proprietà in modo che "pacchetti" di workflow possano essere spedite da e per

il workflow engine. I componenti di workflow (il workflow engine, le applicazioni e gli strumenti di workflow) che sono conformi a WAPI-WF possono comunicare attraverso un sistema di messaggistica come Microsoft Exchange.

Capitolo 2

I sistemi informatici per la gestione del Workflow

2.1 Background

La tecnologia informatica si è evoluta a tal punto da essere utilizzata con successo in molti domini applicativi quali le banche, la finanza e le telecomunicazioni, migliorandone la produttività e fornendo maggiori servizi. Nonostante questo succeso però, le più recenti versioni delle applicazioni informatiche presentano due grandi svantaggi.

Primo, esse sono per natura monolitiche. Tutte le attività di sorveglianza e di accesso alle informazioni sono notevolmente codificate e "nascoste" nelle applicazioni. Questi sistemi sono quindi difficili e costosi da mantenere e migliorare. L'avanzamento della tecnologia dei database ha separato con successo l'accesso ai dati di queste aplicazioni; d'altra parte per modificare le attività di sorveglianza bisogna ancora cambiare il codice sorgente.

Secondo, esse sono isolate. Specialmente quelle meno recenti sono solitamente progettate e sviluppate per lavorare indipendentemente e risolvere problemi molto specifici. Il miglioramento della tecnologia delle reti e del *distributed computing* hanno reso possibile la primitiva collaborazione tra queste applicazioni come attraverso lo scambio di messaggi. È comunque ancora necessario integrare informazioni e processi isolati ad un più alto livello in modo da poter fornire soluzioni aziendali che le singole applicazioni sono impossibilitate a dare.

L'idea di base della tecnologia del workflow è quella di separare i processi aziendali e i componenti del workflow management dalle applicazioni esistenti per aumentarne la flessibilità e la manutenibilità. I maggiori impulsi allo sviluppo di questa tecnologia sono la necessità della reingenierizzazione dei

processi aziendali (il cui scopo, come descritto nel Cap. 1, è l'incremento della produttività, la riduzione dei costi e il far fronte velocemente a cambiamenti dell'ambiente) e lo sviluppo e l'affermarsi di tecnologie quali il distributed computing, la tecnologia ad oggetti, la tecnologia dei database, ecc., che facilitano uno scambio di informazioni aperto e sicuro, e una collaborazione in tutta l'azienda.

Oggi è in atto una vera e propria esplosione dei sistemi di Workflow, non solo per la diffusione, quanto per la forte caratterizzazione che ciascuno presenta riguardo sia ad un fine specifico (di *document management*, *customer-oriented*, di *enterprise management*, ecc.) sia agli strumenti utilizzati per interfacciare gli utenti umani (client, light client, browser) e le applicazioni informatiche (API, RPC).

Il limite alla loro diffusione sta nella scarsa capacità di visione per processi delle realtà aziendali e nella conseguente difficoltà nell'individuare il prodotto più adatto. In definitiva, si può dire che spesso il cliente non dispone dei presupposti minimi per condurre una scelta e l'eccessiva disponibilità di alcune aziende ad abbracciare immediatamente le nuove tecnologie (ad esempio Internet/intranet) e proporle nelle applicazioni di workflow management, induce spesso un senso di precarietà accettabile in aree ad elevato turn-over tecnologico, ma abbastanza "pericoloso" in settori ad elevata criticità come la gestione dei processi interni o di core-business.

2.2 I sistemi per il workflow

Sono passati circa dieci anni dall'introduzione del primo sistema di workflow. Nonostante il gran numero di ricercatori e produttori, nonostante le ottimistiche previsioni sul mercato e nonostante la forte inpennata della tecnologia, i sistemi di workflow non si sono ancora ampiamente diffusi.

Le cause di questa situazione sono numerose. Le seguenti sono le principali da un punto di vista tecnico :

Infrastrutture. I sistemi di workflow sono molto di più che "motori" (*engine*) che eseguono i processi di workflow. La corretta esecuzione di un tale processo richiede un adeguato supporto dell'infrastruttura sottostante. Per esempio, le tecnologie come l'elaborazione ditribuita (*distributed computing*), l'elaborazione ad oggetti e la sicurezza sono necessarie al "workflow engine" per invocare applicazioni esterne. Sfortunamente queste tecnologie, come CORBA o ActiveX/DCOM sono "mature" per applicazioni reali solo in questi ultimi tempi.

Standar. La mancanza di uno standard è stato uno dei maggiori ostacoli ad un'ampia diffusione dei sistemi di workflow. Nonostante i database relazionali, ogni produttore di questi sistemi ha un proprio modello di workflow, linguaggio di specifica e API (Application Program Interface). Gli sforzi della Workflow Management Coalition (WfMC) hanno dato un significativo progresso, ma vi è ancora molta strada da percorrere.

Complessità. Lo sviluppo di applicazioni orientate al workflow è un'attività complessa che va oltre la semplice definizione dei processi : bisogna far interagire le applicazioni esterne al workflow engine, è necessario gestire al meglio tutte le risorse del workflow e settare tutte le infrastrutture di comunicazione. Sfortunatamente gli odierni sistemi di workflow aiutano molto poco, quindi tutte le maggiori applicazioni di workflow richiedono una lunga, intensa e costosa collaborazione tra i venditori dei sistemi di workflow e gli sviluppatori delle applicazioni.

Tecnologia. Sebbene i progressi dell'IT siano notevoli, la tenologia del workflow è ancora lontana dall'essere consolidata. È noto che nessuno dei prodotti per il workflow esistenti o prototipi di ricerca può fornire lo stesso livello di supporto di quello dato dai DBMS a una sicura e consistente esecuzione dei processi. È anche vero però che alcune applicazioni di workflow non necessitano di tale supporto ma è importante per questi sistemi avere tale capacità in modo che applicazioni vitali che sfruttano altre tecnologie (ad es. database) correntemente utilizzate in azienda possano essere reingenierizzate tramite il workflow.

2.3 Le caratteristiche dei prodotti attuali

Analizzati dalla prospettiva dei sistemi distribuiti e dei database, lo stato corrente della tecnologia dei WFMS può essere caratterizzata considerando il grado di dipendenza dei processi dall'interazione umana e del software nell'eseguire e coordinare le attività. Tale caratterizzazione è rappresentata in Fig. 2.1.

I prodotti attuali per il Workflow Management sono essenzialmente basati sulle immagini. Lo scopo di tali sistemi è automatizzare e gestire immagini, dati, testo, e altre informazioni di un'azienda. Questi sistemi sono anche centrati sui documenti o i dati. La principale funzione di un processo di workflow "*data-centric*" è l'instradamento dei dati (ad es. un documento di progetto) in modo che le persone possano lavorare localmente sulle informazioni.

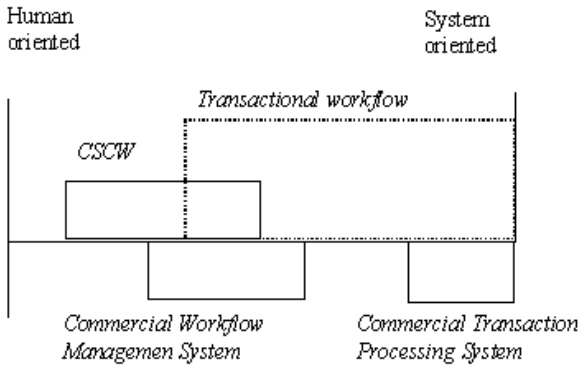


Figura 2.1: Le tecnologie rispetto alle attività umane e automatizzate

In tempi recenti, molti produttori di sistemi per il workflow hanno sia sviluppato sia rinominato i loro sistemi come non basati sulle immagini. I sistemi basati sui processi, o "*process- centric*", (al contrario dei data-centric) formalizzano e applicano le attività di sicurezza.

Di seguito sono elencate le principali caratteristiche dell'attuale generazione dei prodotti per il workflow.

Rappresentazione grafica. Uno dei più significativi miglioramenti degli odierni sistemi di workflow è la capacità di specificare e rappresentare i processi di workflow tramite mappe grafiche. Nella mappa, i flussi di controllo e dei dati, così come gli altri componenti di workflow, sono rappresentati graficamente usando icone e linee che le collegano. L'idea è fornire una vista del processo comprensibile anche a i non programmatori come gli analisti aziendali, i consulenti per la reingenierizzazione e gli utenti finali.

La mappa dei processi del workflow è oggi un componente standard di tutti i prodotti di workflow. Essa però varia da produttore a produttore sia per le informazioni che rappresenta, sia nel modo in cui è rappresentata. Per esempio, qualche prodotto supporta mappe a un solo livello mentre altri mappe gerarchiche (ad es. una mappa principale con icone che rappresentano sotto-mappe). Alcuni prodotti gestiscono i dati di workflow esplicitamente nella mappa, altri no. Anche la granularità è diversa: per es. alcuni prodotti non includono le specifiche delle attività di workflow e altri non descrivono con sufficiente chiarezza e completezza i processi.

Architettura. I sistemi di workflow sono per natura sistemi distribuiti. Tuttavia la maggior parte dei prodotti utilizza architetture client-server e

il software è installato su più macchine; il workflow engine agisce da coordinatore. Gli altri componenti del sistema di workflow, come il monitor, l'inizializzatore e il controllore dei processi sono tutti "client" del workflow engine. Le applicazioni esterne che eseguono le attività del workflow possono essere sia geograficamente distribuiti che su differenti piattaforme.

La maggior parte dei workflow engine è ancora centralizzata, nel senso che l'intera esecuzione di un processo è trattata da un singolo "engine" (o una serie di "engine" che condividono gli stessi dati). È anche possibile che in alcuni sistemi di workflow partano sottoprocessi in una macchina diversa come fasi dell'esecuzione del processo corrente; però l'esecuzione del processo e del sottoprocesso possono essere considerate separate con una minima interazione se non tramite scambio di dati all'inizio e alla fine dell'esecuzione del sottoprocesso.

Modello dei dati. Nel modello di riferimento della WfMC, sono stati identificati tre tipi di dati di workflow: *process control data* (dati per il controllo del processo), gestiti solo dal sistema per il wokflow management; *process relevant data* (dati essenziali del processo), utilizzati sia dalle applicazioni che dal workflow management system; e *application data* (dati delle applicazioni), utilizzati solo dalle applicazioni di workflow. L'idea è separare le attività di controllo (ad es. il controllo del flusso delle attività e i dati da esso utilizzati) dai dettagli dell'applicazione (ad es. i dati utilizzati per eseguire i task).

Tutti i prodotti per il workflow hanno il proprio modello dei dati ma alcuni di essi sono in parte differenti dal modello della WfMC. Per esempio molti sistemi di workflow non fanno differenza tra "process relevant data" e "application data"; in questi tipi di sistema i workflow engine hanno Accesso a tutti i dati di workflow (inclusi i dati delle applicazioni).

Il modello utente. Nel modello utente sono specificati i ruoli che l'utente deve avere e quelli che deve coordinare. Lo scopo è separare il concetto di ruolo logico, che rappresenta la specifica delle capacità necessarie ad eseguire le attività, dal concetto di risorsa fisica, che deve avere certe caratteristiche per eseguire le atività. I progettisti del processo specificano i ruoli per i task di workflow e le risorse con le caratteristiche necessarie da assegnare al processo nella fase di esecuzione. Il vantaggio maggiore è l'indipendenza del processo di workflow dalle risorse specifiche che possono quindi cambiare nel suo ciclo di vita.

Esistono in commercio prodotti per il workflow che non distinguono tra ruoli logici e risorse fisiche. Comunque molti sistemi supportano il "modello

utente" e alcuni presentano modelli più complicati che permettono la specifica dell'organizzazione degli utente e del manager, le funzioni e i processi che un utente è autorizzato ad eseguire, ecc... .

Funzionalità dei ruoli. Quasi tutti i sistemi per il workflow permettono l'esecuzione dei processi in maniera più dettagliata di una semplice sequenza di attività. Per gestire un flusso di controllo complesso i sistemi di workflow devono fornire molte funzionalità ai ruoli. La maggioranza dei workflow systems possiede specifici moduli software (*"rule engine"*) che processano le richieste dei ruoli, mentre per la definizione delle funzionalità alcuni sistemi forniscono linguaggi specifici, altri un più semplice editor grafico.

Strumenti per test, analisi, animazione e simulazione. Gli strumenti per il test simulano un processo di workflow, tramite l'input di dati semplici e eventi esterni, come il completamento di attività, scadenze di termini ed eccezioni. L'analisi è necessaria per rilevare errori logici ed ottenere una stima dei tempi di completamento, e suggerire anche modifiche alla specifica dei processi per migliorarne l'efficienza. L'animazione serve a mostrare un processo di workflow sullo schermo di un computer e valutarlo durante il progetto. La simulazione fornisce la percezione del comportamento dinamico, a regime o all'inizio, la localizzazione di eventuali colli di bottiglia, ecc.

Strumenti per la monitorizzazione. Questi strumenti possono fornire al progettistista diversi punti di vista dell'esecuzione dei processi. Essi illustrano quale o quali attività sono correntemente in esecuzione, da chi sono eseguite, le priorità, le scadenze, le durate e le dipendenze. Gli amministratori del sistema di workflow possono usare tali strumenti per vedere statistiche sui tempi di completamento delle attività, carichi di lavoro e capacità degli uenti, oppure per generare reports e periodici sommari sull'esecuzione dei processi di workflow. Altre applicazioni includono la possibilità di "estrarre" i modelli dei processi dai dati stessi, e validare tali modelli in base all'attuale esecuzione.

2.3.1 Principali carenze dei sistemi commerciali

Limiti sul concetto di task. Un primo difetto della maggioranza dei sistemi di workflow presenti sul mercato è basato sulla limitata visione del concetto di *task* :

- nessun prodotto supporta i *task multi- utente*, escludendo la possibilità che più persone partecipino ad un'attività. L'avvento di tecnologie

di comunicazione sempre più evolute (come la video-conferenza) e il distributed computing possono facilitare la condivisione di più risorse ed attività.

- molti sistemi non permettono che un task venga interrotto durante la sua esecuzione; questo è un grosso limite in quanto

  - le persone che eseguono le attività di workflow necessitano di pause e riposo;
  - i task possono essere interrotti da altri task con priorità diversa;
  - alcuni task potrebbero necessitare di un input esterno non immediatamente disponibile.

- Quasi nessun prodotto supporta i *task multi-risorsa* poiché in fase di progettazione si collegano i task a singole applicazioni e anche sistemi più flessibili non permettono funzionalità adeguate all'utente per cambiare l'applicazione assegnata al task.

- Come già visto nel Cap. 1 la modellazione del workflow, a differenza dei modelli di processi utilizzati del Business Process Reengineering, è pensata per progetti "piatti", pressoché senza una struttura gerarchica, senza prevedere la presenza di sotto-processi.

- Essendo la maggioranza dei sistemi odierni basata su architettura client-server, sono scarsamente utilizzati i *task distribuiti*, nati per architettura distribuita. Lo scenario che si prospetta è la possibilità di installare più workflow engine su macchine diverse (in un architettura distribuita), in modo da scambiarsi informazioni e "distribuire" l'esecuzione dei processi fra i diversi siti. Questo è l'argomento principale della tesi e verrà approfondito nei capitoli successivi.

Limiti nella definizione dei ruoli. Oltre alle limitazioni sul concetto di task abbiamo anche restrizioni sulla definizione dei *ruoli* : nel progetto sono specificate le attività da svolgere, i ruoli e le persone assegnate a questi task; la persona è assegnata al task a run-time e quasi mai è permesso al progettista specificare un ordine di preferenza tra le persone. Questo limite appare evidente tutte le volte che un ruolo ricorra più volte nello stesso processo o quando sono coinvolti task da eseguirsi ciclicamente : sarebbe logico poter assegnare sempre la stessa persona al task in tutti i passi del ciclo, date le informazioni e i dati che questa ha acquisito nella prima esecuzione.

Limiti nel process management. I principali limiti nella gestione dei processi sono 1. le *condizioni di sincronizzazione* e 2. l'*ottimizzazione dinamica*.

1. Solitamente per gestire le condizioni di sincronizzazione si utilizzano "allarmi" a scadenze temporali ben definite ; raramente i sistemi di workflow supportano una "supervisione" dello stato del singolo task (ad esempio per modificarne la priorità di esecuzione). Una conseguenza importante è sicuramente la mancanza di una gestione flessibile della condizione di *and*, in modo da poter continuare l'esecuzione del processo anche se non tutti i task paralleli sono terminati.

2. Oltre alle condizioni di sincronizzazione basati su allarmi temporali, i sistemi di workflow attuali forniscono anche meccanismi di avvertimento (*warnings*) per segnalare eventuali malfunzionamenti ; quasi mai sono presenti algoritmi di autocorrezione o meccanismi che permettano di modificare l'accesso dei task a una risorsa (in modo che più persone possano risolvere i "colli di bottiglia"); raramente è possibile cambiare l'allocazione dei task alle risorse (per allocare un task da un utente ad un altro, come ad esempio per assegnare un task la cui esecuzione è "critica" per il processo a utenti più esperti) o anche la stessa definizione di processo.

Limiti nel data management. Come descritto nel Par. 2.3 il limite più comune dei WFMSs è la mancanza di una distinzione tra i "relevant data" (dati usati sia dalle applicazioni client che dai workflow management system) e gli "application data" (dati utilizzati nelle applicazioni che si interfacciano al workflow engine) che vengono quindi memorizzati allo stesso modo. Questa lacuna si manifesta soprattutto in applicazioni (e processi) che devono gestire grandi quantità di dati (*data-intensive*) dove appare evidente la mancanza di un DBMS.

2.4 Tendenze del mercato

Il Workflow Management è un'area tecnologica giovane e in continua evoluzione. Nella tabella 2.1 e nelle Figure 2.2 e 2.3 si riportano i dati attuali e previsioni future sul mercato e spesa mondiale in prodotti e servizi di workflow.

Non è ancora chiaro quali caratteristiche avrà la prossima generazione di prodotti ; nelle seguenti sezioni verranno elencate alcune delle più importanti e generali tendenze del mercato.

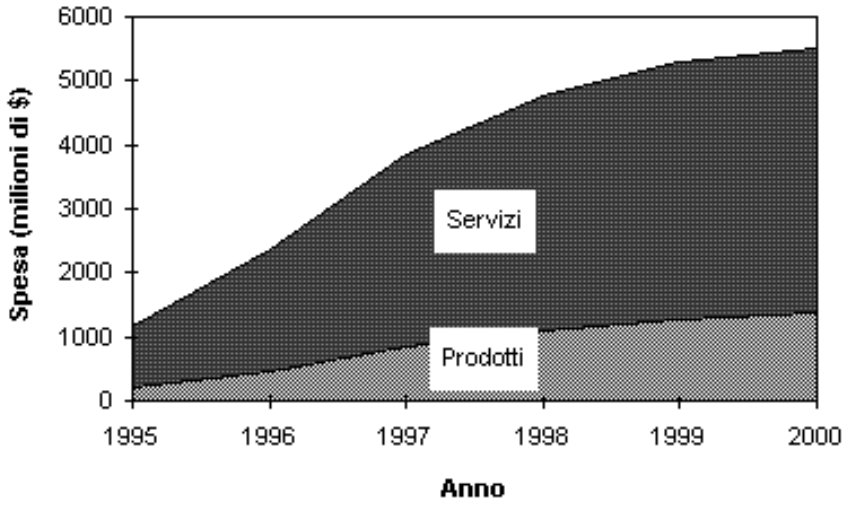


Figura 2.2: Spesa mondiale (milioni di $) in prodotti e servizi di workflow.

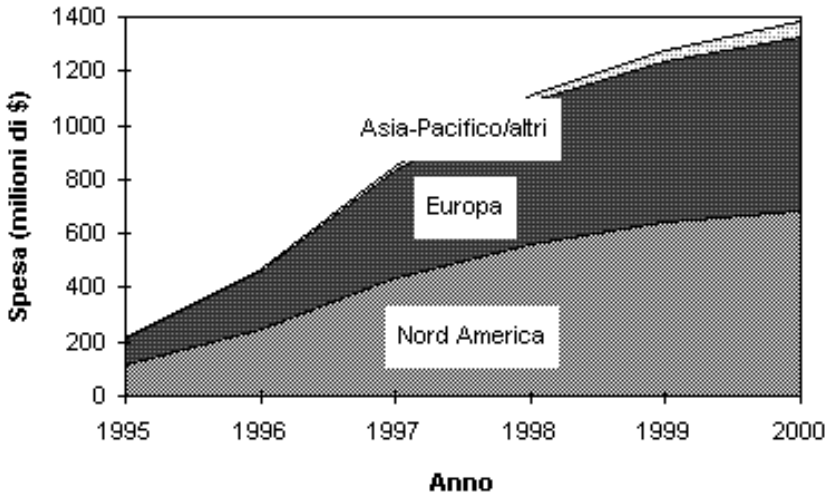


Figura 2.3: Spesa mondiale (milioni di $) in prodotti di workflow per area geografica.

Anno	*Servizi di workflow e tecnologie affini*	*Generici tools*	*Transactional Workflow*
1992	226(DL),186(DL)	115(DL)	
1993	628(ID)	250	44
1994	1200(ID)	540	106
1995	1880(ID),2000(CW)	810	184
1996	2500(IDC,DL)	1120(DL)	293
2000	7000(Ovum)	2000(Ovum)	

Fonti: ID = IDC & Advante; DL = Delphi Consulting; IT = International Data Corp.

Tabella 2.1: Tendenze attuali e future del mercato del Workflow

2.4.1 Interfacce aperte ed estendibili

Un'azienda che voglia sfruttare la tenologia del workflow deve già possedere (o installare) una rete di computer, applicazioni (ad es. fogli elettronici), dati (memorizzati in files, database, ecc...) e altre informazioni. Per essere utile un Workflow Management System deve quindi inserirsi funzionalmente nel sistema informativo aziendale.

La maggior parte dei prodotti di workflow include un'interfaccia per la programmazione di applicazioni. Le applicazioni esterne possono essere integrate con il sistema di workflow e i dati esterni possono essere utilizzati per l'esecuzione dei processi di workflow attraverso, ad esempio, i "data handlers" (gestori di dati) delle applicazioni. Per essere un sistema di workflow realmente aperto, sono necessarie interfacce aperte ed estendibili in modo da incorporare le altre risorse esistenti e le informazioni necessarie all'esecuzione del processo di workflow. Per esempio, gli eventi (o allarmi) sono uno dei maggiori metodi per l'interazione tra i processi e l'ambiente esterno. Un processo per la gestione di una rete di telecomunicazione deve essere in grado di reagire ad allarmi generati dalla rete stessa e generare eventi che la modifichino. Ci sono ricerche che sono indirizzate allo studio di sistemi che supportino proprio l'interfaccia dei sistemi di workflow con l'ambiente circostante.

Un altro esempio che richiederebbe un'interfaccia estendibile è l'integrazione dei direttori comuni presenti nel sistema informativo aziendale : le informazioni riguardo gli utenti e la gerarchia aziendale sono necessarie all'assegnazione delle risorse per l'esecuzione dei task del workflow ; un modo non molto funzionale, utilizzato da alcuni sistemi, consiste nella richiesta agli utenti di autoregistrarsi nel sistema di workflow. Un modo più flessibile pre-

vede la presenza di un'interfaccia per intergrare al workflow engine i direttori comuni già presenti ; in questo modo non solo si risparmia tempo nello sviluppo dell'applicazione di workflow, ma rende la futura manutenzione più semplice ed evita possibili inconsistenze di dati.

2.4.2 Gli ambienti di sviluppo dei processi

Lo sviluppo di applicazioni workflow è generalmente difficoltoso causa la loro complessità. Gli attuali prodotti per il workflow tentano di risolvere il problema fornendo interfacce utente grafiche per lo sviluppo e progetto dei processi. Questi strumenti (*GUI tools*) comunque sono indicati solo per un aspetto del problema : la specifica della mappa dei processi. Un probelma più arduo da risolvere rimane quello dell'integrazione del processo di workflow con il complesso e eterogeneo ambiente informatico aziendale. Molti produttori di WFMS progettano e vendono i loro sistemi come strumenti generici in grado di coprire tutte le aree applicative.

Sicuramente se la complessità dello sviluppo di applicazioni di workflow diminuirà, tramite l'uso di un buon ambiente di sviluppo, questa tecnologia sarà destinata ad espandersi ulteriormente. Attualmente sono presenti sul mercato prodotti di workflow specifici (*special-purpose*) che forniscono mappe dei processi, maschere per i dati e strumenti per la gestione delle applicazioni specifici di quel dominio applicativo per il quale sono nati.

Per i prodotti di workflow *general-purpose* sono presenti speciali pacchetti basti su general-purpose workflow engine. Ad esempio, FileNet ha introdotto VisualFlo/Payable per i conti da pagare ; HP ha prodotto AdminFlow per l'amministarzione aziendale. Si prevede che saranno sviluppati pacchetti per coprire domini applicativi quali le telecomunicazioni, le banche e la finanza.

Come appena menzionato, un importante aspetto di un ambiente per lo sviluppo dei processi di workflow è interfacciarsi alle applicazioni esterne; queste possono essere incluse in una libreria e quindi riutilizzate dai processi; in questo modo lo sviluppo dei processi può essere ulteriormente semplificato se le attività di workflow (la specifica logica dei ruoli che mappano l'applicazione esterna, i dati necessari all'esecuzione dei task, i meccanismi di comunicazione, ecc.) possono essere riusate. Sfortunatamente la maggioranza dei prodotti commerciali non forniscono questo livello di riuso, e le attività di workflow contengono le informazioni del processo specifico (ad esempio la posizione nel processo) in modo da non poter essere riusate. È quindi indispensabile separare le parti indipendenti e specifiche delle attività di workflow: per esempio, i prodotti dell'HP, distinguono tra le due parti e permettono il riuso degli elementi indipendenti delle attività di workflow.

Questo permette a sistemi di workflow specisl-purpose, come AdminFlow, di essere facilmente sviluppati basandosi su general-purpose workflow engine.

2.4.3 Wide Area Workflow

L'attuale generazione dei prodotti per il workflow è stata criticata per la "rigidità" dei modelli dei processi, per la limitata considerazione delle applicazioni esterne e per le restrizione sulle piattaforme hadware. L'avvento delle reti mondiali (wide area networks) e l'esplosione del world wide web hanno aperto nuove opportuinità per il workflow: molti sistemi forniscono interfacce al web (come la successiva versione del prodotto da noi utilizzato Action Workflow System) e ci sono anche progetti di ricera che sviluppano il workflow *on the web*.

Nelle previsioni dei ricercatori , un possibile cambiamento riguarda l'ambiente-utente dove i task sono eseguiti: si potrà accedere ai sistemi attraverso interfacce aperte come e-mail, telefono, fax, Web e intranet/extranet. Il vantaggio dei *wide area workflow* sarà un più veloce tempo di risposta alle esigenze degli utenti e una più grande produttività, fornendo a molteplici punti d'accesso la disponibilità delle stesse informazioni.

2.5 La ricerca

Il workflow è un'area di ricerca molto attiva che coinvolge sia le università che l'industria. D'altra parte ciò ha un piccolo impatto sui prodotti commerciali perchè nei più recenti sistemi di workflow, che si sono sviluppati da diverse aree quali l'automazione delle procedure d'ufficio, sistemi di controllo del lavoro e sistemi per la gestione di documenti, si cercava di definire i modelli base, le architetture e le funzionalità; i ricercatori hanno invece focalizzato la loro attenzione nell'intodurre l'avanzata tecnologia dei database ai sistemi di workflow in maniera troppo poco flessibile e funzionale perchè i produttori potessero applicarla ai loro sistemi.

Nelle seguenti sezioni verranno elencati alcuni progetti di ricerca sul workflow e si evidenzieranno le differenze tra la tecnologia dei database e quella del workflow.

2.5.1 Transactional Workflow

Una transazione, nelle applicazioni di database, è un'unita di esecuzione con proprietà *ACID* : o tutte o nessuna delle sue azioni hanno effetto (*Atomicity*); essa porta il database da uno stato consistente ad un altro (*Consi-*

stency); i suoi effetti sono permanenti una volta che essa raggiunge lo stato di "committed" (*Durability*); le transazioni multiple possono essere eseguite parallelamente, ma gli effetti globali devono essere equivalenti a quelli di un'esecuzione sequenziale (*Isolation* o *Serializability*).

I modelli di workflow che supportano transazioni con tali proprietà sono stati studiati da molti ricercatori ed è stato mostrato come sia possibile e molto utile incorporare la semantica delle transazioni, come il "recovery", l'atomicità e l'"isolation", per assicurare una corretta e funzionale esecuzione del workflow.

D'altra parte un processo di workflow è fondamentalmente diverso da una transazione in un database. Innanzitutto un ambiente di workflow e più complicato di quello di un database e coinvolge componenti eterogenei. Inoltre, un processo di workflow è strutturalmente più complesso e la sua esecuzione comporta spesso un controllo e flusso dei dati molto articolato tra tutte le varie attività di workflow. La specifica di un processo di workflow può includere salti condizionati, esecuzione parallela di attività, cicli, e altre complesse strutture di controllo.

Le tecniche di ricovery, come il *logging*, sono state adottate con successo dai sistemi di workflow ed esistono sistemi che supportano un'affidabile esecuzione dei processi. Mancano ancora però, e sono in fase di ricerca, sistemi che supportano un'esecuzione dei processi atomica e consistente. Nelle seguenti sezioni si tratteranno il controllo della concorrenza e la "compensazione" nei sistemi di workflow come risultato delle differenze appena viste tra la tecnologia del workflow e dei database.

Compensation

Il meccanismo di "compensazione" (*compensation*) è stato introdotto per simulare le proprietà transazionali nelle applicazioni data-intensive, le quali sarebbero diventate troppo "pesanti" se implementate come una singola transazione ACID. L'idea era implementare tali applicazioni come una sequenza (o *saga*) di transazioni ACID tali che le risorse necessarie ad esse solo in un determinato passo, potessero essere rilasciate dopo che la corrispondente transazione fosse terminata. L'atomicità era simulata *compensando* transazioni già terminate in ordine inverso.

Nei sistemi di workflow, la compensazione è usata per gestire i fallimenti delle attività. Quando un'istanza di attività di un processo fallisce, il workflow management system porta la sua esecuzione ad un punto consistente (*save point*), cioè ad un precedente passo del processo; il save point rappresenta uno stato intermedio accettabile dell'esecuzione del processo e possibilmente un punto di decisione dove possano essere eseguite determinate

azioni per risolvere il problema che ha causato il fallimento, o scegliere un percorso d'esecuzione diverso per evitare altri errori. Le attività di compensazione saranno invocate per il "roll-back" dell'esecuzione del processo e per "disfare" (*undo*) gli effetti delle attività completate.

È più complicato implementare la compensazione nei sistemi di workflow che non nei database per due ragioni :

1. le specifiche della compensazione (ad es. quando, che cosa, e come compensare) è più difficoltosa, causa la complessità delle attività e processi. Molte volte è più difficile l'attività di compensazione che l'originale attività da compensare.

2. È molto importante l'ottimizzazione di un processo di compensazione (ad es. quali attività non necessitano di compensazione); a differenza delle transazioni dei database, che possono essere compensate efficientemente e facilmente, la compensazione nel workflow può essere molto costosa. È quindi importante evitare compensazioni non necessarie.

Esistono progetti di ricerca riguardo la specifica statica degli scopi della compensazione. Ad esempio, si discute di un miglioramento di IBM FlowMark che permette al progettista dei processi di specificare (a design-time) le sfere di compensazione per determinarne lo scopo e fino a che punto essa agisca in caso di fallimenti. Il fallimento di un'attività può causare la compensazione della sola attività coinvolta, dell'intera area che la coinvolge o anche di tutte le aree che da essa dipendono. Un simile approccio è stato anche proposto per i processi gerarchici. Lo scopo della compensazione è determinato in maniera "bottom-up": prima si ritorna al determinato save point nella transazione contenente l'attività fallita, quindi al determinato save point del livello più alto che contiene la transazione se il corrente livello non può gestire l'errore.

Riassumendo, lo scopo della compensazione è annullare gli effetti che hanno causato il fallimento di unttività in modo che essa possa riprendere; quindi un'attività di workflow necessita di compensazione se essa contribuisce al fallimento del processo, e/o se la sua ri-esecuzione è diversa da quella originale. È comunque possibile che un'attività specificata in uno statico progetto di compensazione non contribuisca a un particolare fallimento. Per esempio, un attività a_1 interagisce con un'attività a_2 se un'altra attività concorrente a_3 è eseguita prima di a_1: non c'è bisogno di compensazione per l'attività a_1 se a_2 fallisce prima che anche a_3 sia partita. Quest'informazione, comunque, non sarà disponibile se non a run-time.

Nei DBMS, una compensazione è corretta se tutte le azioni tra il save point e il punto di fallimento (*failure point*) sono compensate e nell'esatto

ordine inverso dell'originale esecuzione. Nei workflow systems è necessario un più "rilassato" criterio per la compensazione a solo scopo di ottimizzazione (ad esempio non richiedendo un ordine per compensazioni commutabili).

Controllo della Concorrenza

Il controllo della concorrenza è una tecnica classica sfruttata nei DBMS che assicura l'isolamento dell'esecuzione di una transazione dalle altre. Sebbene il controllo della concorrenza sia stato considerato non necessario o troppo costoso per molte applicazioni di workflow, esso può essere molto importante per quelle in cui vi sono operazioni "vitali" (*mission-critical*) che richiedono una costante supervisione dell'ambiente di esecuzione.

Nei database il controllo della concorrenza consiste nell'isolamente delle transazioni tramite operazioni di read/write atomiche, visibili al DBMS. Nei sistemi di workflow, il WFMS assicura l'isolamento dell'esecuzione delle attività di workflow tramite operazioni di read/write atomiche mentre le operazioni esterne sono ad esso invisibili. Il WFMS è responsabile della consistenza dell'intero ambiente d'esecuzione, che include sia il database interno, visibile al WFMS, che i sistemi esterni, invisibili ad esso.

La serializzabilità, come usata per le transazioni dei database, è troppo restrittiva per molte applicazioni di workflow. La ragione principale è il fatto che le attività di workflow hanno generalmente una lunga durata; sarebbe quindi inaccettabile schedulare sequenzialmente le attività in conflitto come operazioni di read/write delle transazioni dei database. Per assicurare una corretta e funzionale esecuzione dei processi è necessario adottare un criterio di correttezza più "rilassato". È presente anche una ricerca che adotta le tecniche dei database ma permette una flessibile specifica dei requisiti della consistenza: per esempio, raggruppando una collezione di attività di un processo di workflow in un'unità consistente e utilizzando il tradizionale controllo della concorrenza per assicurare l'isolamento delle unità (in termini di serializzabilità). La corretta esecuzione delle attività all'interno di un'unità consistente è assicurata dal rafforzamento delle dipendenze tra i dati e le esecuzioni.

2.5.2 Dynamic Workflow

Sebbene vi siano ricerche e anche prodotti sul mercato che permettono la modifica della definizione dei processi a run-time, questa possibilità è ancora considerata essere rara e costosa.

I sistemi di workflow *dinamici* sono sistemi speciali che non prevedono "pre-specificate" definizioni dei processi. Essi partono con qualche attività

iniziale; quando un'attività termina la sua esecuzione, ne saranno selezionate altre in base allo stato d'esecuzione e ai risultati del corrente task. In altre parole, il workflow è allo stesso tempo specificato ed eseguito.

Questi sistemi sono adatti per applicazioni di workflow dove la specifica dei processi è frequentemente modificata o non può essere pre-specificata. Per esempio, molti progetti partono con un task iniziale (ad es. per controllare i requisiti) e seguono una linea d'esecuzione generale. Task diversi sono eseguiti in ordine diverso in base allo stato d'esecuzione e al progetto iniziale.

Per questi tipi di sistemi molti argomenti di ricerca appena visti necessitano di cambiamenti. Per esempio, le specifiche dei requisiti della consistenza sarà diversa, poiché manca una visione completa del processo. Per la stessa ragione, è molto complicata la coordinazione (soprattutto in ambienti distribuiti) dell'esecuzione delle attività. Recenti ricerche hanno affrontato il workflow dinamico cercando di implementare tali sistemi utilizzando "agenti mobili"; tali lavori sono però ancora nelle fasi iniziali e non rigurdano gli argomenti appena trattati.

Capitolo 3

Il Workflow in ambiente distribuito

3.1 Architetture di rete

3.1.1 Modello Client-Server

Il termine "Client-Server" indica il rapporto d'interazione che esiste tra il Server ed il Client ed indica un modello in cui i compiti relativi ad un'elaborazione sono suddivisi tra Server e Client. In una situazione normale ("stand-alone") la nostra applicazione, ad esempio un database, è formata da un eseguibile (il DBMS) e da uno o più file di dati.

Quando abbiamo bisogno di un'informazione, attraverso l'eseguibile eseguiamo tutte le operazioni necessarie al trattamento dei dati ed alla visualizzazione della risposta. Tutte le operazioni vengono eseguite dalla stessa applicazione. La tecnologia Client Server spezza le applicazioni in due parti, ciascuna specializzata per un compito, ciascuna assegnata al dispositivo HW/SW meglio attrezzato per eseguirle. In altre parole, da un lato abbiamo una stazione (ad es. un PC) che ha funzioni di front-end, sulla quale gira un SW che si occupa di comporre le richieste e di visualizzare le risposte, dall'altro abbiamo una macchina più potente, con funzioni di back-end (un Server di rete), sul quale gira un SW che si occupa di recepire le richieste, elaborare le risposte, inviarle lungo la rete sino al Client che ha dato il via al processo.

3.1.2 Modello Time Sharing e Resource Sharing

L'architettura Client Server è considerata un modello molto efficiente per l'elaborazione dei dati e rappresenta l'evoluzione dei modelli Time Sharing dei

Mainframe ed è passata per il modello Resource Sharing, ancora oggi molto diffuso. Mentre il modello Time Sharing è formato da un host e da terminali "stupidi", il modello Resource Sharing affida al terminale, questa volta una stazione intelligente, il compito di elaborare le informazioni necessarie.

Nel primo modello il Server, di solito un potente mainframe o un minicomputer molto dotato, si assume l'onere di tutte le operazioni, lasciando al terminale solo la funzione di display per il Data-entry.

Nel secondo modello il Server è semplicemente un File Server, in grado di mettere a disposizione file e stampanti. La capacità elaborativa è tutta sul client.

Un esempio (database del personale): se un utente avesse bisogno di conoscere la media delle assenze, calcolata per ciascuna categoria di personale di un determinato mese, avrebbe, nel modello Time Sharing, attivato un programma sul Mainframe attraverso il quale sarebbe stata formulata la richiesta dell'informazione, sempre sul Mainframe verrebbero elaborati i dati e formulata la risposta. In una architettura di tipo Resource Sharing, l'utente avrebbe attivato un applicativo sul Client, avrebbe richiesto al File Server i dati necessari all'operazione che li avrebbe inviati tutti al client, il quale si sarebbe occupato dell'elaborazione e della visualizzazione della query.

Quali sono i vantaggi e quali gli svantaggi ?
Il mondo dei Mainframe è il sofisticato mondo dei super elaboratori, grande potenza elaborativa, velocità, sicurezza dei dati ecc. I costi sono altissimi sia per l'acquisto che per la manutenzione conservativa dell'hardware e del software. La manutenzione evolutiva richiede tempi lunghi e molto denaro. Il secondo modello apporta un significativo cambiamento, utilizzando come terminale intelligente un familiare PC. I costi generali sono molto più bassi ed è molto più facile eseguire un upgrade graduale di un sistema basato su PC, perché la capacità elaborativa è frazionata sulle varie macchine.

Di contro il "traffico sulla rete", ossia la quantità di dati che transita "sul filo" cresce enormemente: per eseguire una query su un archivio di migliaia di tuple, sulla rete viaggiano migliaia di tuple, perchè è sul Client che avviene l'elaborazione. Il modello Client Server combina i due modelli precedenti. Grazie al fatto che le macchine diventano sempre di più performanti a costi sempre più bassi, oggi è possibile utilizzare un potente PC come Server di Database, rendendo possibile la concentrazione di tutte le incombenze d'elaborazione più impegnative sul Server, mentre al Client è lasciata la parte d'interrogazione, visualizzazione, la parte di data entry ecc. Rifacendosi all'esempio di prima, il traffico sulla rete risulta ridotto al minimo: transita verso il Server la richiesta d'informazioni, il Server esegue tutte le operazioni per il calcolo della query sul personale e rinvia solo il risultato della elaborazione.

3.1.3 Architettura distribuita

In un'architettura distribuita abbiamo più macchine collegate attraverso una rete di interconnessione, non necessariamente omogenee come potenza elaborativa, che cooperano nell'eseguire un certo processo. In questo caso i dati che servono per l'applicazione possono essere distribuiti nei diversi siti con replica oppure no.

Nei sistemi Client Server vi sono uno o più Server che svolgono ben distinte funzioni; nelle architetture distribuite una certa applicazione è condivisa a livello di esecuzione da più macchine, ciascuna con propri dati. Per quanto riguarda i database, nei sistemi Client Server e Time Sharing il DBMS e i file di dati risiedono tutti su un'unica macchina (il Server e il Mainframe rispettivamente); nei sistemi Resource Sharing per definizione i dati risiedono solo sul file Server centrale, mentre i DBMS sono replicati sulle varie macchine intelligenti. In un'architettura distribuita ogni sito ha una porzione del DBMS che può anche metterlo a disposizione degli altri Server. Per quanto riguarda i dati abbiamo tre diverse possibilità: replicare totalmente i dati su tutti i Server, replicarli solo parzialmente, oppure destinare un solo Server per accogliere tutti i dati.

3.1.4 Vantaggi e problemi del Distributed Computing

Trasparenza dell'architettura. Uno dei maggiori vantaggi di un sistema distribuito è la trasparenza dei dettagli dell'architettura di rete. Ciò si traduce nel fatto che gli utenti richiedono i loro servizi al sistema senza sapere quali macchine li soddisferanno. Deve essere presente un meccanismo d'accesso globale che permetta ai processi di un nodo di accedere a processi in altri nodi come se essi fossero processi locali. Questa caratteristica è nota come *location tranparency*. Assicurare una trasparenza totale è comunque difficile causa la comunicazione spesso insicura e con imprevedibili ritardi della rete di comunicazione.

Tolleranza ai guasti. In certe aree applicative i guasti ai sitemi informatici sono spesso fatali. I sistemi distribuiti tornano utili in quanto non c'è un solo punto di possibili guasti; grazie al meccanismo della replica dei dati, le informazioni perdute presso un sito, possono essere recuperate in un altro, dove ne è presente una copia. Chiaramente la gestione della consistenza dei dati è un altro argomento che sarà trattato separatamente.

Condivisione delle risorse. Le risorse nei sistemi distributi possono essere condivise da più utenti in siti diversi, e poiché una risorsa può essere

acceduta da nodi remoti, non è necessaria una replica di tutte le risorse in tutti i nodi.

Chiaramente ci sono molti problemi: l'eterogeneità, la mancanza di un clock comune, spazi di indirizzamento diversi, ritardi e malfunzionamenti della rete di comunicazione e comportamenti non prevedibili dell'interazione di processi concorrenti sono solo alcune delle tipiche problematiche dei sistemi distribuiti.

Sicurezza. La sicurezza è un altro arduo problema di questi sistemi. Non è un problema piccolo neanche per sistemi centralizzati; una sicurezza globale deve fornire la possibilità di autenticazioni e controlli degli accessi al sistema univoco in tutti i siti, anche remoti.

Inoltre nascono nuovi problemi come "side-effects" dei sistemi distribuiti. Per esempio, il problema della consistenza dei dati può crescere causa la replica (utilizzata per incrementare l'affidibilità e la disponibilità di questi sistemi) nei diversi siti.

3.1.5 Modelli di comunicazione nei sistemi distribuiti

Uno dei più importanti aspetti, e spesso anche "collo di bottiglia", nei sistemi distribuiti, è la comunicazione tra i vari computer (*nodes*).

Si tratteranno ora due tipi di modelli di comunicazione sviluppati per questi sistemi.

- *Scambio di messaggi* : la comunicazione basata sullo scambio di messaggi prevede che i processi interagiscano tramite l'invio di messaggi, utilizzando le primitive *Send* e *Receive*. La primitiva *Send* è affidabile e prevede la ritrasmissione del messaggio fino a che non è ricevuto con successo dal nodo di destinazione. I messaggi sono inviati a "porte", invece che direttamente ai nodi, dove vengono automaticamente ricompattati in flussi di dati indipendenti nei nodi riceventi.

- *Remote Procedure Call* : la comunicazione basata sulle Remote Procedure Call (RPC) fornisce al programmatore la possibilità di invocare procedure remote (localizzate in siti remoti) allo stesso modo di una procedura locale. Le RPC possono essere sincrone (in cui la comunicazione si riduce a un meccanismo di chiamate/risposte bloccanti) o asincrone (dove il chiamante non si preoccupa del successo o meno della chiamata).

  Queste procedure rappresentano il passaggio dall'usuale programmazione centralizzata ad un tipo di programmazione distribuita, in cui non

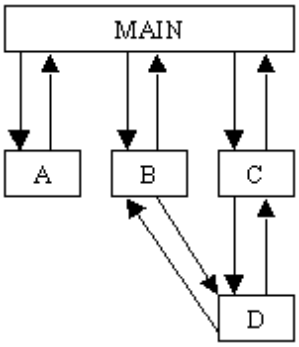


Figura 3.1: Programmazione centralizzata

solo si hanno i dati situati su siti diversi, ma anche porzioni di codice (procedure e funzioni) che possono essere eseguite in remoto mediante una "chiamata a procedura remota".

Lo stile di programmazione centralizzata, utilizzato attualmente per la realizzazione di programmi in ambienti non distribuiti, è basato sulla chiamata di procedure: ciascuna svolge un determinato e ristretto compito. Un programma può essere rappresentato schematicamente in figura 3.1.

In un sistema distribuito è utile associare ai dati le procedure per la loro gestione, così che l'elaborazione è più veloce in quanto i dati sono sempre locali all'applicazione. Sulla rete passano solo i parametri di input ed eventualmente ritornano i parametri di output. La figura 3.2 rappresenta schematicamente un'esempio di chiamata RPC.

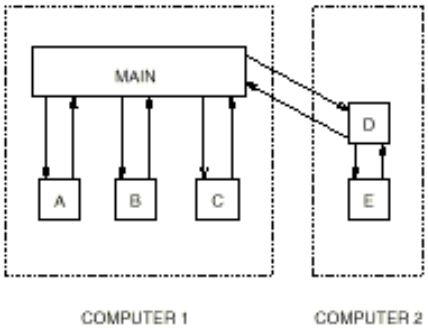


Figura 3.2: Schema d'esempio di una *RPC*

Le RPC nate per gli ambienti client/server, in cui si sfruttava la maggior

potenza di calcolo del server anche per eseguire le procedure (*computational server*), sono poi state estese ai sistemi distribuiti: ogni server possiede le procedure necessarie per la gestione dei propri dati.

Analizziamo il funzionamento delle RPC nei sistemi distribuiti, a tal fine consideriamo due siti: uno lo definiamo "server" in quanto possiede le RPC ed è il sito in cui avviene l'elaborazione dei parametri della procedura, l'altro "client" poichè richiede i servizi al server. Occorre notare che questa terminologia è utilizzata unicamente per stabilire chi offre e chi richiede un servizio in un determinato intervallo temporale, infatti i ruoli non sono fissi e dipendono dalla procedura che si deve utilizzare (il sito che l' ha definita è il server). Lo spazio di indirizzamento rimane sempre distinto, le chiamate avvengono unicamente attraverso lo scambio di pacchetti attraverso la rete. La Figura 3.3 mostra come avviene una chiamata.

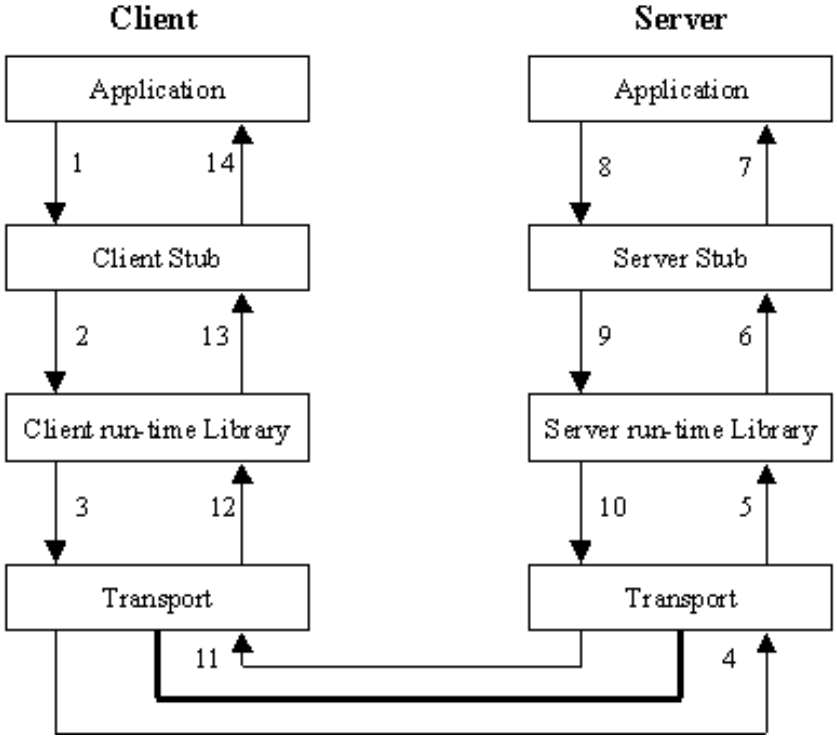


Figura 3.3: Schema di una chiamata ad una procedura remota

Il client effettua la chiamata localmente ad una propria interfaccia (*client stub*), da questo livello inizia, attraverso un mapping che permette di localizzare la procedura, l' esecuzione vera e propria che com-

porta il trasferimento dei parametri verso il server. Il client effettua le seguenti operazioni:

- recupera i parametri necessari dal proprio spazio di indirizzamento;
- trasforma i parametri seguendo lo standard della rete di comunicazione (il *Network Data Representation* NDR);
- chiama la funzione nella libreria run-time locale che mappa la RPC remota e passa i parametri richiesti.

A questo punto è il sistema operativo ad occuparsi dei dettagli implementativi per l' esecuzione della richiesta. Il server risponde ad una RPC eseguendo i seguenti passi:

- la libreria run-time del server accetta l' RPC e preleva i parametri dalla rete;
- i parametri vengono passati allo stub che e ettua la conversione dal formato NDR al formato dei dati richieso dal server;
- viene eseguita la procedura.

Quando la procedura sul server è terminata, il risultato prodotto viene ritornato al client eseguendo una sequenza di operazioni simili alle precedenti, dal server verso il client.

3.2 Il Workflow in ambiente distribuito

I sistemi di workflow sono per natura distribuiti: le applicazioni esterne che eseguono i task di workflow sono spesso geograficamente decentrate e inoltre anche i WFMS possono essere essi stessi distribuiti. La caratteristica comune dei WFMS distribuiti è la distribuzione delle funzioni: componenti di workflow diversi che eseguono varie funzioni di workflow, come la definizione, l'esecuzione, il monitoraggio dei processi, e l'assegnamento delle risorse, sono localizzati in siti diversi. I componenti dei WFMS interagiscono a vicenda tramite scambio di messagi o RPC. Un'altra forma di distribuzione è l'esecuzione di funzioni di workflow da parte di componenti di WFMS ultipli e funzionalmente equivalenti che condividono memorie comuni. Per esempio, l'esecuzione di un processo di workflow può essere eseguito collettivamente da diversi workflow engine che condividono gli stessi dati per la definizione

dei processi e gli stati d'esecuzione. Tali sistemi forniscono una migliore scalabilità e resilienza ai guasti del workflow engine, ma sono ancora vulnerabili a errori nella memorizzazione dei dati.

La difficoltà maggiore della distribuzione è l'esecuzione collettiva di un processo da parte di WFMS multipli e indipendenti (che non condividono i dati) . In tali sistemi, ogni WFMS è un completo sistema di workflow con il proprio engine e dati, e non c'è un server centrale che mantiene tutte le informazioni sull'esecuzione di tale processo. Questi sistemi sono migliori sia in sicurezza che prestazioni: le attività di workflow possono essere eseguite dai WFMS che sono più vicine alle corrispondenti applicazioni esterne (risparmiando i costi di comunicazione tra WFMS e applicazioni) e i WFMS accedono localmente sia alla definizione dei processi che agli stati d'esecuzione (riducendo i costi di comunicazione tra WFMS e dispositivi di memorizzazione dei dati). Sono sistemi anche più affidabili perché un problema a uno o più WFMS (incluso il corrispondente dispositivo di memorizzazione) non fermano l'esecuzione del processo.

Se si vuole implementare questi sistemi in una realtà aziendale, bisogna anche affrontare i problemi della replica dei dati e della coordinazione dell'esecuzione dei processi. La replica dei dati è necessaria per assicurare un'affidabile esecuzione dei processi; per esempio, un processo in esecuzione può "sopravvivere" a un guasto del singolo WFMS se la sua definizione e gli stati d'esecuzione sono replicati su più WFMS indipendenti. La replica dei dati però può risultare molto costosa; essa può essere fornita dai sistemi di workflow stessi o, come nel progetto della presente tesi, dal DBMS sottostante. Il vantaggio maggiore di un sistema di workflow che incorpora già il meccanismo di replica è la flessibilità: ad esempio i processi di workflow possono essere eseguiti a diversi livelli di affidabilità, senza replica (efficiente ma vulnerabile a guasti dei singoli WFMS) o con replica totale (oneroso ma affidabile).

La coordinazione dell'esecuzione è necessaria quando più di un WFMS, collettivamente, esegue un processo di workflow. Per esempio, l'esecuzione di un'attività da parte di un WFMS può causare la sospensione dell'esecuzione dell'intero processo, influenzando tutti gli altri WFMS; la chiave è trasferire le informazioni del processo a un sito quando è necessario e nel giusto ordine. Le informazioni statiche, quali la definizione dei processi, possono essere replicate in tutti i siti importanti, ma le informazioni a run-time, come gli stati delle istanze di processo, devono essere trasferite, sempre a run-time, da sito a sito. Ciò può essere fatto sia facendo circolare le informazioni riguardanti un processo e la sua esecuzione attraverso i diversi siti, o pre-compilando la definizione dei processi per determinare in quali siti le attività debbano essrer eseguite (vedi [MAG+95]). Il vantaggio del primo approccio è la fles-

sibilità, nel senso che un WFMS può scegliere di eseguire un'attività in un sito in accordo all'ambiente d'esecuzione a run-time. Lo svantaggio è l'alto costo della comunicazione, poichè il "package" di informazione potrebbe essere molto ampio. L'ultimo approccio, al contrario, può essere facilmente implementato poiché solo le informazioni rilevanti sono trasferite al sito, ma è poco flessibile. Per esempio, se un'attività è pre-assegnata a un sito non accessibile in un certo momento, gli altri siti non possono riprendere l'esecuzione se non hanno le informazioni. Un altro problema è che molti prodotti di workflow assegnano le risorse a un'attività solo a run-time; il sito che viene pre-assegnato per eseguire un'attività di workflow quando vengono date le specifiche del processo, può localizzarsi troppo lontano dalle risorse per poterle invocare.

Il controllo della concorrenza e la compensazione possono anche complicarsi quando il WFMS è distribuito: se ad esempio l'esecuzione di attività in conflitto su siti diversi devono essere coordinate per assicurarne la consistenza di tutta l'esecuzione. Il meccanismo del "locking" non è generalmente accettabile, poiché le attività di workflow hanno un tempo d'esecuzione molto lungo. La serializzabiltà, invece, può anche non essere necessaria all'esecuzione.

Nel paragrafo seguente verranno illustrate le funzionalità di un sistema per il workflow client-server, Action Workflow System, e nel capitolo 5 verranno descritti i passi per rendere questo sistema distributo.

3.3 Action Workflow System

Nella tesi sono stati studiati ed utilizzati due sistemi commerciali per la parte implementativa del lavoro: Action Workflow System della Action Technology Inc. e il DBMS Microsoft SQL Server. Il primo, un WFMS client-server che verrà descritto nei paragragi successivi, sfrutta come back-end server le funzionalità di DBMS di SQL Server, al quale sarà dedicato il capitolo sucessivo.

Analizziamo ora le caratteristiche di Action Workflow System.

Componenti funzionali

Nella Fig. 3.4 sono mostrati i componenti funzionali di Action Workflow System. Dalla parte del server, Action Workflow System consiste di un motore di workflow, il Process Manager, due file di rete e cinque classi di tabelle di

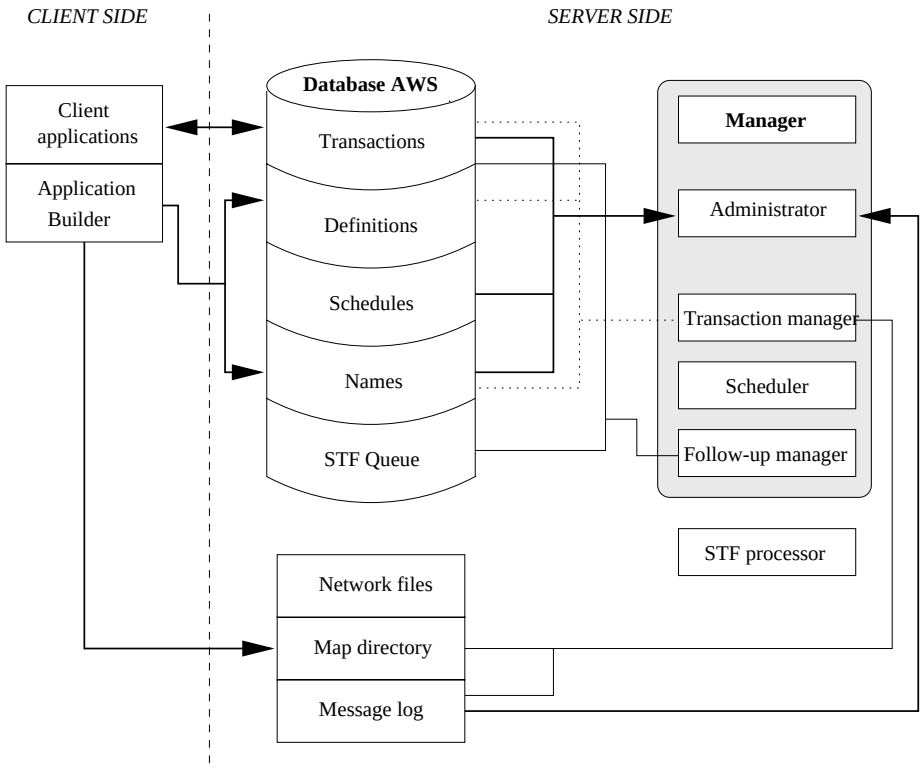


Figura 3.4: I componenti funzionali di ActionWorkflow

database (di nome *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. L'Action Workflow Manager è fornito di un'interfaccia utente (vedi Fig 3.3), attraverso cui l'amministratore di sistema accede alle tabelle ed al file di log; esso sfrutta tre processi di sistema, attivi in memoria:

- 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.

Il Follow-up Manager memorizza i messaggi nella quinta tabella (la coda STF), gestita dall'*STF (Standard Transaction Format) Processor*, che si connette ad Action Workflow 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 Action Workflow System), gestisce le tabelle delle definizioni e dei nomi e il direttorio delle mappe.

Action Workflow 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.
Per quanto riguarda il database *aws*, possiamo dire che in esso vengono memorizzate, all'inizio, tutte le informazioni necessarie a ricostruire la definizione di processo (la mappa generata dal Process Builder e compilata dal Manager, più il mappaggio ruoli-identità). Queste informazioni vengono poi lette dalle applicazioni client 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). Ulteriori approfondimenti sulla struttura del database verranno fatti in seguito, in quanto su di esso si basa la distribuzione del WFMS.

Architettura del sistema

L'architettura tecnica del sistema è mostrata in fig. 3.3; I server del sistema Action Workflow, sfruttano tutti un unico sistema di memorizzazione e di comunicazione tra le applicazioni client e il Manager. Le applicazioni client, così come il Process Builder, possono risiedere sia su sistemi Windows NT, sia su sistemi Windows 95 (98) e Windows 3.11. Il Manager invece, che si appoggia su un database Sql Server, può risiedere solo su un server Windows NT.

Ogni componente condivide lo stesso livello virtuale di database, che permette ai sistemi di accedere al database server SQL.

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. Innanzi tutto, per poter

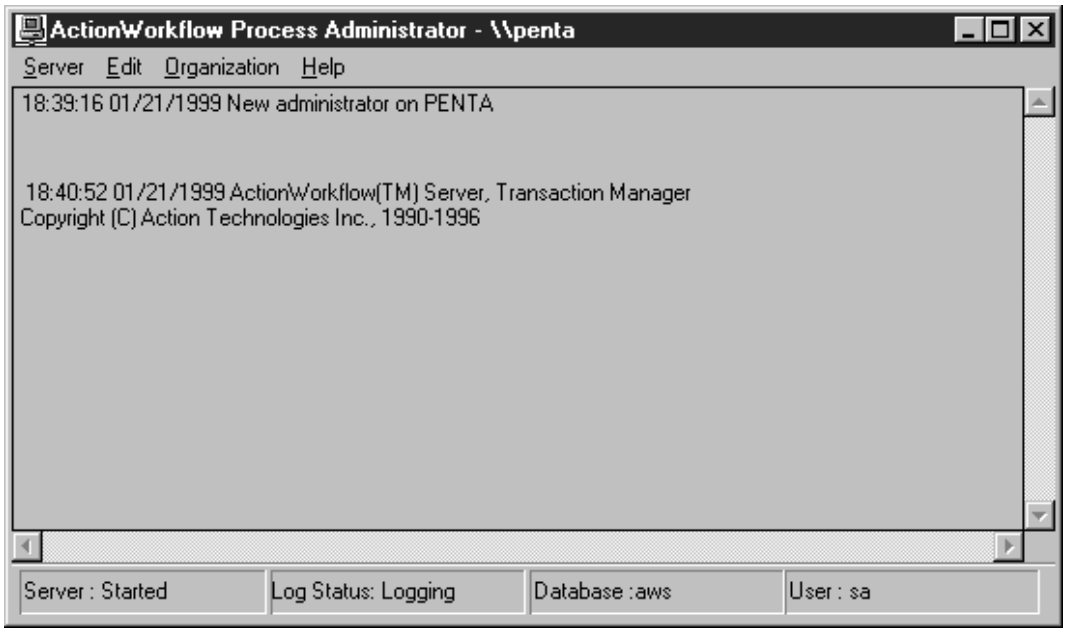


Figura 3.5: Schermata dell'Action Workflow Administrator

lavorare è necessario connettersi all'SQL Server su cui è installato il database di Action Workflow (e il processo *aws_srvn*), poi lanciare manualmente il Process Manager. 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, da generare a parte tramite strumenti di sviluppo quali il Visual Basic o Visual C++. A corredo del sistema è fornita una applicazione di esempio, la *Call Track Application*, che permette al programmatore di estrarre utili aiuti per generare nuove applicazioni.

La Call Tracking Application è una applicazione sviluppata in Visual Basic 4.0. Con il file eseguibile 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 Action Workflow System fanno frequente riferimento a quest'esempio, che viene spiegato e commentato in ogni sua parte.

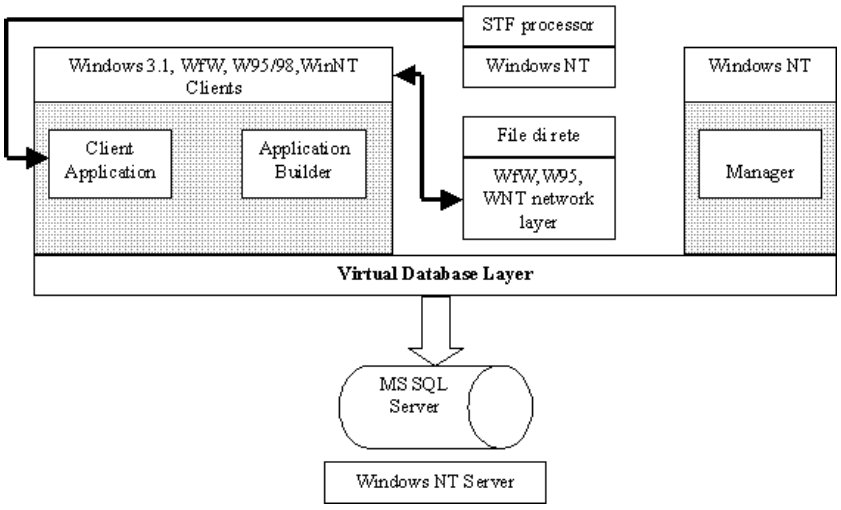


Figura 3.6: Architettura di Action Workflow System

L'applicazione riguarda la procedura aziendale di analisi e correzione di problemi nel software, in seguito ad una richiesta di un cliente. La mappa (in Fig 3.7), 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 preparata la mappa , bisogna compilarla dal Process Builder generando un nuovo file di estensione ".awo"). Questa operazione (che include anche il controllo di consistenza, controllando che siano rispettate tutte le regole di definizione della mappa) chiude la sessione di lavoro sul Process Builder. A questo punto, dal Process Manager, si "installa" la definizione di processo (l'Administrator eseguirà una nuova compilazione della mappa, in particolare degli script), si aggiungono le "identità" e i "ruoli"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).

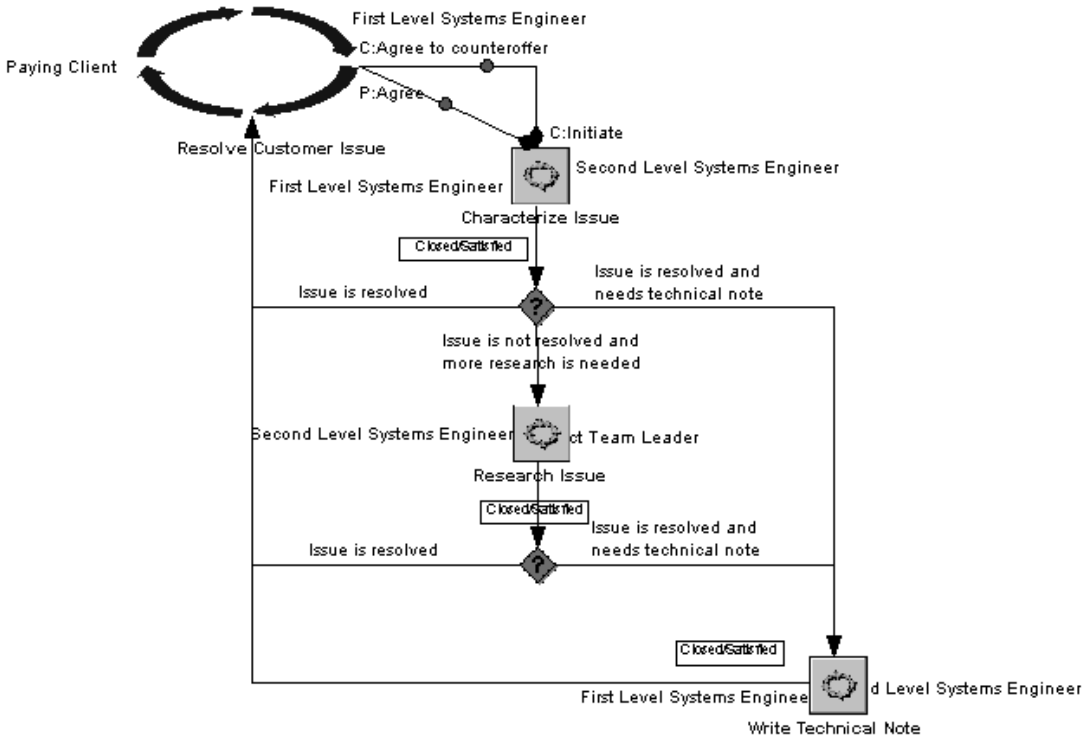


Figura 3.7: La mappa della Call Tracking Application

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.

. Di grande utilità per lo sviluppo di nuove applicazioni, è il file che contiene, già implementate tramite le API di Action Workflow System (contenute nella libreria dinamica `AWSAPI.DLL`), tutte le funzioni citate sopra: dal recupero delle worklist, all'esecuzione di un atto, al login nel sistema.

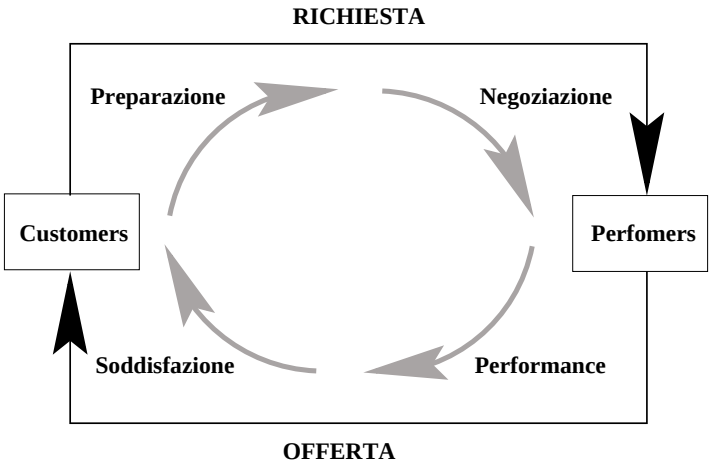


Figura 3.8: Workflow Loopsecondo ATI

3.3.1 Modellazione dei processi secondo ATI

Come descritto nel Cap.1, la modellazione dei processi di workflow sviluppata presso l'Action Technology Inc., si basa sul "workflow loop" o "conversazione";in esso, non troviamo nessun riferimento ad *attività* o *task* (come definito nello standard della WfMC), mentre l'unico termine utilizzato è quello di *workflow*. Il processo è visto come un insieme di cicli workflow, uno principale, gli altri secondari (sottoprocessi). Ogni singolo ciclo 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 3.8). 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.
Ogni processo deve seguire alcune regole base:

- *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*.

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.

Action Workflow 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 Action Workflow 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, Action Workflow 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 Action Workflow 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 Action Workflow System è fornito di un linguaggio (Action Workflow 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 Action Workflow 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.

3.3.2 Note critiche su Action Workflow

Diamo ora un breve elenco degli aspetti positivi e negativi del sistema da noi utilizzato. Gli aspetti positivi del sistema possono riassumersi nelle seguenti considerazioni:

- le regole da seguire per il disegno della mappa del business process, assicurano una definizione dei processi sicura ed efficiente;

- l'attenzione è rivolta alle responsabilità delle persone;
- è un sistema molto flessibile, in quanto lo sviluppatore può crearsi ad hoc le interfacce grafiche al "motore" di workflow.

I principali aspetti negativi invece sono:

- si forzano le persone ad avere interazioni di tipo ciclico, invece di rapporti diretti, come avviene attraverso concatenazioni di attività e task;
- non si possono sviluppare applicazioni distribuite;
- manca una concreta ed efficiente gestione dei dati;
- la versione da me utilizzata si è dimostrata un sistema molto fragile, infatti i malfunzionamenti sono frequenti, dal blocco della compilazione delle mappe (ad esempio per un timeout di SQL Server), ai problemi sorti all'atto dell'installazione di nuove definizione dei processi dopo averne cancellati altri (era necessario infatti far ripartire il Process Manager e "ripulire" il database *aws* a mano, in quanto la ricostruzione del database da Setup di programma non inizializzava alcune tabelle);

Prima di descrivere le sostanziali modifiche apportate al sistema Action Workflow (per questo si rimanda al Cap. **??**), dalla distribuzione del database *aws*, al supporto di una gestione dei dati distribuita tramite l'integrazione del modello DOM, si riporta ora un capitolo dedicato alla funzionalità di DBMS distribuito di SQL Server; infatti gran parte del lavoro della tesi si è svolta proprio su questo sistema.

Capitolo 4

SQL Server 6.5: funzionalità di Distributed Computing

4.1 Caratteristiche generali

Le caratteristiche di potente RDBMS disegnato per un ambiente client/server per piattaforma Windows NT, già presenti in SQL Server 6.0, sono state potenziate e arricchite da tutte le funzionalità necessarie per la realizzazione di un sistema completo di Distributed Computing (Elaborazione Distribuita). Tra queste sono comprese :

- *Distributed Management Framework* : ambiente per l'amministrazione centralizzata di architetture distribuite.

- *Distributed Transaction Coordinator* : integrato con SQL Server, il DTC coordina e fornisce strumenti per la strutturazione dell'esecuzione di transazioni suddivise in un insieme distribuito di componenti software, sia su un singolo computer che su un sistema di computer in rete. Con un sistema di chiamate a Stored Procedures remote fa sì che manipolazioni di dati su server multipli vengano considerati come una singola unità di lavoro. Attraverso l'interfaccia grafica di SQL Enterprise Manager è possibile controllare l'esecuzione di una transazione e il suo stato, risolvere manualmente eventuali problemi e, quando necessario, forzare l'esecuzione delle query.

- *Internet Database Connector* e *SQL Server WEB Assistant*: IDC permette ad un utente WEB di inviare la sua query, dall'interno di una pagina HTML, sia per il reperimento che per l'aggiornamento dei dati

in un database. Si tratta di un'applicazione che utilizza le API ODBC e che consente di creare collegamenti tra campi di pagine HTML e SQL Server senza la necessità di scrivere CGI (Common Gateway Interface), le quali collegavano, tramite la scrittura di righe di programma, in maniera statica, i singoli campi da visualizzare con il foglio HTML; le CGI erano onerose da definire e difficili da mantenere.
Il WEB Assistant permette la pubblicazione di dati via WEB. Grazie anche alla presenza di stored procedures specifiche - (`sp_makewebtask, sp_runwebtask, sp_dropwebtask`), è possibile attivare precessi dinamici per la formattazione dei dati in formato HTML. Esso fornisce anche un'interfaccia grafica per la creazione di siti WEB database-driven.

Un'ultima ma fondamentale caratteristica di SQL Server 6.5 è la capacità di replica dei dati; questa funzionalità verra approfondita nelle sezioni seguenti.

4.2 Replica dei dati

La replica è il meccanismo che permette l'allineamento dinamico, in tempo reale o temporizzato, di database (dati) disseminati sulla rete.

Il meccanismo della replica di SQL Server si rifà alla metafora Publisher/Subscriber. Il modello deriva dal fatto che quando un dato è reso disponibile per la replica è detto "published" (pubblicato); il server che richiede e ottiene un dato published è detto "Subscription Server" o "Subscriber".

Il server che contiene i dati sorgenti, pronti per essere resi published è chiamato "Publication Server" o "Publisher". Secondo una configurazione di replica standard, i dati replicati si "muovono" solo in una direzione, dal Publication Server ai Subscription Servers. In questo modo significa che i dati sono trattati dagli utenti del Subscriber come read-only (vedi Fig. 4.1). Nei capitoli sucessivi si descriverà invece la procedura (gli script da installare) per configurare una tipologia di replica di tipo bidirezionale, dove i due o più server sono sia Publisher che Subscriber.
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 tut-

Figura 4.1: Replica dei dati unidirezionale

ti 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.

4.2.1 Componenti del sistema di replica

I principali componenti, illustrati nella Fig 4.2, sono:

- Synchronization Process: esso prepara i file di sincronizzazione iniziale contenenti sia lo schema (lo script SQL che può creare, se non esiste, la tabella di destinazione in remoto) che i dati da replicare; memorizza i file nella directory di lavoro di SQL Server (`\MSSQL\REPLDATA`) registra i job di sincronizzazione nel distribution database. Il processo di sincronizzazione ha effetto solo sui nuovi subscriber.

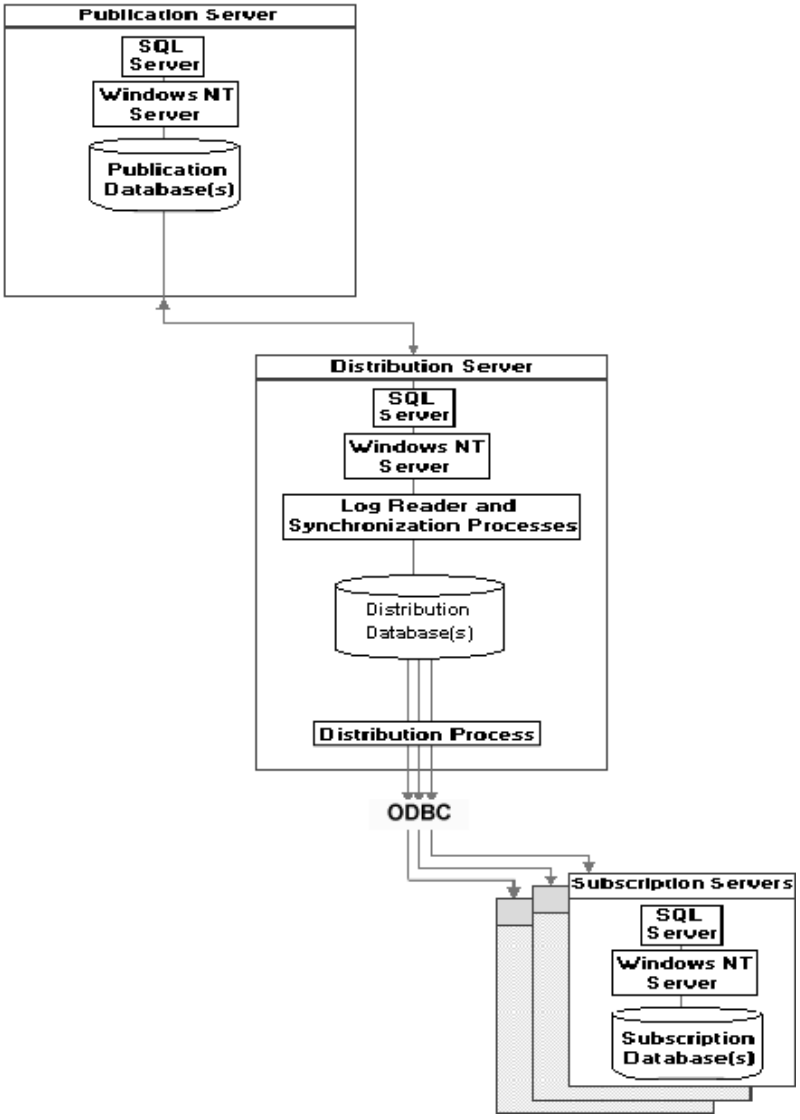


Figura 4.2: I componenti del sistema di replica di SQL Server

- Distribution Database: è 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: 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: è 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.

4.2.2 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 4.3).

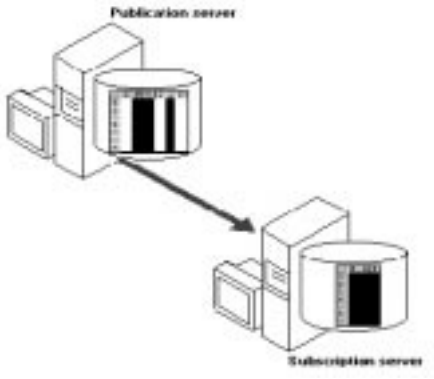


Figura 4.3: Partizione verticale

– Pubblicazioni di tabelle partizionate orizzontalmente: possiamo creare articoli di replica costituiti da alcune righe di una tabella (frammentazione orizzontale, vedi Fig 4.4).

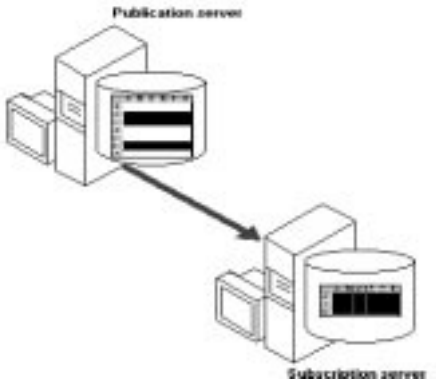


Figura 4.4: Partizione orizzontale

– Pubblicazioni di tabelle partizionate in modo mixed: possiamo partizionare una tabella in modo verticale e orizzontale contemporaneamente (vedi fig 4.5).

- 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.

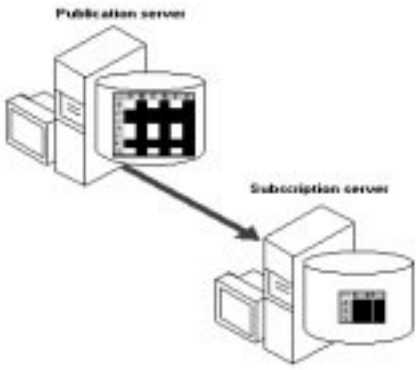


Figura 4.5: Partizione mista

- 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 uno o più 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).

4.2.3 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.

4.2.4 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.

4.2.5 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

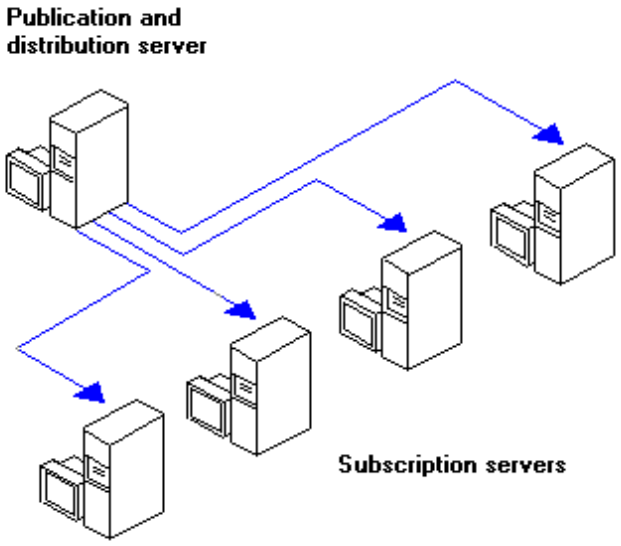


Figura 4.6: Tipologia central publisher

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

4.2.6 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, ...).

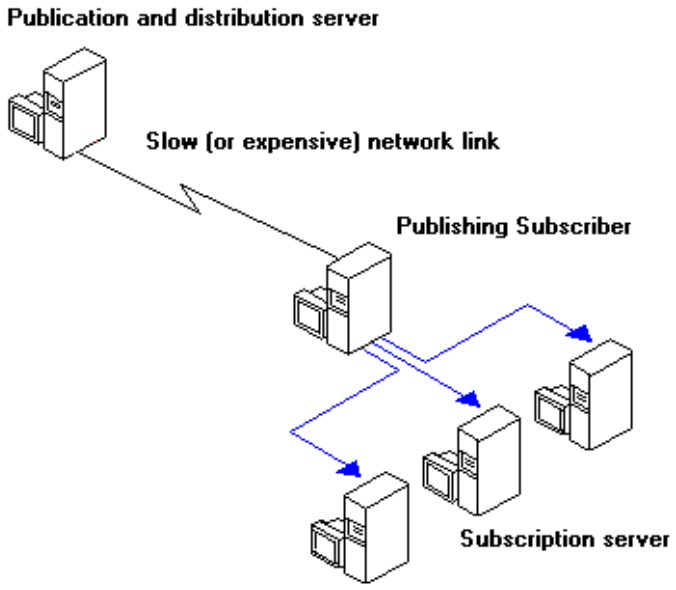


Figura 4.7: Tipologia Publishing Subscriber

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 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.

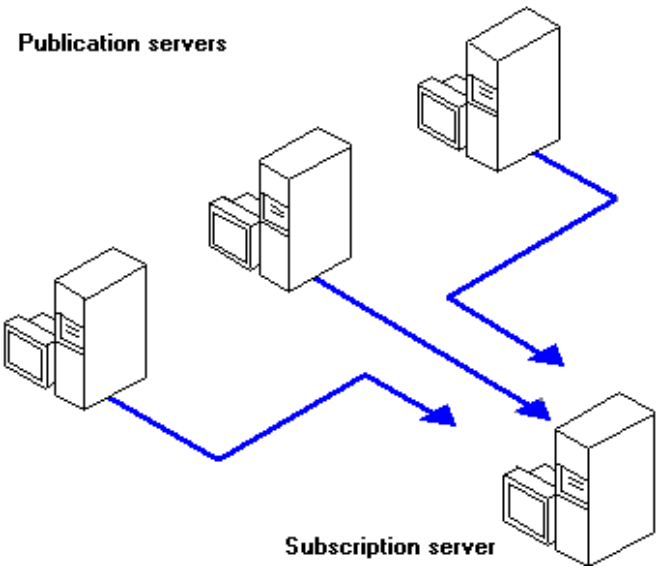


Figura 4.8: Tipologia central subscriber

4.2.7 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 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.

4.2.8 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

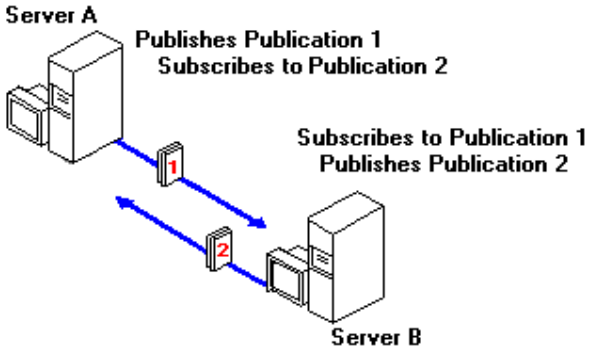


Figura 4.9: Tipologia Publisher/Subcriber

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.

4.2.9 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 permet-

[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.

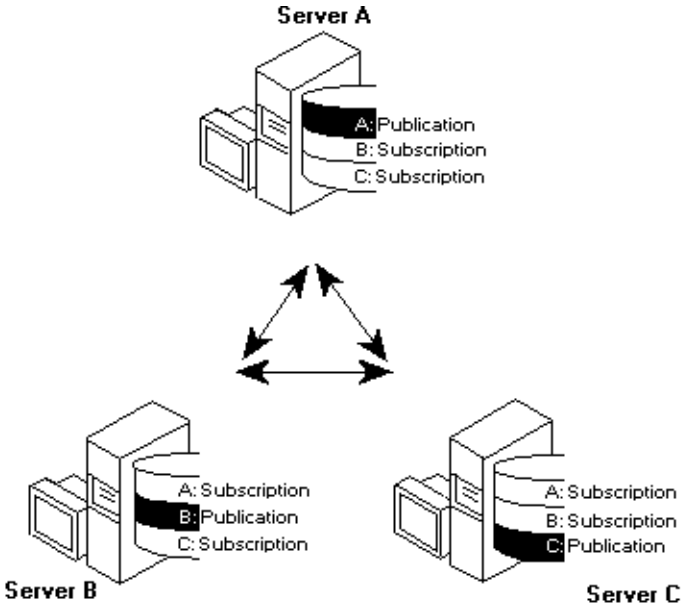


Figura 4.10: Tipologia multiple publisher of one table

tono di scrivere ed eseguire transazioni distribuite e chiamate a procedure remote, che sono stati necessari per l'implementazione del lavoro svolto.

4.2.10 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.

4.2.11 Stored Procedures utilizzabili nella Replica

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".

- `sp_changesubscriber` permette il cambiamento di un subscription server. Tutti i task di distribuzione per i subscribers del publisher saranno modificati.

- `sp_replsync` usata da un subscription server per conoscere il completamento di una sincronizzazione manuale.

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`, `sp_helpsubscriberinfo`.

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.

- `sp_enumfullsubscribers` ritorna una lista di tutti i subscribers che hanno sottoscritto tutti gli articoli di una publication.

Le altre funzioni di questo gruppo (oltre alle corrispondenti funzioni "drop" per rimuovere pubblicazioni (`sp_droparticle, sp_droppublication`) e articoli e alle funzioni "help" di debug (`sp_helparticle, sp_helparticlecolumns, sp_helppublication, sp_helppublicationsync, sp_helpreplication db`)) 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 (`sp_changearticle, sp_changepublication`. 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 aggiungere un articolo alla subscription.

- `sp_unsubscribe` cancella in remoto una sottoscrizione ad un particolare articolo di una o più publication.

Anche in questo caso sono presenti le funzioni di "drop" per rimuovere le subscription dai server (`sp_dropsubscription`, le "help" per visualizzare lo stato attuale per il debug (`sp_helpsubscription` e le funzioni di modifica delle proprietà di una sottoscrizione (`sp_changesubstatus, sp_changesubscription`).

Replication operations Sono le stored procedures utilizzate per le operazioni base della Replica :

- `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.

- `sp_distcounters` visualizza le informazioni più recenti su tutti i subscribers. È usato dall'SQL Performance Monitor.

- `sp_replcleanup` rimuove le transazioni dalle tabelle del distribution database dopo che esse sono state distribuite con successo ai databse del subscriber.

- `sp_MSkill_job` 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.

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.

- `sp_replcmds` ritorna i comandi per le transazioni "marcate" per la Replica.

- `sp_replcounters` ritorna le statistiche per tutti i database pubblicati.

- `sp_repldone` modifica il record identificatore dell'ultima transazione distribuita del server.

- `sp_replflush` esegue il "refresh" della cache degli articoli.

- `sp_repltrans` ritorna un set di risultati di tutte le transazioni nel transaction log del publication database, marcate per la replica ma non marcate come distribuite.

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.

4.2.12 Tabelle di sistema utilizzate nella Replica

Il processo di Replica sfrutta delle tabelle dal publication database, distribution database, dai database di destinazione e dal master database.

Tabelle del Publication database Il publication database usa le seguenti tabelle per il processo di replica:

- *sysarticles*: ha una riga per ogni articolo assegnato dal publication server.
- *sysobjects*: ha un campo settato dalla replica per ogni oggetto marcato per la replica. È settato quando un articolo è creato e inizialmente sottoscritto.
- *syspublications*: ha una riga per ogni pubblicazione assegnata dal publication server.
- *syssubscriptions*: associa gli ID di articoli pubblicati con gli ID di tutti i subscription server in attesa di ricevere dei dati.

Tabelle del Distribution database Il Distribution database contiene le seguenti tabelle dedicate al processo di replica:

- *MSjob_commands*: contiene un'entry per ogni comando associato a una transazione nella tabella *MSjobs*.
- *MSjob_subscriptions*: associa un subscriber ad un particolare articolo. (Questa è l'informazione dal lato subscriber memorizzata nel distribution database.)

- *MSjobs*: contiene le attuali transazioni. Vi è un'entry per ogni transazione memorizzata nel distribution database.
- *MSsubscriber_info*: contiene le informazioni usate da SQL Executive per passare i job.
- *MSsubscriber_job*: associa ogni subscriber con il comando che necessita ricevere.
- *MSsubscriber_status*: contiene le informazioni di stato delle transazioni batch inviate dai subscribers.

Tabelle dei database di destinazione Idatabase di destinazione mantengono le informazioni sulla replica nella tabella :

- *MSlast_job_info*: contiene l'ID dell'ultimo job (di un processo batch) applicato con successo. Un job è una transazione completa memorizzata nel distribution database.

  Inoltre, se presente, questa tabella contiene il nome della pubblicazione, il nome dell'articolo e la descrizione di un job abbia fermato temporaneamente la distribuzione aspettando la sincronizzazione manuale.

Tabelle del *master* database Tutti i server partecipanti alla replica mantnegono alcuni dati (quelli correlati al processo di replica) nella tabella *sysservers* nel *master* database; le due tabelle contenute in questo database sono:

- *sysservers*: usata sia dal subscription che dal publication database. La colonna "srvstatus" può assumere i valori RPC, PUBLISH, SUBSCRIBE, e DISTRIBUTE. Un publication server usa tutte e quattro le opzioni per registrare i server con cui partecipare alla replica. Un subscription server usa solo l'opzione RPC per registrare i publication server che esso autorizza ad inviare le pubblicazioni.

  Le opzioni possono essere visualizzate eseguendo la procedura `sp_helpserver`.
- *sysdatabases*: usata dai publication e subscription server. Settando il bit di stato è possibile permettere ad un databse di pubblicarsi. Sul subscription server, si setta un bit di stato permettendo ad un database si sottoscriversi, e setta l'ID del repl_publisher a DBO.

Nella figura 4.11 sono illustrate tutte le tabelle di sistema e i loro legami logici.

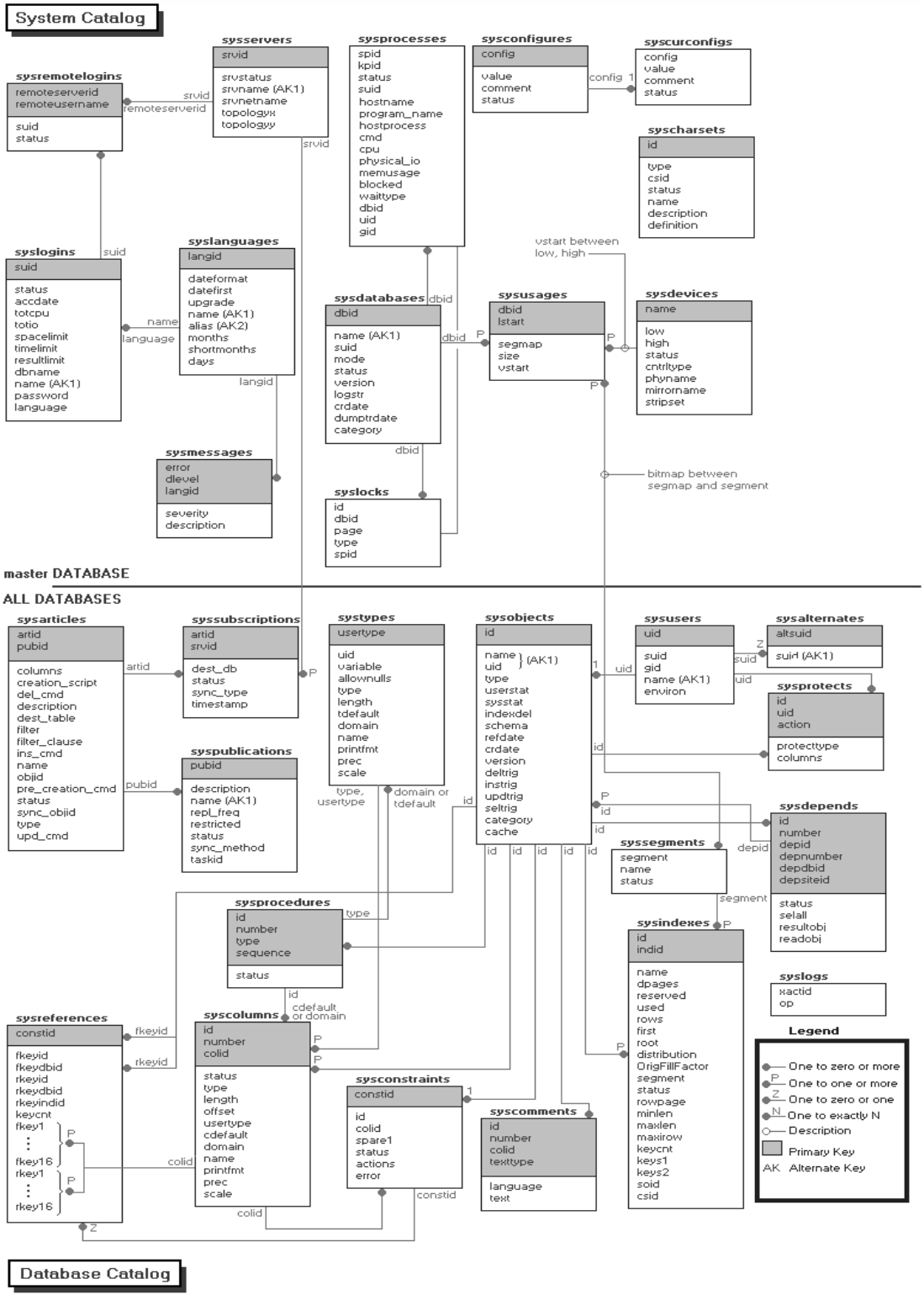


Figura 4.11: Diagramma delle tabelle di sistema

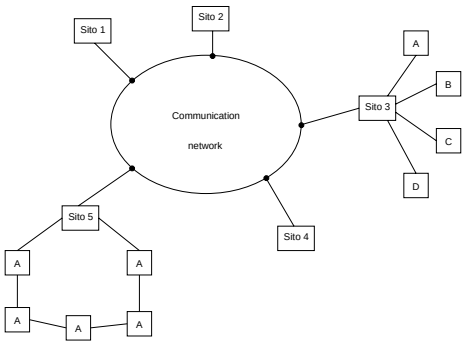


Figura 4.12: Esempio di sistema misto: distribuito e client/server

4.3 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.
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 4.12.

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 4.13 è rappresentato

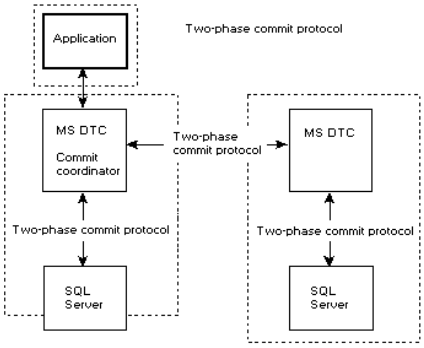


Figura 4.13: Implementazione del two-phase commit protocol nel MS-DTC

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.

Quando un resource manager lavora per conto di una transazione, essa gli viene assegnata chiamando il transaction manager. Mentre l'elaborazione della transazione procede, il transactione manager tiene traccia dei resource managers "arruolati" nella transazione.

Quando la transazione verrà completata con successo invocando il metodo della transazione *Commit*, essa chiamerà il MS DTC per eseguire il *commit* definitivo. MS DTC, attraverso il *2-phase commit protocol*, ottiene il commit da tutti i resource manager impiegati per la transazione: in una prima fase, MS DTC chiede a tutti i resource manager se sono "pronti" per il commit, se tutti rispondono positivamente, nella seconda fase MS DTC manda in broadcast il messaggio di commti a tutti.. Se una parte della transazione fallisce o se un resosurce manager non risponde alla richiesta di preparazione, MS DTC notifica a tutti i resource manager che la transazione è stata abortita.

4.3.1 Scenari d'esecuzione - configurazione locale

Come descritto prima, il "supervisore" di una transazione distribuita è detto, in terminologia MS DTC, **Transaction Manager**, mentre i partecipanti alla transazione, che interagiscono con i diversi database relazionali, risiedenti in questa situazione in una sola maccina NT, sono i **Resource Manager**.

Le transazioni, dal punto di vista dell'utente finale, sono unità di esecuzione ACID che raggiungono lo stato di commit o abort. Le applicazioni, i resource manager e i transaction manager cooperano per implementare le proprietà ACID. Ora tratteremo in maniera più approfondita il ruolo di ognuno di questi partecipanti.

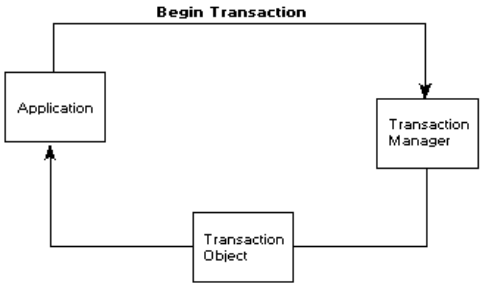


Figura 4.14: Ruolo delle applicazioni

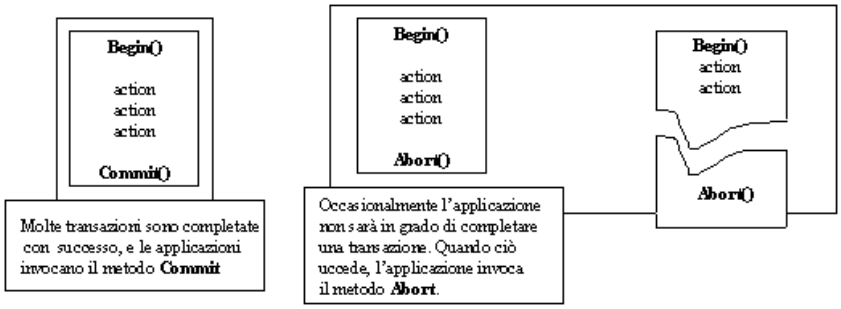


Figura 4.15: Ruolo delle applicazioni: *Commit* e *Abort*

Il ruolo delle applicazioni nelle transazioni

L'applicazione che sfrutta uno o più database , inizia una transazione ottenendo un "transaction object" dal Transaction Manager; tutte le fasi di lavoro successive di quest'applicazione sono associate a quest oggetto. I casi di successo e insuccesso dell'esecuzione di una transazione sono illustrati in Fig. 4.15.

Ruolo di un Resource Manager nelle transazioni

All'inizio quando un *Resource Manager* (al quale si interfacciano le applicazioni) viene installato in uno scenario d'esecuzione, contatta il Transaction Manager per dichiarare la sua disponibilità. Quindi aspetta per eventuali richieste d'esecuzione da parte di applicazioni. Quando arriva una richiesta, etichettata come un nuovo "transaction object", il Resource Manager si "arruola" (*enlists*) alla transazione; in questo modo il resource manager verrà invocato dal transaction manager quando una transazione raggiungerà il commit o l'abort.

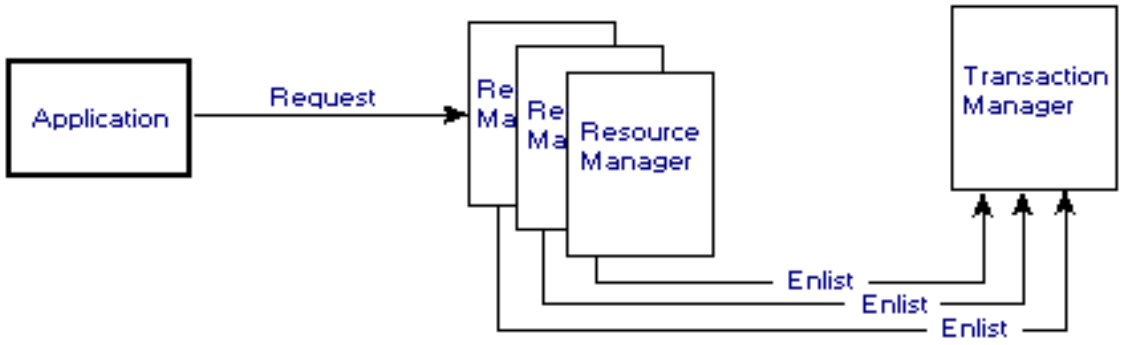


Figura 4.16: Ruolo di un Resource Manager

Il Resource manager esegue quindi la transazione: ad esempio un `insert`, `delete` o `update` di records in un database relazionale (vd. Fig.4.17). Esso acquisisce abbastanza informazioni da permettere di eseguire l'"undo" o il "redo" delle modifiche *locali* della transazione.

Quando l'applicazione chiama il **Commit**, il transaction manager inizia il "two-phase commit protocol": esso si informa se tutti i resource manager arruolati sono pronti per fare il commit della transazione.

Tipicamente un resource manager memorizza i nuovi e i vecchi dati in memoria permanente (nastro e/o disco) in modo da poter eseguire il recovery in caso di guasti; se esso non può essere pronto, informa il transaction manager che dichiarerà l'abort della transazione. Una volta pronto, un resource manager deve aspettare finché non ottiene una chiamata di commit o abort (per la transazione distribuita) dal transaction manager. Tipicamente, l'intera preprazione e il protocollo per il commit durano una frazione di secondo.

Se vi sono guasti al sistema , la notifica del commit o dell'abort non può arrivare per minuti o ore: il resource manager crea quindi dei "locks" sui dati modificati dalla transazione in modo da isolarli dalle modifiche di altre transazioni.

Ruolo del Transaction Manager

Il Transaction Manager è il gestore dei "transaction objects": li crea e gestisce la loro atomicità e persistenza. Le applicazioni lo interrogano per creare un oggetto della transazione invocando il metodo `Begin Transaction`.

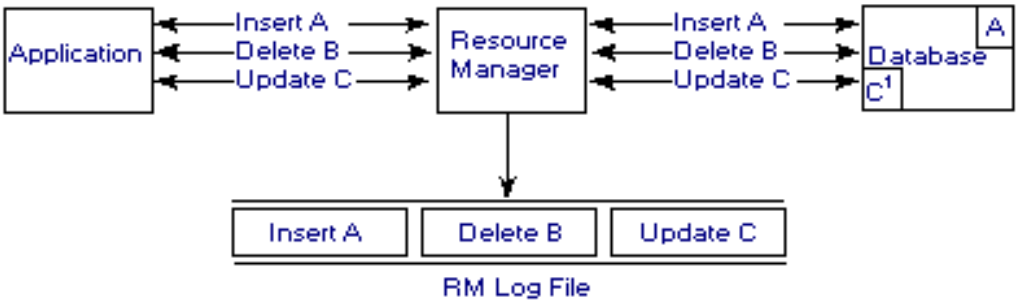


Figura 4.17: Ruolo di un Resource Manager: esempio di `insert`, `delete` e `update` di record

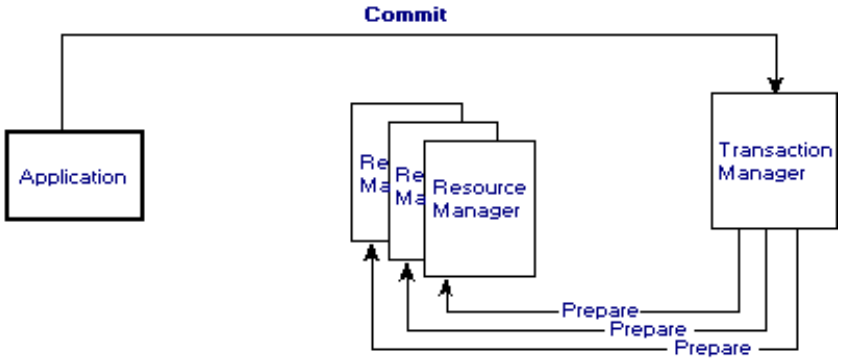


Figura 4.18: Ruolo di un Resource Manager: **Commit**

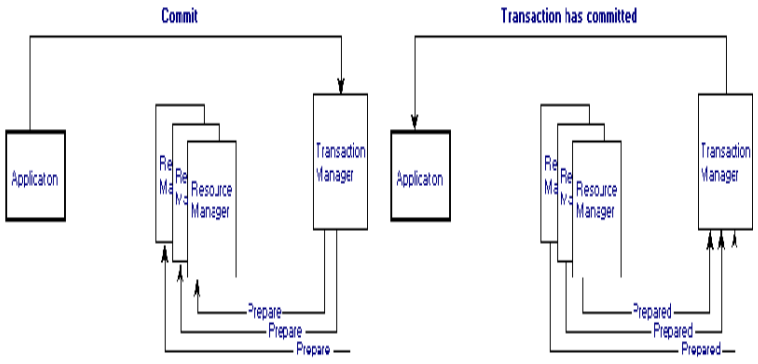


Figura 4.19: Ruolo di un Resource Manager: **Commit**

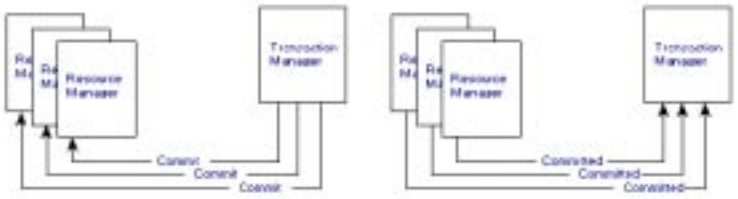


Figura 4.20: **Commit**: fasi d'interazione tra Resource e Transaction manager

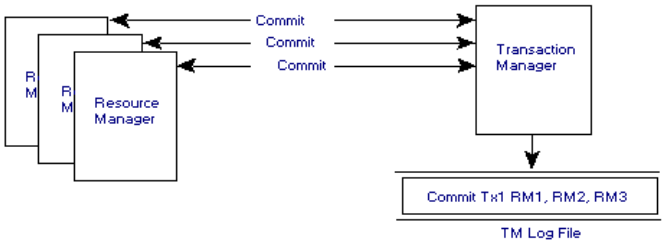


Figura 4.21: Transaction Manager: **Commit** e scrittura su log

Quando un resource manager partecipa inizialmente ad una transazione, invoca il transaction manager per essere assegnato ad essa; il Transaction Manager ne tiene traccia e quando è chiamato in causa per un commit o un abort, applica come già descritto, il two-phase commit protocol.

Il Transaction Manager tiene anche un file di *log* (un file sequenziale che registra gli eventi di una transazione) su disco: sono memorizzati l'inizio della transazione, i resource managers arruolati e i commit. Durante l'elaborazione normale, il Transaction Manager scrive solo sul log file; se esso fallisce, , ricostruisce lo stato più recente della transazione leggendo il log.

Il transaction manager è fornito anche di un'interfaccia per le operazioni di gestione delle transazioni e registra gli eventi più importanti sul log di sistema; questi eventi possono essere visualizzati usando l'"event viewer" di sistema (di Windows NT). Esso inoltre ha un'interfaccia grafica di gestione integrata in *Sql Enterprise Manager*, che permette di configurare il sistema, vedere le transazioni, e gli stati "incerti" di commit o abort.

4.3.2 Transazioni distribuite

Finora abbiamo trattato che tutte le applicazioni e i vari resource manager fossero su una singola macchina NT. MS DTC supporta anche le transazioni distribuite su una o più macchine. Ogni sistema ha un transaction manager locale; tutte le applicazioni e i resource manager dialogano con il loro transaction manager locale.

Quando una transazione all'inizio visita un nuovo sistema, i sistemi coinvolti nella richiesta stabiliscono una relazione. Il sistema che ha originato la richiesta informa il suo transaction manager locale che la transazione ha una relazione "uscente" (*outgoing relationship*) verso il transaction manager dell'altro sistema. In modo simile, il sistema che riceve la richiesta informa il suo transaction manager locale che la transazione ha una relazione "entrante" (*incoming relationship*) con il transaction manager del primo sistema.

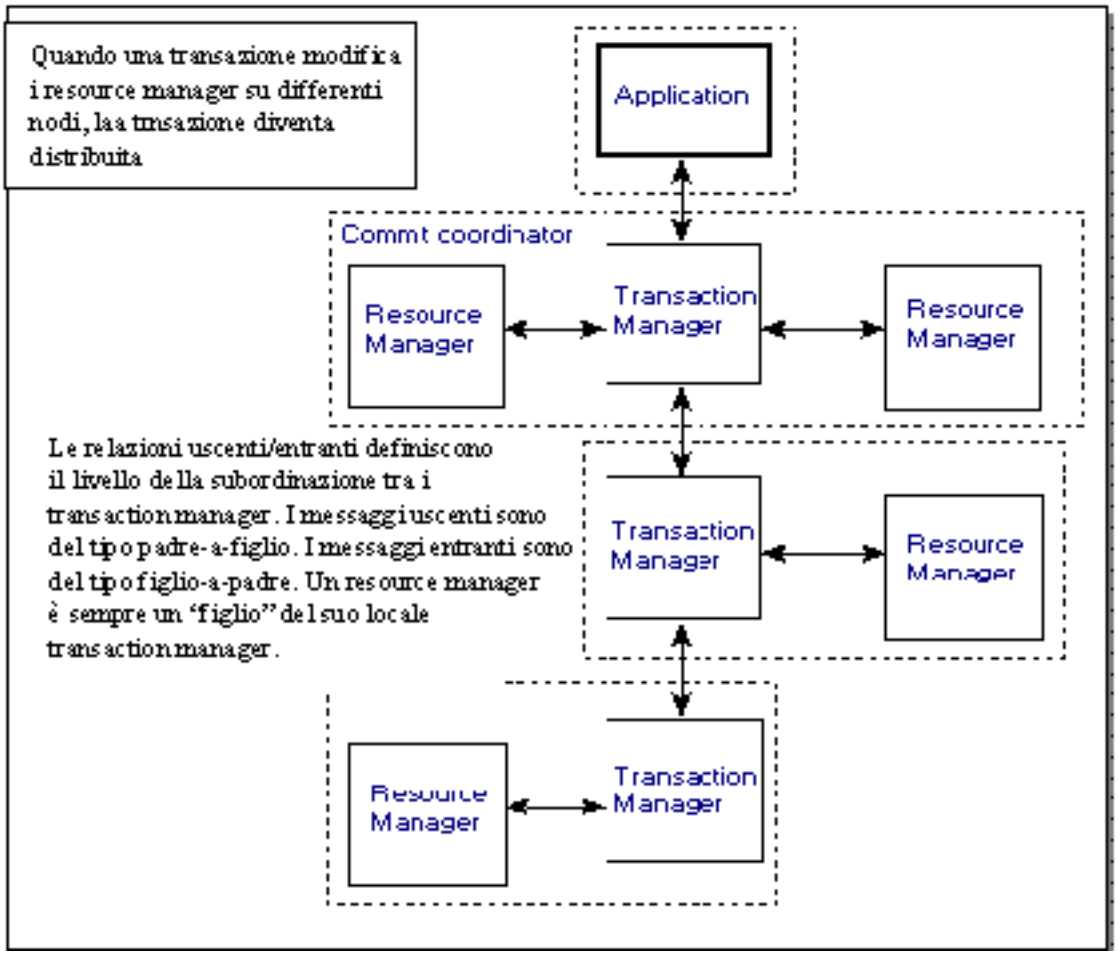


Figura 4.22: MS DTC in ambiente distribuito

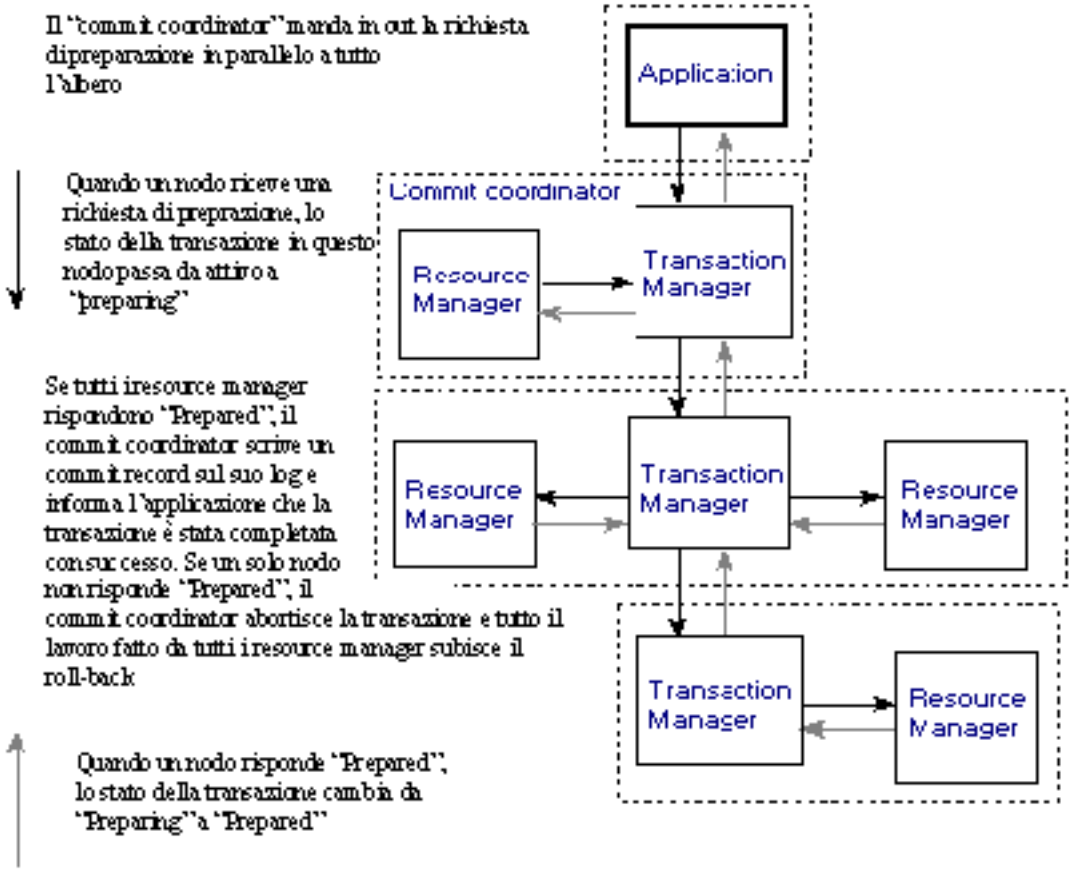


Figura 4.23: MS DTC in ambiente distribuito

In questo modo si crea un "albero" di relazioni chiamto *transaction's commit tree*; anche i resource manager arruolati fanno parte di questo albero ed hanno una relazione entrante/uscente con il loro locale transaction manager.

Quando una transazione distribuita raggiunge in commit o l'abort, il messaggio di preparazione (al commit o all'abort) fluisce all'esterno del commit tree. Ogni nodo dell'albero può unilateralmente abortire una transazione in ogni momento, prima di accettare la richiesta di preparazione inviata nella prima fase. Una volta che un nodo sia pronto, esso rimane in attesa finché il "coordinatore del commit" gli conferma il commit o l'abort della transazione. Il transaction manager radice dell'albero è il coordinatore globale per il commit; egli prende la decisione finale sul commit o l'abort di una transazione.

La figura 4.3.2 illustra il flusso di messaggi nella prima fase del "two-phase commit protocol" di una transazione distribuita. Se una macchina subisce un guasto e dopo riparte, il transaction manager locale determinerà l'esito di tutte le transazioni incerte e leggerà il suo file di log per determinare l'esito delle transazioni delle quali era il commit coordinator. Per le transazioni incombenti dagli altri sistemi, il transactoin manager legge il log file per determinare se precedentemente era stato notificato il loro esito; per quelle che rimangono in dubbio, interrogherà il loro manager per capire il loro esito. Il transaction manager risponde inoltre alle interrogazioni degli altri transaction manager sulle transazioni incerte inviate a loro.

Le transazioni possono rimanare incerte per un lungo periodo: le risorse da esse modificate rimangono nello stato di "lock" e non disponibili per altri transaction manager. Il progammatore può sfruttare un'interfaccia grafica per il commit coordinator, per risolvere quelle transazioni che sono in uno stato non determinato e forzarle a uno stato di commit o abort; quando il sistema guasto si riconnette, MS DTC rivela le azioni eseguite manualmente dall'operatore e segnala quelle inconsistenti.

4.3.3 Recovery dell'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.

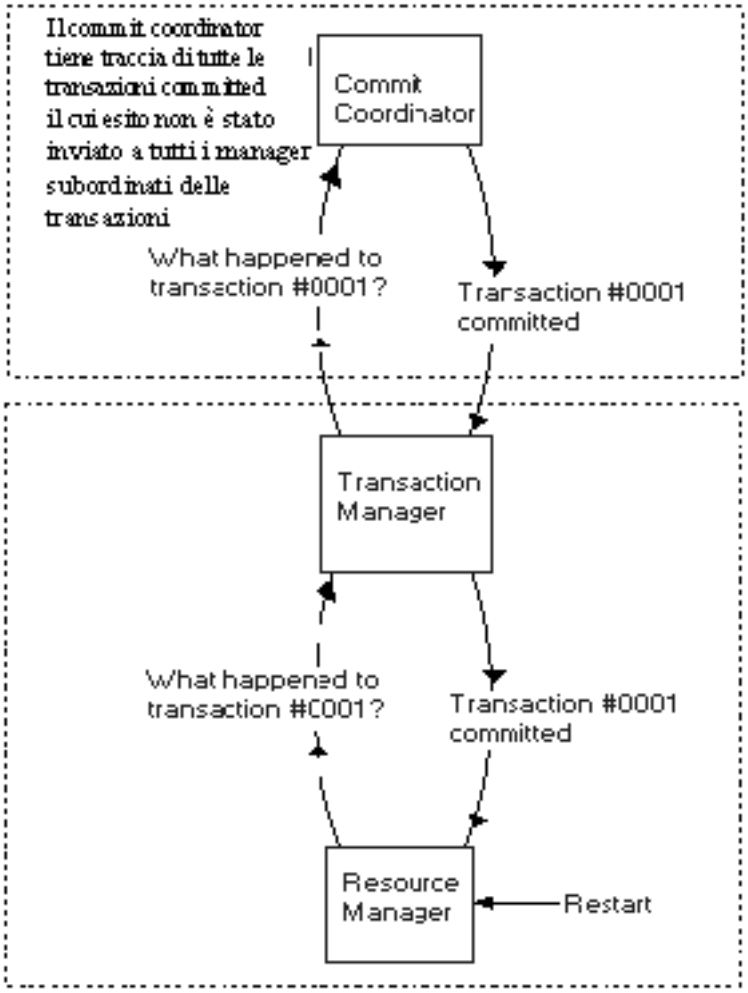


Figura 4.24: MS DTC in ambiente distribuito: Restart del sistema

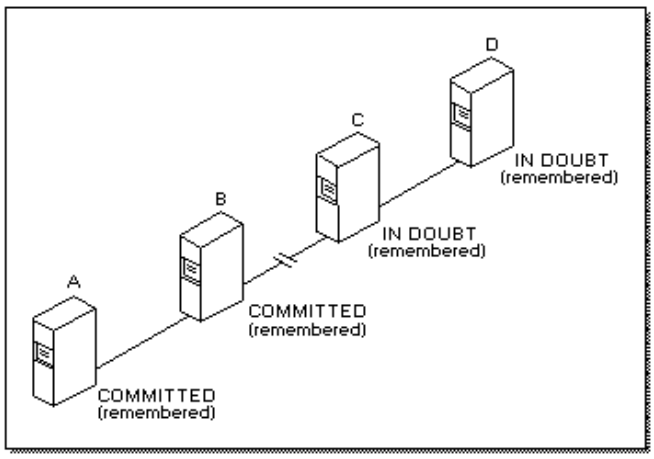


Figura 4.25: Recovery nel MS-DTC

4.4 Integrità dei dati e controllo della concorrenza in SQL Server 6.5

I due punti fondamentali nella gestione dei dati in un DDBMS, sono:

1. assicurare che le transazioni concorrenti possano produrre gli stessi effetti come se esse fossero isolate;

2. coordinare le modifiche ai dati in un modo consistente e compatibile con la gestione della sicurezza nell'organizzazione.

4.4.1 High Speed Locking

Il processo per ottenere un *lock* è in media dell'ordine di qualche microsecondo e le informazioni relative vengono memorizzate in una tabella residente in memoria. Vengono impiegati livelli multipli di locking (a seconda del grado di esclusione richiesto) e sono acquisiti i lock necessari ogni volta che uno statement SQL viene eseguito. SQL Server usa :

- *Shared locks* per operazioni read-only. Se viene applicato un shared lock a una pagina o pagine di dati, una seconda transazione può acquisire un altro shared lock sebbene la transazione non sia completata. Di default i shared locks vengono rilasciati non appena le pagine di dati non sono più necessarie.

- *Exclusive locks* per operazioni di scrittura - `UPDATE`, `INSERT`, `DELETE`. Quando è settato un exclusive lock, nessun'altra transazione può acquisire un lock finché SQL Server non rilascia il lock originale alla fine della transazione.

- *Update locks* quando una transazione vuole modificare una pagina prima di un'operazione di write. Questi tipi di locks riducono i conflitti tra "scrittori" multipli.

Di default, i cambiamenti generati dagli statement SQL diventano visibili alle altre transazioni solo dopo che questi cambiamenti hanno raggiunto lo stato di "committed".

Consistenza dell'operazione di Read

SQL Server, per la consistenza delle letture e per il controllo della concorrenza fornisce livelli configurabili d'isolamento delle transazioni. Questi livelli (vedi Tab. 4.1) determinano come le transazioni interagiscano a vicenda e sono configurabili sia globalmente per sessione sia individualmente, per ogni statement SQL.

4.4.2 Automatica prevenzione, individuazione e correzione dei deadlock

Uno stato di *deadlock* avviene quando due o più transazioni acquisiscono un lock su oggetti separati: ognuna vuole acquisire un lock su un oggetto dell'altra transazione; la prima aspetta il rilascio del lock della seconda, che a sua volta non lo lascierà finché il lock sull'oggetto della prima transazione non sarà rilasciato. Il risultato è che entrambe le transazioni sono in uno stato d'attesa indefinito.

Microsoft SQL Server evita automaticamente i tipi più comuni di deadlock eliminando o diminuendo molto l'uso di lock estesi durante gli update, lo "split" delle pagine, e il mantenimento degli indici. Se avviene uno stato di deadlock, il DBMS lo elimina automaticamente facendo il roll-back della prima transazione che "romperà" il deadlock per tutte le altre transazioni in attesa (se ce ne sono più di una). In questo modo le altre transazioni ottengono il lock e sono libere di continuare la loro esecuzione.

SQL Server previene automaticamente anche il problema della *lock starvation* (stato determinato dalla continua e "monopolistica" lettura da parte

di una transazione di pagine o tabelle, forzando l'attesa indefinita delle altre transazioni), coordinando gli accessi alle pagine e alle tabelle, con un protocollo simile al *first- come, first-served*.

4.4.3 Locking a livello di riga e di pagina

SQL Server utilizza prevalentemente il locking a livello di pagina, in modo da bilanciare l'"overhead" dell'individuazione dei deadlock e il throughput delle transazioni. Nella maggioranza delle situazioni aziendali (molte transazioni OLTP), il locking a livello di pagina è prestazionalmente più efficiente del locking a livello di riga, susciettibile di eccessive chiamate al lock manager.

Il vantaggio maggiore di una strategia con locking a livello di riga è una più elevata ed efficiente gestione della concorrenza. I difetti invece di questa strategia sono:

- **Eccessivo overhead del Lock Manager**. Un numero maggiore di lock deve essere gestito ed acquisito per un dato set di transazioni.
- **Incremento dell'overhead dell'I/O**. Il sistema può memorizzare le informazioni sullo stato dei lock nelle stesse righe dati, quindi ogni volta che un lock è acquisito un'operazione di I/O deve essere eseguita alla pagina in cui la riga è memorizzata.

4.4.4 Controllo della concorrenza

SQL Server impiega diverse tecniche per fornire un alto livello di concorrenza; le più importanti sono:

- Controllo della concorrenza "ottimistico";
- Locking ad alta granularità;
- Indici clustered .

Controllo ottimistico della concorrenza

Il controllo ottimistico della concorrenza parte dall'assunzione che le altre transazioni (usualmente) non modificheranno i dati letti. In questo modo non si hanno dei lock nella corrente riga ed è supportato un alto grado di concorrenza. Le altre transazioni possono liberamente leggere o scrivere i dati; se comunque una transazione modifica la riga, ciò viene rilevato, l'update o la read vengono rifiutati, e automaticamente avviene il refresh del buffer della riga nel client con le nuove informazioni.

Locking ad alta granularità

Le modalità di locking cambiano a seconda delle situazioni: il locking esteso è ridotto al minimo durante la creazione degli indici o l'allocazione delle pagine. Il locking è interamente eliminato per i databse che sono in modalità "single-user" o definiti come read-only.

Indici clustered

Un problema significativo per la gestione dei database è il cosidetto "hot spotting": un "hot spot" avviene quando gli `INSERT` sono concentrati tutti sull'ultima pagina di una tabella, causando un collo di bottiglia. Questo succede normalmente quando un dato non è ordinato. Tramite gli indici clustered si previene tutto questo: questi indici clustered ordinano i dati nell'ordine dell'indice, incorporando le pagine dati in una struttura B-tree dell'indice stesso. I benefici sono molti: riduzione dei costi di I/O e incremento delle prestazioni; inoltre i dati desiderati possono essere acceduti in un unico accesso a disco anche quando gli statement SQL ne prevedano di più. Gli "hot spots" sono evitati grazie al fatto che le inserzioni di riga sono automaticamente distribuite nella tabella. Per ridurre lo splitting delle pagine durante l'inserimento di nuove righe, SQL Server fornisce al DBA la possibilità di settare un parametro apposito (*fill factor*) per gli indici clustered, riservando e gestendo spazi in ogni pagina per nuove righe.

4.4.5 Tipi d'integrità dei dati

Il rafforzamento dell'integrità dei dati riguarda il mantenimento della loro consistenza e correttezza, validando i contenuti di colonne individuali, verificando i valori di colonna rispetto a un altro, validando i dati di una tabella comparandola a un'altra e controllando che il database sia modificato accuratamente e con successo dopo ogni transazione. L'integrità dei dati ricade in cinque categorie :

1. L'*Entity integrity* preserva l'unicità delle entità (righe) di una tabella. L'entity integrity è usualmente rafforzata attraverso primary keys di una tabella o attraverso indici unici.

2. *Domain integrity* si riferisce a validi inserimenti in una data colonna. Essa è assicurata da restrizioni dei tipi di dato, formati, o range di possibili valori per una colonna.

3. *Referential integrity* mantiene la consistenza tra tabelle multiple di un database. In SQL Server ciò può essere garantito dichiarativamente, attraverso vincoli nelle tabelle, o attraverso la programmazione di triggers e stored procedures.

4. *User-defined integrity* permette di rafforzare attivamente i ruoli aziendali dell'organizzazione: SQL Server permette ciò tramite la programmazione dell'architettura del server.

5. *Distribuited data integrity* mantiene la consistenza dei dati replicati o partizionati in multipli database o multipli server.

La tabella 4.2 elenca le opzioni d'integrità dei dati in SQL Server.

4.5 Caratteristiche di rete di un ambiente SQL Server

L'ambiente di elaborazione delle transazioni on-line (OLTP) tipicamente genera un grande numero di piccoli pacchetti, e l'ammontare dei dati da trasmettere tra il server e i client è relativamnete modesto. Molte applicazioni utilizzano pesantemente stored procedures che, non solo riducono l'elaborazione del server, ma anche il traffico di rete. Le stored procedures sono invocate passando il nome della procedura e i suoi parametri nel server; anche se qualche volta esse possono ritornare in out un elevata quantità di dati, tipicamente una stored procedure da in uscita un set di dati molto ridotto.

Comunque, sia usando le stored procedures sia non usandole, il volume maggiore di OLTP avviene ancora nel server, e raramente i livelli fisici di rete (interfacce di rete e cablaggi) hanno una significativa influenza sulle prestazioni globali.

L'ambiente di supporto alle decisioni e di elaborazione batch utilizza la rete in un modo leggermente diverso. L'utilizzo della rete aumenta, poiché la quantità di dati da trasmettere tra server e client è significativamente più grande, e permette che il software e l'hardware di rete formino pacchetti più grandi anche se sono trasmessi in un periodo di tempo più breve.

Nelle prossime sezioni verranno analizzate le differenze di prestazioni in base alla dimensione dei pacchetti e ai tipi di protocolli di rete.

4.5.1 Protocolli di rete e dimensione dei pacchetti

SQL Server comunica con i client attraverso le cosidette *Tabular Data Stream* (TDS). I pacchetti TDS hanno una dimensione di default di 512 bytes. Le

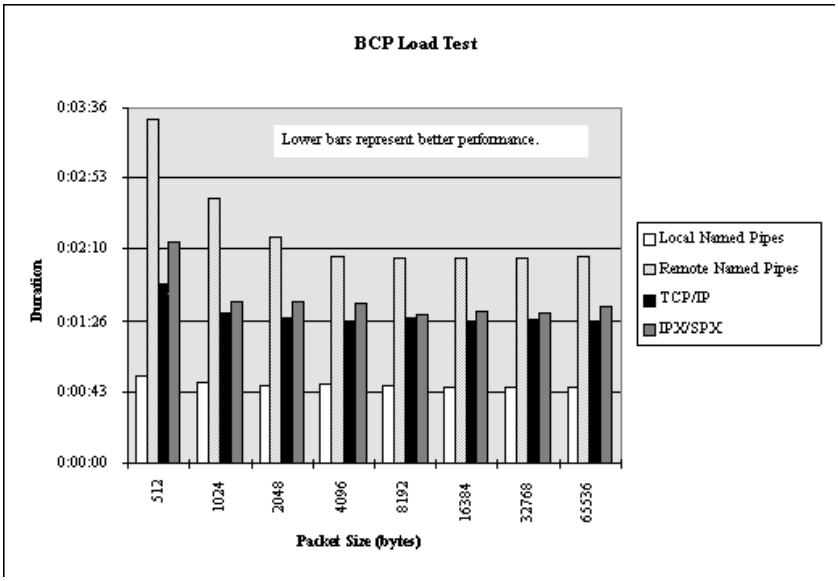


Figura 4.26: Prestazioni SQL Server; fonte Compaq Laboratories

applicazioni possono gestire la dimensione dei pacchetti usando la chiamata alla DB-Library `DBSETLPACKET()`; Con le utilities BCP e ISQL si può variare la dimensione usando il parametro `[-a packetsize]`.

Per trasferimenti tra client e server più piccoli, come quelli per l'esecuzione di stored procedures, la dimensione dei pacchetti condiziona molto le prestazioni. Comunque, per un trasferimento più grande, come il caricamento e lo scaricamento di BCP o l'esecuzione di statement SQL "data intensive", è preferibile usare una dimensione di pacchetto più grande per avere prestazioni migliori.

4.5.2 Protocolli di rete: Named Pipes, NWLink IPX/SPX e TCP/IP

La Figura 4.26 mostra le differenze di prestazioni in base al protocollo di rete per il "BCP load".

Dalla figura si può notare come una più grande dimensione dei pacchetti, aumenti le prestazioni e si posson scoprire le notevoli differenze tra un protocollo e l'altro.

Per un ambiente che invece sfrutta pesantemente le stored procedures, con un minimo trasferimento dati, né i tipi di protocollo, né la dimensione dei pacchetti, impatterà significativamente sulle prestazioni.

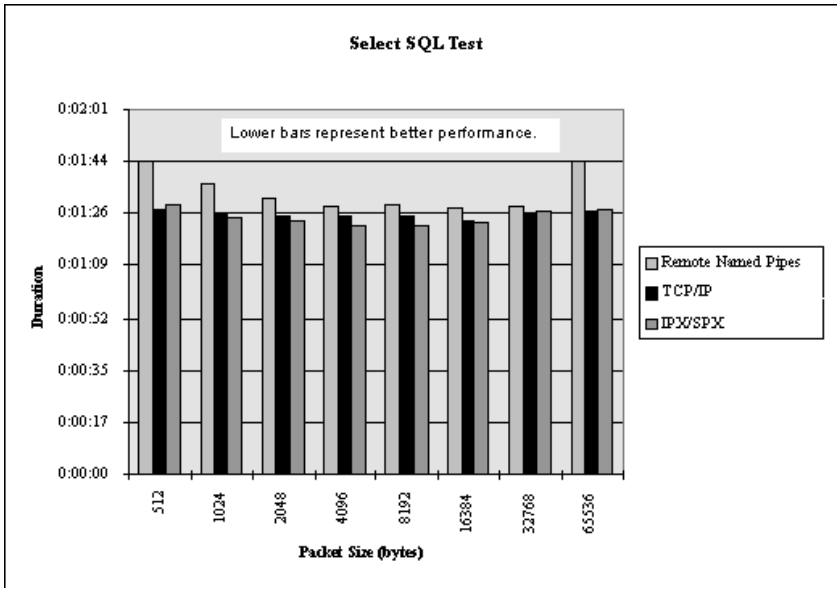


Figura 4.27: Prestazioni SQL Server; fonte Compaq Laboratories

Nel test di Fig. 4.26 è stato misurato il tempo il caricamento di 250.000 righe di tabelle; alcune conclusioni sono evidenti: 1) pacchetti di dimensioni maggiori aumentano le prestazioni, 2) la connessione locale a Named Pipes è più veloce, e 3) il protocollo TCP/IP sembra fornire prestazioni globali di rete migliori.

La Figura 4.27 rappresenta le prestazioni con pacchetti di varie dimensioni e vari protocolli di rete durante una selezione di 10.000 righe da tabelle con un semplice statement SQL: `select * from <table>`. Questo test rappresenta query che dovrebbero gestire grandi set di dati che il server deve fornire in uscita.

In base ai risultati in figura, si possono fare le seguenti osservazioni: 1) i pacchetti più grandi migliorano le prestazioni, anche se in maniera meno evidente, 2) le differenze di prestazioni usando diversi protocolli di rete sono meno evidenti, e 3) i protocolli TCP/IP e IPX/SPX sembrano produrre un miglioramento meno significativo del Named Pipes.

Livello d'isolamento	**Descrizione**
READ UNCOMMITTED	Indica a SQL Server di non usare i shared locks e non rispettare gli exclusive locks durante le letture. A questo livello le applicazioni devono leggere valori entro transazioni "uncommitted" che potrebbero subire il "roll-back" in futuro ("*dirty reads*"), e quindi dovrebbero essere usate solo da quelle transazioni che presentano spesso tale comportamento.
READ COMMITTED	Indica a SQL Server di usare i shared locks mentre vengono letti i dati (comportamento di default). A questo livello, le transazioni non possono avere delle "dirty reads".
READ REPEATABLE READ SERIALIZABLE	Garantiscono che i dati ripetutamente letti da una transazione abbiano lo stesso valore e che le "dirty reads", le letture non ripetibili, e valori indefiniti (valori che subiscono il roll-back dopo essere stati letti) non possano avvenire.

Tabella 4.1: READ Consistency

Tipo d'integrità	**Prime opzioni**	**Opzioni di SQL Server**
Entità	Indici unici	PRIMARY KEY UNIQUE constraint Proprietà IDENTITY Indici unici
Dominio	Tipi di dato, Default, Ruoli	DEFAULT constraint FOREIGN KEY constraint CHECK constraint NOT NULL constraint Tipi di dato, Defaults, Ruoli
Referenziale	Triggers	FOREIGN KEY constraint CHECK constraint Triggers
User-defined	Ruoli, Triggers e Stored Procedures	Tutti i livelli di vincoli di colonna e tabella in `CREATE TABLE WITH CHECK OPTION on VIEWS`, Ruoli, Triggers e Stored Procedures

Tabella 4.2: Integrità dei dati

Capitolo 5

Progetto di distribuzione di Action Workflow

5.1 Tipologie di distribuzione di un WFMS

I sistemi di workflow, i quali gestiscono e regolano il passaggio tra più partecipanti di documenti, dati e processi, sono per natura distribuiti: le applicazioni esterne che eseguono i task di workflow sono spesso geograficamente decentrate e inoltre anche i WFMS possono essere essi stessi distribuiti. La caratteristica comune di questi tipi di sistemi è la *distribuzione delle funzioni*: componenti di workflow diversi che eseguono varie funzioni di workflow, come la definizione, l'esecuzione, il monitoraggio dei processi, e l'assegnamento delle risorse, sono localizzati in siti diversi. I componenti dei WFMS interagiscono a vicenda tramite scambio di messaggi o RPC. Un'altra modalità di distribuzione è l'esecuzione di funzioni di workflow da parte di multipli e *funzionalmente equivalenti* componenti di WFMS che *condividono* memorie comuni e quindi instanze di processo comuni. Per esempio, l'esecuzione di un processo di workflow può essere eseguito collettivamente da diversi *workflow engine* che condividono gli stessi dati per la definizione dei processi e gli stati d'esecuzione. Tali sistemi forniscono una migliore scalabilità e resilienza ai guasti del sistema, ma sono ancora vulnerabili a errori nella memorizzazione dei dati.

La difficoltà maggiore nella seconda classe di WFMS, è la coordinazione e il sincronismo nell'esecuzione collettiva di un processo da parte di "motori di workflow" multipli e indipendenti (che non condividono i dati) . In tali sistemi, ogni WFMS è un completo sistema con il proprio engine e dati, e non c'è un server centrale che mantiene tutte le informazioni sull'esecuzione di tale processo. Questi sistemi sono migliori sia in sicurezza che presta-

zioni: le attività di workflow possono essere eseguite dai WFMS che sono più vicine alle corrispondenti applicazioni esterne (risparmiando i costi di comunicazione tra WFMS e applicazioni) e i WFMS accedono localmente sia alla definizione dei processi che agli stati d'esecuzione (riducendo i costi di comunicazione tra WFMS e dispositivi di memorizzazione dei dati). Sono sistemi anche più affidabili perché un problema a uno o più WFMS (incluso il corrispondente dispositivo di memorizzazione) non fermano l'esecuzione del processo (a meno di mutue dipendenze tra i processi stessi).

Il sistema di workflow da noi utilizzato (Action Workflow) è progettato per architetture client-server: su un unico server centralizzato è installato il *workflow engine*, sul quale si installano i processi (disegnati tramite *mappe* poi compilate dal Process Builder); gli utenti si collegano al server dai soli client, tramite un'interfaccia grafica da progettare ad hoc per ogni distinta applicazione di workflow, la quale interrogherà il server centrale (e quindi il workflow engine) sulle attività e i compiti da svolgere.
Questo sistema presenta due rilevanti carenze :

- limiti imposti dall'architettura client/server, quali la stretta dipendenza dal server centrale, i guasti e la necessitá di una rete di comunicazione molto efficiente;
- mancanza di una gestione efficiente dei dati;

Le soluzioni da noi proposte a questi limiti sono rispettivamente:

- ditribuzione del sistema di workflow su uno o più server;
- integrazione ad Action Workflow del sistema di replica di dati di SQL Server, in particolare della tipologia di replica definita nel modello Dynamic Ownership sviluppato in questa università.

Si analizza ora in dettaglio l'approccio al primo problema.

5.2 Action Workflow: da sistema client/server a sistema distribuito

Come accennato prima, il sistema Action Workflow é basato su architettura client/server (vedi fig. 5.1): l'utente si collega dal client periferico (che può anche essere un semplice PC dove siano installate l'applicazione e le API) al server centrale tramite un'interfaccia grafica; ogni azione eseguita da applicazione, è trasformata dalle API in modifiche alle tabelle di *aws*.

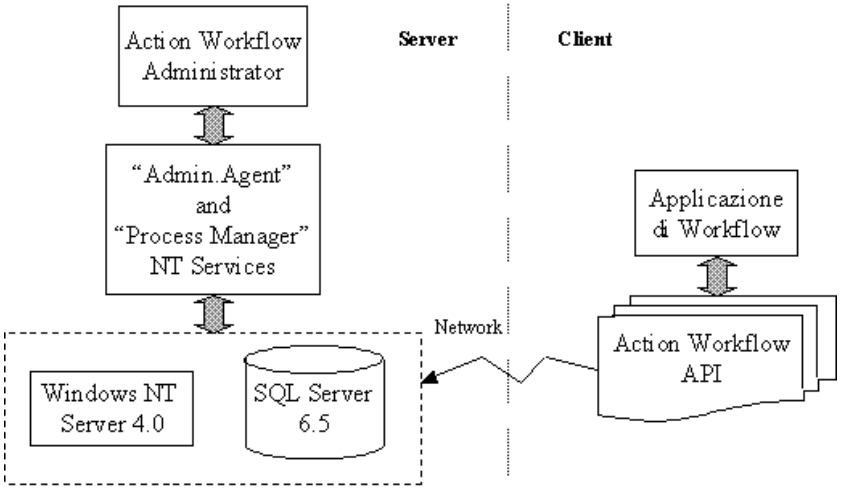


Figura 5.1: Action Workflow: configurazione Client/Server

Dal lato server, sono residenti e attivi in memoria due servizi di NT, l'*Administrator Agent* e il *Process Manager*, che costituiscono il *workflow engine*; esso è il "coordinatore" dell'intero sistema e si occupa ad esempio di eseguire il *Commit* delle transazioni e di compilare gli script della mappa.

I processi (le mappe compilate nel Process Builder) vengono installati nel server tramite il ActionWorkflow Administrator, il quale memorizza le definizioni, lo stato, le transizioni tra attività e tutte le informazioni relative alla specifica applicazione di workflow progettata, nel database *aws* residente solo nel server (é un database di SQL Server).

La prima fase del processo di distribuzione consiste quindi nel replicare questo database su piú server, in modo da avere copie identiche e sincronizzate di *aws* in piú siti.

Una volta replicato il database, l'ActionWorkflow Administrator verrà installato su tutti i server (viene così generato il database *aws* in tutti i siti), mentre il *business process* (la mappa compilata) dovrà essere installata solo su un server; sará poi il sistema di replica di SQL Server a distribuirlo a tutti gli altri siti: in questo modo si avranno due copie perfettamente sincornizzate della stessa istanza. I dettagli del *setup* iniziale verranno comunque descritti in seguito.

Prima di descrivere le fasi di configurazione della replica di *aws*, possiamo osservare che l'esecuzione dei processi risulta così affidabile, in quanto:

- la coordinazione tra i processi nei diversi server è garantita dal sistema

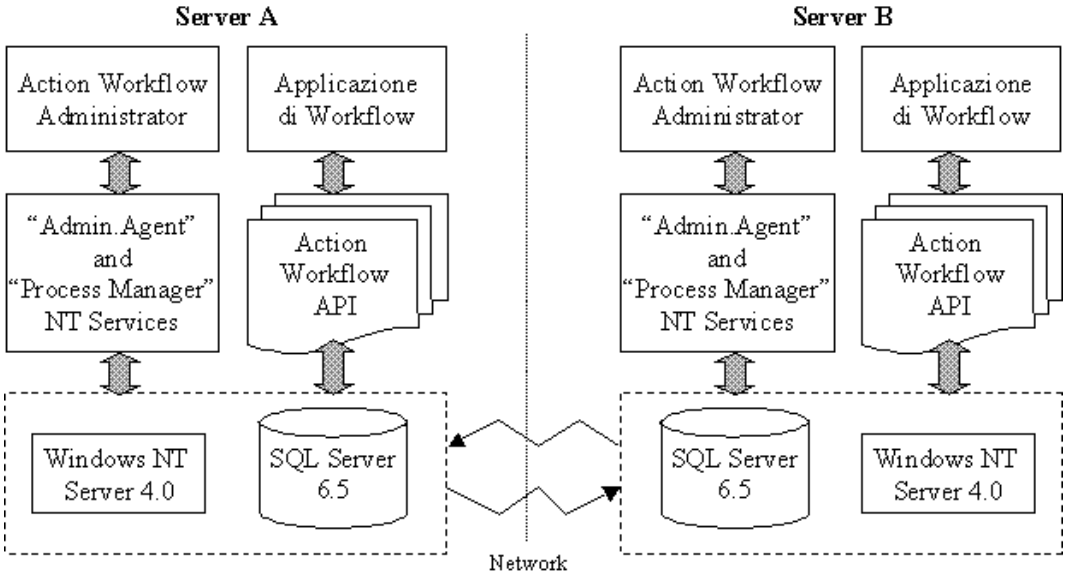


Figura 5.2: Action Workflow in configurazione distribuita

di replica di SQL Server: sono residenti nei vari server copie identiche e soprattutto sincronizzate di definizione e istanze di processo;

- le applicazioni client accedono solo localmente sia alla definizione dei processi che agli stati d'esecuzione, risparmiando sui tempi di comunicazione;
- si evitano blocchi del sistema dovute a dipendenze tra processi, tipiche di quella classe di sistemi di workflow distribuiti visti inizialmente prima, cioè dove possono esistere su siti diversi istanze di processo diverse, dipendenti le une dalle altre (in questa configurazione, infatti, l'istanza di processo è la stessa);
- il meccanismo di replica di SQL Server assicura un consistente sincronismo tra i processi residenti su siti diversi: se un certo server A (o anche solo il workflow engine in esso residente) dovesse per un certo periodo essere inutilizzabile, le modifiche apportate al processo sul server B verrebbero apportate in seguito anche al primo sito, grazie ai processi di ditribuzione di SQL Server (il *Log Reader* e *Distribution Process*, i quali tengono memoria delle transazioni eseguite su /em aws e le ridistribuiscono al Server A non appena questo ritorna utilizzabile.

Lo svantaggio è l'alto costo della comunicazione all'atto della sincronizzazione, poichè il "package" di informazioni potrebbe essere molto ampio (si

replicano bidirezionalmente circa 130 tabelle). Inoltre da un punto di vista funzionale, non risulta molto flessibile: non si riescono a gestire situazioni dove delle attività possano essere pre-assegnate a determinati siti, dove ad esempio risiedono localmete specifiche risorse necessarie solo a determinati processi.

5.2.1 Replica del database *aws*

Nel seguente paragrafo verrà illustrato come è stato affrontato il problema della distribuzione di Action Workflow nella tesi precedente, dalla replica undirezionale alla progettazione della replica bidirezionale del database *aws*.

Il primo problema da risolvere è stato il fatto che il database *aws* non è fornito di chiavi primarie, mentre SQL Server permette la replica solo di articoli formati di tabelle aventi chiavi primarie (definite con il vincolo `CONSTRAINT PRIMARY KEY`). In *aws* l'integritá dei dati viene garantita attraverso indici `UNIQUE` o contatori per generare degli ID numerici incrementati ad ogni inserimento (in sostituzione delle chiavi primarie) e tramite opportuni trigger per gestire le relazioni tra le tabelle (non vi sono infatti `FOREIGN KEY`).

Replica unidirezionale. A un primo tentativo di generare delle *superchiavi* (per permettere la replica), cioè chiavi che comprendano tutti i campi della tabella in questione, esclusi i campi che ammettono i *NULL*, il sistema di workflow si bloccava dando errore di violazione di chiave. Da ciò la scelta di procedere in altro modo: si sono suddivise le tabelle di *aws* in due gruppi da trattare diversamente a seconda delle caratteristiche; nel primo gruppo abbiamo:

- tutte le tabelle che hanno indice unique composto di soli campi che non ammettono i "NULL";
- tutte le tabelle prive di indici unique e composte di soli campi che non ammettono i "NULL".

A queste tabelle si aggiunge una chiave primaria o sui campi dell'indice o su tutti i campi. Da notare che la chiave primaria deve essere di tipo `UNCLUSTERED` in quanto gli indici sono di tipo `CLUSTERED` e ogni tabella ammette solo un indice di questo tipo.

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: 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 5.3.

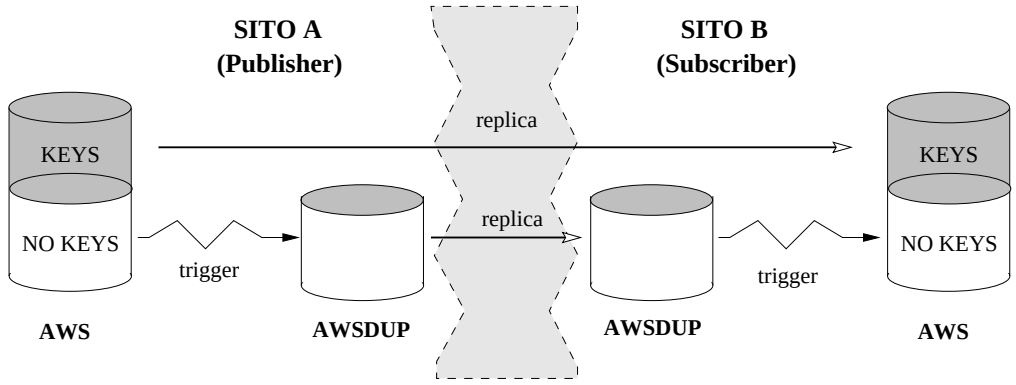


Figura 5.3: Replica unidirezionale di *aws*

Fasi successive: replica bidirezionale La replica bidirezionale ha comportato i seguenti problemi: 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 portava a generare degli errori.

Per risolvere il problema si sono modificati i trigger in modo da accorgersi dell'origine dei dati che li hanno fatti scattare. In particolare, i trigger su *aws* distinguono 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 5.4.

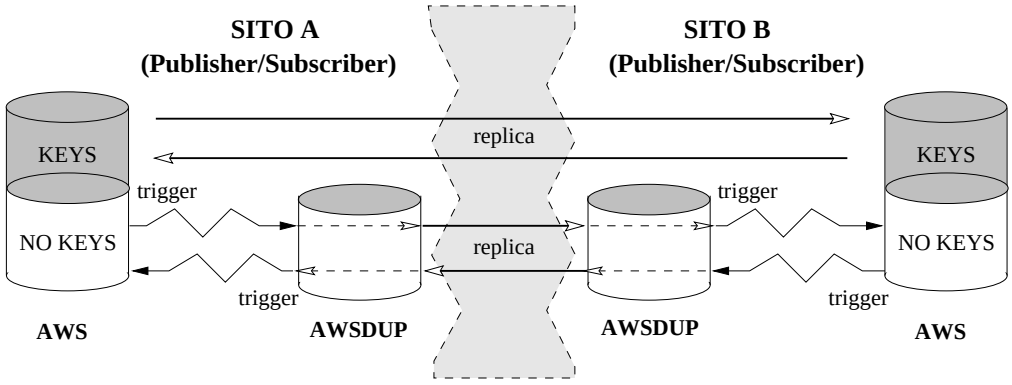


Figura 5.4: Replica bidirezionale di *aws*

Il metodo usato per fare in modo che i trigger distinguessero una modifica locale da una mofica da replica, è stato quello di utilizzare la variabile globale di SQL Server `@@NESTLEVEL`, la quale tiene conto nel numero di transazioni innestate partendo dal default "0" e incrementando il valore ogni volta che una nuova transazione inizia all'interno di un'altra. I trigger sui due *aws* erano quindi di questo tipo:

```
begin
  if @@NESTLEVEL = 1
     begin
      <sposta i dati su awsdup>
     end
end
```

Su *awsdup* analogamente:

```
begin
  if @@NESTLEVEL = 1
     begin
      <sposta i dati su aws>
     end
end
```

Supponiamo ora che avvenga un inserimento di un dato al sito A nel database *aws* (vedi fig. 5.5. 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". I dati vengono spostati su *awsdup* dal trigger per essere poi trasferiti sull'altro sito grazie alla replica di SQL Server.

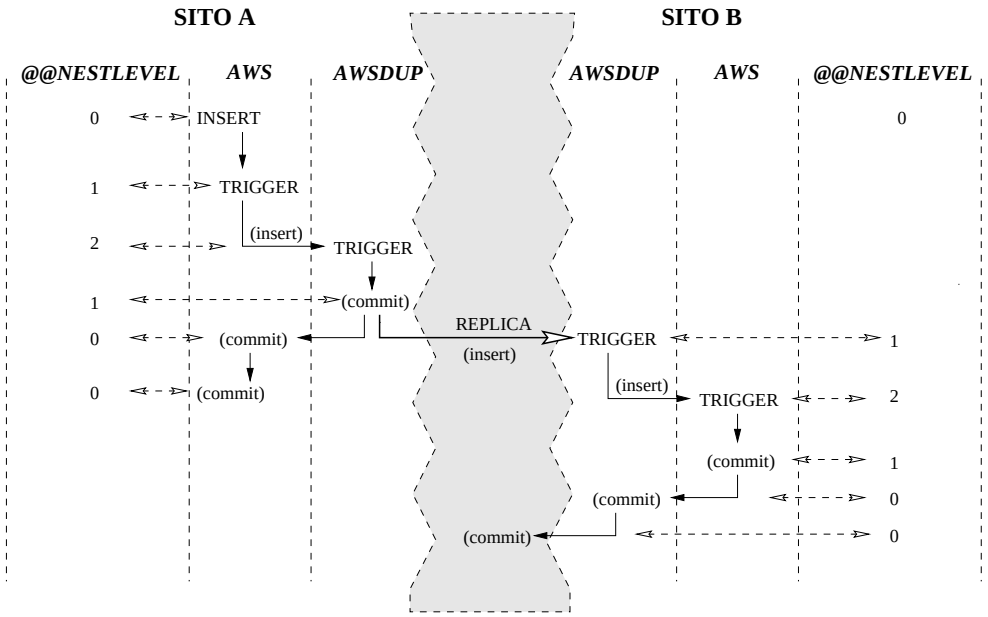


Figura 5.5: Il funzionamento dei trigger per la replica di *aws*

Su *awsdup*, l'insert dovuto ai dati che arrivano da *aws*, fa scattare il trigger, che però, iniziando all'interno del precedente, incrementa "@@NESTLEVEL" ponendola a "2". Il controllo allora fallisce, e il trigger su *awsdup* non esegue 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. 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, 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* nel sito B sincronizzando i due database.
Per quanto riguarda 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 (i campi di `SURR_KEY`), 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.

A questo punto si conclude il lavoro svolto nella tesi precedente, e nelle seguenti sezioni verranno descritte soluzioni diverse riguardo alla replica di *aws*, alla luce dei problemi sorti all'atto dell'installazione ed esecuzione del sistema.

5.2.2 Problemi della replica bidirezionale

I problemi nati all'atto dell'installazione sono stati i segenti:

- SQL Server non permette la replica bidirezionale di una tabella a meno di avere delle partizioni orizzontali, come ad esempio se da un lato replico le righe aventi chiave *k1* e dall'altro replico le righe aventi chiave *k2*, in modo da avere inserimenti esclusivi sui due server (vedi fig. 5.6). Se però non ho nessuna partizione orizzontale, il processo di distribuzione di SQL Server (il *Distribution Process*) si blocca dando errore di duplicazione di chiave. Analizzando le tabelle del *Distribution database*, le quali tengono traccia delle transazioni destinate alla replica, si scopre il motivo: quando inserisco per la prima volta un dato (con valore di chiave *k1*) sul server A, esso viene replicato correttamente sull'altro server; quando successivamente immetto un dato sul server B (con chiave *k2*), il sistema di replica del Server B, copia sia *k1* che *k2* sul server A, dando violazione di chiave nel server A (tentando di inserire un'altra volta una riga con chiave *k1* sul Server A).

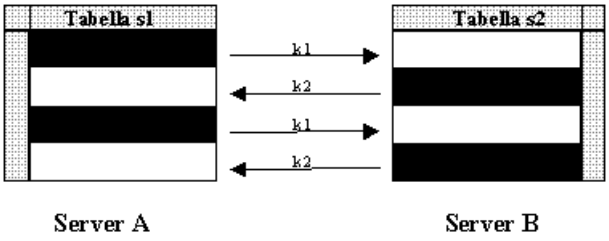


Figura 5.6: Replica bidirezionale con tabelle aventi partizioni orizzontali

- SQL Server dava errori nella definizione dei Trigger:

  1. le variabili di tipo *varchar* non presentavano la dimensione e di default erano settati a 1 da SQL Server; il trigger si bloccava ogni

volta ogni volta che doveva gestire campi di caratteri più grandi di 1;

2. non si possono dichiarare variabili di tipo *text* o *image*, presenti in alcune tabelle di *aws*, all'interno di trigger e store procedure;

- il metodo per distinguere se una modifica avveniva in locale o da replica tramite il controllo sulla variabile di SQL Server "@@NESTLEVEL" non funzionava quando le modifiche al database erano fatte direttamente da Action Workflow: il programma infatti eseguiva le modifiche ad *aws* tramite delle store procedure innestate ed in sequenza, settando a valori non predicibili la variabile "@@NESTLEVEL".

5.2.3 Modifiche alla configurazione di replica

Per quanto riguarda il problema della replica bidirezionale, è stata utilizzata la possibilità, che SQL Server offre, di creare store procedure personali, da essere poi invocate in remoto dal processo di replica in luogo dell'usuale INSERT, UPDATE o DELETE; facciamo un esempio: dovendo replicare una tabella dal Server A al Server B, creo sul Server B una store procedure di inserimento, modifica e cancellazione di dati secondo le caratteristiche che mi interessano; creo quindi un articolo sul Server A che invocherà, all'atto della replica, queste store procedure. Facendo una modifica sul Server A, la store procedure installata sul Server B apporterà questa modifica anche al Server B (lo schema di replica è riportato in fig. 5.7).

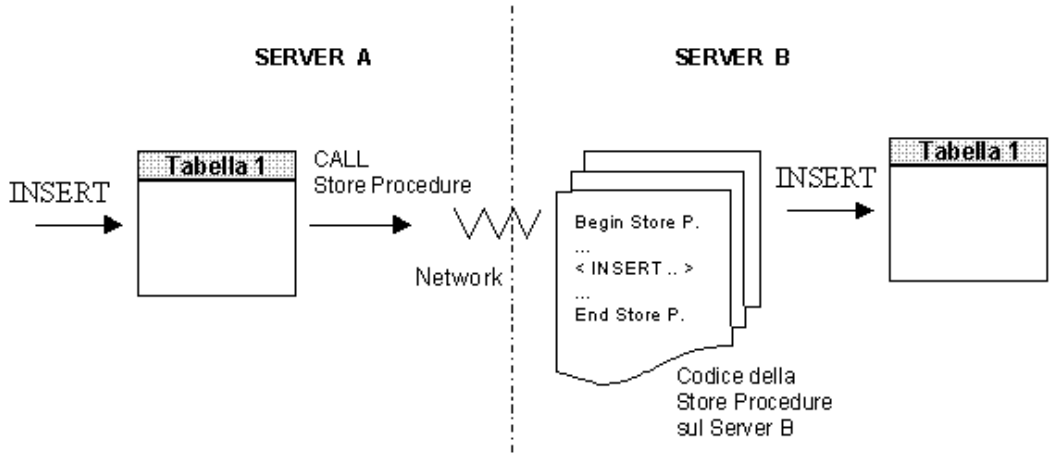


Figura 5.7: Replica bidirezionale: schema di chiamata a procedura remota

In queste store procedure, si provvederà anche all'inserimento dei dati sia

su *aws* che su *awsdup*. Viene qui riportato un esempio di store procedure per l'INSERT:

```
CREATE PROCEDURE spADSDBARCHDFNAMES_INS @ID int,
                 @ArchProfileId int, @BPDefName varchar(65)
AS

/** per evitare conflitti di un doppio inserimento
    viene fatto un controllo sull'esistenza o meno
    del valore da inserire **/

 if NOT exists
    ( select * from ADSDBARCHDFNAMES
         where
           ID_ADSDBARCHDFNAMES = @ID AND
           (ArchProfileId = @ArchProfileId or ArchProfileId is NULL) AND
       (BPDefName = @BPDefName or BPDefName is NULL)
    )

 begin  /* inserimento in awsdup */
   insert ADSDBARCHDFNAMES values (@ID,@ArchProfileId ,@BPDefName)
 end

if NOT exists
    ( select * from aws.dbo.ADSDBARCHDFNAMES
      where
       (ArchProfileId = @ArchProfileId or ArchProfileId is NULL) AND
       (BPDefName = @BPDefName or BPDefName is NULL)
     )

 begin /*inserimento in aws */
  insert aws.dbo.ADSDBARCHDFNAMES values (@ArchProfileId ,@BPDefName)
 end

/* aggiornamento tabella SURR_KEY */

update awsdup.dbo.SURR_KEY
set ID_ADSDBARCHDFNAMES = @ID + 1

GO
```

Queste procedure sono installate solo su *awsdup*, e poichè eseguono l'inserimento anche su *aws*, i trigger su *awsdup* vengono eliminati. Importante è notare come questo tipo di metodologia per la replica bidirezionale, sia anche quella proposta nella guida in linea della nuova versione di SQL Server (la 7.0);

Modifiche ai Trigger

Ricordiamo che i Trigger vengono generati automaticamente dal DOM Builder (scritto in C++) sfruttando la classe MFC (Microsoft Foundation Class) che offre i "metodi" e gli attributi per collegarsi via ODBC a qualsiasi database. Le modifche alla definizione e generazione dei Trigger sono state le seguenti:

- si sono aggiunte le dimensioni di tutti i campi anche alle variabili all'interno dei trigger (sfruttando un ulteriore attributo `m_nPrecision` della classe C++ *CRecordset*);
- le variabili nei trigger corrispondenti a campi delle tabelle di tipo *text* o *image* vengono dichiarati come *binary*;
- per l'univocità (anche tra server diversi) dei cursori all'interno dei trigger, si fa precedere il nome del cursore dal nome del server e della tabella.

Installazione dei Trigger

Il DOM Builder generava dei trigger "nuovi", cioè con le sole funzionalità di duplicazione dei dati da *aws* ad *awsdup*; quindi all'atto dell'installazione (con l'istruzione `CREATE TRIGGER`) venivano cancellati i trigger originali di Action Workflow. Poichè era molto onerosa un'operazione manuale (di tipo copia/incolla) che innestasse i nuovi trigger in quelli originali, si è pensato di rendere quest'operazione automatica nel DOM Builder. La classe C++ *cRecordset*, che rende possibile il reperimento d'informazioni da un database via ODBC, non è fornita di metodi che leggano o modifichino i Trigger. Si sono quindi utilizzate le librerie Visual Basic presenti nel pacchetto d'installazione di SQL Server (le librerie C++ non erano presenti nel pacchetto a noi a disposizione): si è creata un'applicazione Visual Basic, richiamata all'interno del DOM Builder, che legge i Trigger originali di Action Workflow e li concatena adeguatamente con i nuovi trigger generati dal DOM Builder, generando quindi un unico file d'installazione.

Con queste modifiche la replica bidirezionele, con accesso ai dati direttamente dall'interfaccia grafica di SQL Server, funzionava; quando però le modifiche avvenivano da Action Workflow, il sistema di replica si bloccava: esso infatti accedeva ad *aws* tramite delle store procedure in serie che settavano la variabile globale di SQL Server @@NESTLEVEL a valori maggiori di 1, quindi il trigger, che doveva copiare i dati da *aws* ad *awsdup* se @@NESTLEVEL fosse stata uguale ad "1", non scattava. Per risolvere questo ulteriore problema, si è creata una nuova tabella, di nome `TRANSAZ`, con tanti campi interi quante le tabelle di *aws*, tutti inizializzati a valori diversi da zero; le store procedure che copiavano i dati in remoto, sia in *aws* che in *awsdup*, prima della modifica dei dati, settano il campo corrispondente alla tabella interessata alla replica a zero, modificano i dati e quindi risettano questa variabile a un valore diverso da zero. Il trigger, quindi, quando avveniva una modifica dei dati, controlla che il campo corrispondente in `TRANSAZ` sia diverso da zero, e quindi copia i dati anche in *awsdup*, in caso contrario non compie alcuna operazione.

La situazione finale quindi è la seguente: all'inizio del trigger, al posto del controllo "`IF @@NESTLEVEL = 1`", viene inserito il controllo sulla tabella `TRANSAZ` "`if ( Select ID_nometabella FROM TRANSAZ <> 0)`", dove `nometabella` è la tabella su cui si sta cercando di eseguire una certa modifica. Inoltre all'inizio di ogni store procedure di replica si setta il campo corrispondente della tabella `TRANSAZ` a "0" (in modo l'inserimento effettuato dalla store procedure non faccia scattare i trigger):

```
begin transaction
  UPDATE TRANSAZ
  SET    ID_nometabella = 0
commit transaction

    [..]
 /*** codice della store procedure **/
    [..]

begin transaction
 UPDATE TRANSAZ
  SET    ID_nometabella = 10
commit transaction

GO /** fine store procedure **/
```

Si può notare come l'UPDATE della tabella `TRANSAZ` venga fatta in un'unica transazione, per evitare conflitti o modifiche concorrenti inconsistenti.

5.3 Integrazione ad Action Workflow di una gestione di dati distribuiti

La maggior parte dei sistemi per la gestione del Workflow, concentrano le loro funzionalità nella gestione dei processi e attività. La gestione del flusso dei dati è spesso trascurata e anche il sistema da noi utilizzato, Action Workflow, offre in questo senso solo minime funzionalità (gli unici dati trattati sono quelli riguardanti la semplice descrizione del processo, come il nome, i partecipanti, i tempi, ecc..).

Sfruttando il fatto che Action Workflow si "appoggia" sul DBMS SQL Server, e avendo distribuito il sistema su più macchine, si è pensato di integrare a questo pacchetto software proprio una metodologia di gestione dei dati distribuita, progettata ed implementata proprio su SQL Server: il modello Dynamic Ownership.

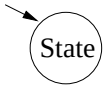
5.3.1 Dynamic Ownership Model

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 e gli altri utenti possono accedervi solo in lettura; grazie a un meccanismo di cambiamento dinamico della proprietà del data-set ogni utente interessato potrà modificarlo a tempo debito, quando ne sarà il possessore.

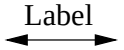
Per descrivere il modello si può utilizzare la simbologia grafica di una *macchina a stati*, in cui sono rappresentati i possibili *stati* che il data-set può assumere e la dinamica con cui può modificare il proprio stato (il risultato di quest'operazione è il *diagramma stati-transizioni*). In ogni 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.

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.
Tramite la transazione per " avocazione" 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 gestire situazioni in cui un utente gerarchicamente più elevato possa imporsi su un subalterno.

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.

Per assegnare un ruolo ad ogni stato si è pensato di "colorare" il diagramma, identificando ogni stato con un colore corrispondente al ruolo:

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.

Si riporta ora per chiarezza un esempio di DST, dove è illustrato l'"iter" d'approvazione di una certa pratica legale (il *data-set*) attraverso i vari organi competenti: la Cancelleria, il Giudice Delegato e la Camera di Consiglio (i *ruoli*):

Collegate al grafo abbiamo poi due tabelle che riassumono i concetti, già rappresentati graficamente, di proprietà degli stati (Tab 5.1), e di transizione di stato (Tab 5.2).

Per la descrizione dell'implementazione del modello su SQL Server, si rimanda a [19] e [18].

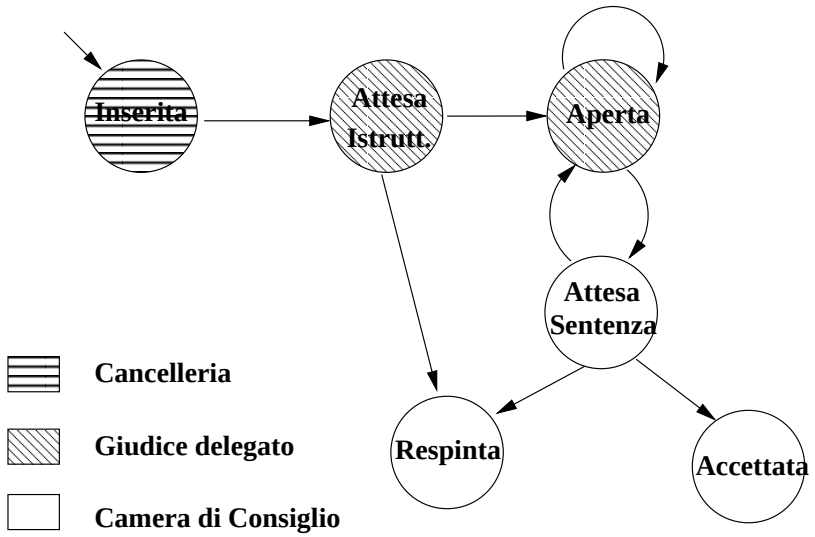


Figura 5.8: DST di una pratica legale attraverso i vari organi legislativi

	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 5.1: Tabella dei possessori di una pratica

5.3.2 Il sistema risultante

Action Workflow System fornisce il supporto ad una efficiente gestione delle attività e processi: l'utente disegna la mappa contenente i flussi di lavoro, la compila e la installa nel Process Administrator; per poi sfruttare le funzionalità del programma, bisogna progettarsi ad hoc un'applicazione in Visual Basic o C++ che, utilizzando le Action Workflow API, vada ad utilizzare i servizi offerti dal prodotto. Ogni utente, abilitato a partecipare ad un certo *business process*, accederà al sistema di workflow in base al determinato ruolo che egli assume; in base al suo ruolo egli avrà determinati compiti e funzioni. Il motore di workflow installato nel sistema "schedulerà" le attività e le azioni eseguite da ciascun utente e le renderà visibili, secondo determinati criteri, ad altri partecipanti al *business process*.

L'installazione del DOM su SQL Server fornisce invece un supporto per una gestione dinamica e distribuita dei dati: un utente può modificare un dato solo se in un certo momento ne è il possesore. Si può quindi notare

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 5.2: Transizioni di stato permesse di una certa pratica legale

come anche in questo caso vi sia l'associazione utente/ruolo.

Integrando le funzionalità di Action Workflow con quelle del DOM, si ottiene proprio un sistema completo sia per la gestione dei processi che per la gestione del flusso dei dati. L'applicazione Visual Basic o C++, che prima era progettata solo per sfruttare i servizi del "workflow engine" di Action Workflow, ora sarà progettata ed arricchita con la capacità di sfruttare la gestione dei dati propria di un DBMS quale SQL Server: quando un utente si collegherà al sistema "Action Workflow + DOM", l'interfaccia dell'applicazione presenterà al suo interno, oltre che a un'indicazione sulle attività eseguite, da eseguire o in esecuzione (la cosidetta *Work Queue* o *Work List*) , anche una rappresentazione più o meno grafica dei dati dati inerenti al particolare *business process*, modificabili o meno in accordo al diagramma DST del DOM.

Rifacendosi all'esempio della pratica legale, quando l'utente si collegherà al sistema tramite l'applicazione Visual Basic o C++, si troverà di fronte una form contente sia i dati del particolare *business process* a cui partecipa (nome, descrizione, partecipanti, ecc.), sia i dati inerenti alla pratica vera e propria (il contenuto delle tabelle che modellano la pratica legale), gestiti secondo le regole del DOM; quindi se ad esempio mi collego come Giudice Delegato, e la pratica legale si troverà negli stati di "Aperta" o in "Attesa d'Istruttoria", sarà da me accessibile anche in scrittura, altrimenti sarà accessibile solo in lettura e quindi non modificabile.

5.3.3 Setup del sistema

Verranno ora descritte più in dettaglio le fasi, da eseguire in sequenza, dell'installazione dell'intero sistema distribuito **Action Workflow System** e **DOM**.

Fase di configurazione del sistema

La fase di configurazione prevede d'installare in ogni sito Action Workflow, e quindi il database *aws*. Vengono inoltre definiti:

- il modello del processo (la mappa), tramite il **Process Builder**, definendo i vari componenti del *business process*; il file verrà compilato e sarà generata una nuova mappa compilata (file di estensione *.awo*) pronta per essere installata nel Process Administrator.
- il progetto del database di supporto all'applicazione di workflow (il database che conterrà i dati coinvolti nel processo aziendale che non possono essere inseriti nella mappa) attraverso il **DOM Builder**; questo ambiente C.A.S.E genererà automaticamente le tabelle del database e gli script per l'installazione della replica in SQL Server secondo le regole del DOM.

Questa fase è stata resa il più uniforme possibile: la definizione dei ruoli e delle identità viene eseguita un'unica volta, all'interno di uno solo degli ambienti di sviluppo prima citati, il DOM Builder: definendo ruoli ed identità all'interno del DOM Builder, essi saranno installati automaticamente anche in Action Workflow.

Configurazione della replica di Action Workflow

Le azioni da compiere in questa fase sono:

- generare da DOM Builder gli script d'installazione della replica del database *aws*; essi sono costituiti da una serie di file d'estensione `.sql` per l'insermento dei trigger (`triggers.sql`) e per l'installazione delle *publication* e articoli (`publication.sql`).
- "ripulire" i database *aws* in tutti siti tramite il file `delinit.sql`;
- inizializzare *aws* in un solo sito tramite il file `init.sql`;
- sincronizzare la replica (in questo modo verrà inizializzato *aws* in tutti i siti);

Fase d'esecuzione

Dopo aver" disegnato" sia il processo che il suo database di supporto, si possono iniziare le fasi d'esecuzione del sistema:

- viene installato il processo (il file *.awo*) nel *workflow engine* all'interno dell'ambiente grafico del **Process Administrator**; esso è automaticamente distribuito a tutti gli altri server definiti nella configurazione di replica: ogni istanza di processo verrà gestita da un insieme di Process Manager adeguatamenti coordinati via replica;

- viene inizializzata la replica del database di supporto al *business process*, installando gli script generati dal DOM Builder.

A questo punto si può realizzare, in Visual Basic o C++, l'applicazione che sfrutterà sia le funzionalità del *workflow engine* che quelle del database distribuito.

viene inserita una riga nella tabella `TXQMAIN`, che contiene un'entry per ogni transazione in esecuzione; quando la transazione avraà termine, verrà settato un particolare campo di questa riga, in modo che il workflow engine, che periodicamente la controlla, se ne accorga; a questo punto tabella (`TXPMAIN`), che memorizza le transazione che hanno raggiunto il *Commit*; e in un processo scheduler residente in memoria (in pratica è composto da due servizi installati in Windows NT: l'*ActionWorflow Administrator Agent*, che interfaccia il Process Administrator a SQL Server, e il *Process Manager* che é il vero motore di workflow).

Conclusioni

Il lavoro svolto nella tesi è partito dall'analisi di un sistema per il workflow management di tipo client/server: provandolo nella pratica si è potuto notare come tra le funzionalità da esso offerte, quali il controllo del flusso delle attività, mancasse la possibilità di avere una gestione efficiente dei dati.

Prima di realizzare il progetto vero e proprio, si è cercato di avere un idea generale sul mondo del workflow, analizzando poi pregi e difetti dei sistemi automatici commerciali per il workflow management; partendo dallo studio fatto nella tesi precedente, in particolare riguardo le analogie tra la modellazione dei processi di workflow e la modellazione del flusso dei dati del DOM, si è continuata la fase di progettazione della distribuzione di Action Workflow, giungendo alla realizzazione di un sistema per il workflow management adatto ad ambienti distribuiti.

In paricolare si è sfruttando il fatto che il sistema utilizzasse come backend server un database SQL Server, e si è quindi pensato di utilizzare la tecnologia di questo DDBMS per potenziarne le funzionalità. Nella prima fase del progetto, si è reso Action Workflow System un sistema distribuito, replicando su più macchine, in maniera bidirezionale, il database *aws*, su cui il sistema memorizzava la definizione dei processi, i tempi d'esecuzione e ogni tipo di azione che l'utente del *business process* eseguiva sulle attività.

Partendo da questo "nuovo" sistema distribuito e osservando che la configurazione dei server era compatibile con quella utilizzata nel modello di gestione di dati del DOM , si è pensato di integrare Action Workflow con il modello Dynamic Owwnership, in modo da avere un flusso dinamico dei dati, in corrispondeza di un flusso dinamico delle attività.

Il sistema risultante era così in grado di utizzare le funzionalità di WFMS per gestire il flusso delle attività e la tecnologia dei DDBMS per avere una gestione efficiente dei dati.

In futuro si potrebbe pensare di completare la distribuzione del sistema, replicando anche il databse *awsarch*, cioè il database utilizzato per archiviare

tutte le istanze dei *business process*. Si potrebbe poi proseguire la fase d'integrazione tra i due sistemi, Action Workflow e DOM: nella tesi si è giunti ad uniformare la definizione dei ruoli e delle identità (si definiscono una volta sola da DOM Builder, sia per la configurazione del DOM, sia per la configurazione di Action Workflow) e un ulteriore passo potrebbe essere quello di "omogeneizzare" la definizione dei flussi di lavoro, in modo da progettare in un unica fase l'intero *business process*.

Appendice A

Esempi di script T-SQL per SQL Server 6.5

In questa appendice vengono proposti esempi di script citati nei capitoli precedenti.

A.1 Script di modifica del database *aws*

In questa sezione mostriamo alcuni degli script generati dal DOM Builder per la configurazione della replica di *aws*.

A.1.1 Aggiunta delle chiavi primarie

Questo script serve ad aggiungere le chiavi primarie alle tabelle che le ammettono, secondo i criteri esposti nel Capitolo 5. Si compone quindi di una lunga serie di statement "ALTER TABLE ADD CONSTRAINT PRIMARY KEY".

```
/* 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

[..]
```

A.1.2 Generazione tabelle *awsdup*

Queste righe di codice servono per creare in *awsdup* copia di quelle tabelle di *aws* su cui non si sono potute aggiungere le chiavi primarie (naturalmente il database *awsdup* deve essere prima oppurtunamente generato tramite l'istruzione SQL `CREATE DATABASE awsdup ON <DEVICE>` ...).

```
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

[..]
```

A.1.3 Tabelle di supporto alla replica

Queste linee di codice creano rispettivamente in *aws* e in *awsdup* le due tabelle `TRANSAZ` e `SURR_KEY` (per il loro utilizzo si rimanda al capitolo 5).

```
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_ADSDBARCHDFNAMES int not null default 1,
     ID_ADSDBARCHIVE int not null default 1,
     ID_ADSDBBPDEFBPDLLMAPNAME int not null default 1,
     ID_ADSDBBPDEFBPROLEMAPPING int not null default 1,

           [...]

     ID_TXMAILADDINFO int not null default 1,
     ID_TXMAILINFO int not null default 1,
     ID_TXPACT int not null default 1,
     ID_TXPADHOC 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_TXPBPINSTANCE int not null default 1,
    ID_TXPCOS int not null default 1,
    ID_TXPCYCLETIME int not null default 1,
    ID_TXPMAIN 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_TXQACT 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_TXQBPINSTANCE int not null default 1,
    ID_TXQCOS int not null default 1,
    ID_TXQCYCLETIME int not null default 1,
    ID_TXQMAIN int not null default 1,
    ID_TXQOBJID 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,
    ID_TXSTFADDINFO int not null default 1,
    ID_TXSTFQUEUE int not null default 1,
    ID_XTRANSLATE int not null default 1,
 )
 go
INSERT INTO SURR_KEY (ID_ADSDBBPDEFBPDLLMAPNAME) values(1)
go


USE aws
go if exists
 (select * from sysobjects where id = object_id('dbo.TRANSAZ')
 and sysstat & 0xf = 3)

 drop table dbo.TRANSAZ
```

```
GO

CREATE TABLE dbo.TRANSAZ (
    tr_ADSDBARCHDFNAMES int not null default 10,
    tr_ADSDBARCHIVE int not null default 10,
    tr_ADSDBBPDEFBPDLLMAPNAME int not null default 10,
    tr_ADSDBBPDEFBPROLEMAPPING int not null default 10,
    tr_ADSDBBPDEFBPROLES int not null default 10,
    tr_ADSDBBPDEFREGISTRY int not null default 10,
    tr_ADSDBDLLREGISTRY int not null default 10,
    tr_ADSDBFXFER int not null default 10,
    tr_ADSDBFXFERIDEN int not null default 10,

            [...]

    tr_TXPTRANSFERIDT int not null default 10,
    tr_TXPUSERDATA int not null default 10,
    tr_TXPWFAPPDATA int not null default 10,
    tr_TXPWFIDT int not null default 10,
    tr_TXPWFPARTICIPANTS int not null default 10,
    tr_TXQACT int not null default 10,
    tr_TXQADHOC int not null default 10,
    tr_TXQBINDATA int not null default 10,
    tr_TXQBPAPPDATA int not null default 10,
    tr_TXQBPIDT int not null default 10,
    tr_TXQBPINSTANCE int not null default 10,
    tr_TXQCOS int not null default 10,
    tr_TXQCYCLETIME int not null default 10,
    tr_TXQMAIN int not null default 10,
    tr_TXQOBJID int not null default 10,
    tr_TXQTRANSFERIDT int not null default 10,
    tr_TXQUSERDATA int not null default 10,
    tr_TXQWFAPPDATA int not null default 10,
    tr_TXQWFIDT int not null default 10,
    tr_TXQWFPARTICIPANTS int not null default 10,
    tr_TXSTFADDINFO int not null default 10,
    tr_TXSTFQUEUE int not null default 10,
    tr_XTRANSLATE int not null default 10,
)
go
INSERT INTO TRANSAZ (tr_ADSDBBPDEFBPDLLMAPNAME) values(10)
```

```
go
```

A.2 Installazione Trigger in *aws*

Questo script, da eseguire in tutti i siti, installa i trigger di copia di dati in *awsdup* concatenati con i trigger originali. Da notare come diversi trigger su una stessa tabella debbano essere concatenati tramite:

```
begin
<codice trigger 1>
end
begin
<codice trigger 2>
end
.
```

Trigger d'inserimento

```
USE aws GO

 /***    File di modifica trigger in aws  (server 1) ***/

if exists (select * from sysobjects where id =
object_id('ADSDBARCHDFNAMES_INS') and sysstat & 0xf = 8)

drop trigger ADSDBARCHDFNAMES_INS

print 'trigger su ADSDBARCHDFNAMES'
go


/*---- ADSDBARCHDFNAMES::Class 61010ffs ------------------*/

/*---- ADSDBARCHDFNAMES::Trigger 6101001s ----------------*/
create trigger ADSDBARCHDFNAMES_INS
  on ADSDBARCHDFNAMES for insert as

     /** Trigger originale **/
  begin
    if (select count(*) from ADSDBARCHIVE, inserted
      where inserted.ArchProfileId = ADSDBARCHIVE.ArchProfileId) = 0
```

```
      begin
    Raiserror 50007 'No such Archive Profile present'
    if @@TRANCOUNT = 1
      rollback transaction
    return
      end
    if (select count(*) from ADSDBBPDEFREGISTRY, inserted
      where inserted.BPDefName = ADSDBBPDEFREGISTRY.BPDefName) = 0
      begin
    Raiserror 50008 'No such BP Definition Name'
    if @@TRANCOUNT = 1
      rollback transaction
    return
      end
  end

    /** Mio trigger **/
begin

if ((select tr_ADSDBARCHDFNAMES FROM TRANSAZ) <>0)
begin declare @ID int
declare @ArchProfileId int
declare @BPDefName varchar(65)

declare @fs int
select @fs = 0
declare A_ADSDBARCHDFNAMES cursor
for select * from inserted open A_ADSDBARCHDFNAMES

while @fs = 0
begin
    fetch next from A_ADSDBARCHDFNAMES into @ArchProfileId, @BPDefName
    select @fs = @@FETCH_STATUS
    if @fs = 0
    begin
        declare penta_SURR cursor for \
              select ID_ADSDBARCHDFNAMES from awsdup.dbo.SURR_KEY
        open penta_SURR
        fetch next from penta_SURR into @ID
        insert awsdup.dbo.ADSDBARCHDFNAMES
         values (@ID, @ArchProfileId, @BPDefName)
```

```
        update awsdup.dbo.SURR_KEY
        set ID_ADSDBARCHDFNAMES = @ID + 1 where ID_ADSDBARCHDFNAMES = @ID
        close penta_SURR
        deallocate penta_SURR
        fetch next from A_ADSDBARCHDFNAMES into @ArchProfileId, @BPDefName
        select @fs = @@FETCH_STATUS
    end
end
close A_ADSDBARCHDFNAMES
deallocate A_ADSDBARCHDFNAMES
end
end
GO
```

Trigger per cancellazione

```
if exists (select * from sysobjects where id =
object_id('ADSDBARCHDFNAMES_DEL') and sysstat & 0xf = 8)
drop
trigger ADSDBARCHDFNAMES_DEL
go

CREATE TRIGGER ADSDBARCHDFNAMES_DEL
    on ADSDBARCHDFNAMES for delete  AS
begin
if ((select tr_ADSDBARCHDFNAMES FROM TRANSAZ) <>0)
begin
declare @ID int
declare @ArchProfileId int
declare @BPDefName varchar(65)

declare @fs int select @fs = 0
    declare A_ADSDBARCHDFNAMES cursor for select * from deleted
    open A_ADSDBARCHDFNAMES
    fetch next from A_ADSDBARCHDFNAMES into
        @ArchProfileId, @BPDefName
        select @fs = @@FETCH_STATUS
        while @fs = 0
    begin
    if @fs = 0
    begin
```

```
        DELETE awsdup.dbo.ADSDBARCHDFNAMES
        WHERE
        (ArchProfileId = @ArchProfileId)
         AND (BPDefName = @BPDefName)

        fetch next from A_ADSDBARCHDFNAMES into
        @ArchProfileId, @BPDefName
        select @fs = @@FETCH_STATUS
        end
end
close A_ADSDBARCHDFNAMES
deallocate A_ADSDBARCHDFNAMES
end
end
GO
```

Trigger per update

```
if exists (select * from sysobjects where id =
object_id('ADSDBARCHDFNAMES_UPD') and sysstat & 0xf = 8)
drop trigger ADSDBARCHDFNAMES_UPD
go

create trigger ADSDBARCHDFNAMES_UPD
    on ADSDBARCHDFNAMES for update  AS
begin
if ((select tr_ADSDBARCHDFNAMES FROM TRANSAZ) <>0)
begin
declare @ID int
declare @ArchProfileId int
declare @BPDefName varchar(65)

declare @fs int
select @fs = 0
    declare A_ADSDBARCHDFNAMES cursor for select * from inserted
    open A_ADSDBARCHDFNAMES
    while @fs = 0
    begin
    fetch next from A_ADSDBARCHDFNAMES into
        @ArchProfileId, @BPDefName
        select @fs = @@FETCH_STATUS
```

```
        if @fs = 0
    begin
        UPDATE awsdup.dbo.ADSDBARCHDFNAMES  SET
        ArchProfileId = @ArchProfileId,
        BPDefName = @BPDefName
        FROM awsdup.dbo.ADSDBARCHDFNAMES d, deleted t
        WHERE
        (d.ArchProfileId = t.ArchProfileId)
         AND (d.BPDefName = t.BPDefName)


        fetch next from A_ADSDBARCHDFNAMES into
        @ArchProfileId, @BPDefName
        select @fs = @@FETCH_STATUS
        end
end
close A_ADSDBARCHDFNAMES
deallocate A_ADSDBARCHDFNAMES
end
end
GO
```

A.3 Script d'installazione della replica di *aws* e *awsdup*

In questa sezione vengono presentati gli script di creazione delle *publication* e degli articoli di replica sia per *aws* che per *awsdup*.

```
/*** Installazione publication in awsdup ***/

use msdb
GO
if exists (select * from systasks where name = 'awsdup_sync')
exec sp_droptask awsdup_sync
go

declare @taskid int
exec sp_addtask 'awsdup_sync','Sync','penta','sa','awsdup',
    1,4,1,4,2,0,0,19981001,0,090000,210000,0,0,
```

```
    0,'',3,2,'',2,1,'',@newid = @taskid OUTPUT

use awsdup
exec sp_addpublication awsdup_pub,@taskid,@status=active
GO

     /** publication di aws **/

use msdb
GO
if exists (select * from systasks where name = 'aws_sync')
exec sp_droptask aws_sync
go

declare @taskid int
exec sp_addtask 'aws_sync','Sync','penta','sa','aws',
    1,4,1,4,2,0,0,19981001,0,090000,210000,0,0,
    0,'',3,2,'',2,1,'',@newid = @taskid OUTPUT

use aws
exec sp_addpublication aws_pub,@taskid,@status=active
GO

        /***  Creazione articoli ***/

use awsdup
go
print 'Articolo su ADSDBARCHDFNAMES'
DELETE aws.dbo.ADSDBARCHDFNAMES
exec sp_addarticle awsdup_pub,ADSDBARCHDFNAMES_art, ADSDBARCHDFNAMES,
ADSDBARCHDFNAMES,@ins_cmd='CALL spADSDBARCHDFNAMES_INS',
@del_cmd='CALL spADSDBARCHDFNAMES_DEL',
@upd_cmd='CALL spADSDBARCHDFNAMES_UPD', @creation_script = NULL,
@pre_creation_cmd='none'
GO

use aws
go
print 'Articolo su ADSDBARCHIVE'
exec sp_addarticle aws_pub,ADSDBARCHIVE_art, ADSDBARCHIVE,ADSDBARCHIVE,
    @ins_cmd='CALL spADSDBARCHIVE_INS',@del_cmd='CALL spADSDBARCHIVE_DEL',
```

```
    @creation_script = NULL,@pre_creation_cmd='none'
GO

use awsdup
go
print 'Articolo su ADSDBBPDEFBPDLLMAPNAME'
DELETE aws.dbo.ADSDBBPDEFBPDLLMAPNAME
exec sp_addarticle awsdup_pub,ADSDBBPDEFBPDLLMAPNAME_art,
ADSDBBPDEFBPDLLMAPNAME,ADSDBBPDEFBPDLLMAPNAME,
    @ins_cmd='CALL spADSDBBPDEFBPDLLMAPNAME_INS',@del_cmd='CALL
spADSDBBPDEFBPDLLMAPNAME_DEL',
    @upd_cmd='CALL spADSDBBPDEFBPDLLMAPNAME_UPD',@creation_script =
NULL,@pre_creation_cmd='none'
GO

use awsdup
go
print 'Articolo su ADSDBBPDEFBPROLEMAPPING'
DELETE aws.dbo.ADSDBBPDEFBPROLEMAPPING
exec sp_addarticle awsdup_pub,ADSDBBPDEFBPROLEMAPPING_art,
ADSDBBPDEFBPROLEMAPPING,ADSDBBPDEFBPROLEMAPPING,
    @ins_cmd='CALL spADSDBBPDEFBPROLEMAPPING_INS',@del_cmd='CALL
spADSDBBPDEFBPROLEMAPPING_DEL',
    @upd_cmd='CALL spADSDBBPDEFBPROLEMAPPING_UPD',@creation_script =
NULL,@pre_creation_cmd='none'
GO

use aws
go
print 'Articolo su ADSDBBPDEFBPROLES'

exec sp_addarticle aws_pub,ADSDBBPDEFBPROLES_art,
    ADSDBBPDEFBPROLES,ADSDBBPDEFBPROLES,
    @ins_cmd='CALL spADSDBBPDEFBPROLES_INS',@del_cmd='CALL
    spADSDBBPDEFBPROLES_DEL',
    @creation_script = NULL,@pre_creation_cmd='none'
GO

use aws
go
print 'Articolo su ADSDBBPDEFREGISTRY'
```

```
exec sp_addarticle aws_pub,ADSDBBPDEFREGISTRY_art,
    ADSDBBPDEFREGISTRY,ADSDBBPDEFREGISTRY,
    @ins_cmd='CALL spADSDBBPDEFREGISTRY_INS',@del_cmd='CALL
    spADSDBBPDEFREGISTRY_DEL',
    @creation_script = NULL,@pre_creation_cmd='none'
GO

   [...]
```

Appendice B

Esempio di progetto di un database distribuito

In quest'appendice verrà presentato un esempio di progettazione di un database distribuito per SQL Server 6.5, dall'analisi dei requisiti, al progetto logico ed implementativo (gli script T-SQL di generazione).
La realtà da modellare tramite questo database è un sistema per la gestione di impianti di vario genere, quali ad esempio impianti per l'energia elettrica, per il gas e per l'acqua.

B.1 Descrizione della realtà da modellare

I principali utenti del sistema, descritti dal codice fiscale, dal nome, dal cognome e dall'indirizzo, sono i seguenti: il manager che organizza e distribuisce gli interventi sugli impianti e gli operatori sul campo che svolgono tali interventi.

Ogni manager ha un username e una o pi password sul calcolatore centrale. Ad un operatore sul campo è associato un livello che ne descrive la propria esperienza; tra gli operatori si distinguono quelli che possono usare un calcolatore portatile e hanno pertanto un username e una password su uno o pi portatili. Ogni operatore appartiene ad un unico gruppo di lavoro, ogni gruppo di lavoro ha un codice identificativo e una descrizione. Ad ogni gruppo di lavoro è assegnato un portatile e all'interno del gruppo di lavoro ci deve essere almeno un operatore con un account sul portatile. I gruppi di lavoro sono caratterizzati da una serie di competenze e classificati da una tipologia di gruppo di lavoro.

Un impianto oltre ad avere un nome, una descrizione e una tipologia, è identificato dalle sue coordinate all'interno dell'area funzionale in cui è

situato; tra gli attributi caratteristici di una zona vi sono alcune informazioni geografiche (in particolare una o pi immagini che rappresentano le mappe della zona) e le date dei giorni lavorativi. Ogni zona è composta da pi aree funzionali. In un'area funzionale sono assegnati uno o pi gruppi di lavoro. Un gruppo di lavoro, a sua volta, in un certo mese dell'anno pu appartenere ad un'unica area funzionale. Ogni impianto fa parte di una rete di distribuzione di cui interessa conoscere, oltre ad una generica descrizione, la lunghezza, la tipologia e le zone che attraversa. Una rete comprende almeno due impianti che rappresentano rispettivamente l'inizio e la fine della rete stessa. Una rete pu essere composta da due o pi sottoreti.

I gruppi di lavoro effettuano sugli impianti una serie d'interventi. Ogni intervento ha una certa tipologia ed è caratterizzato dal tipo di lavoro effettuato. Ogni tipo di intervento pu essere effettuato soltanto da quei gruppi di lavoro che appartengono ad una determinata tipologia. Su un impianto possono essere realizzati pi interventi contemporaneamente a patto che abbiano un codice identificativo diverso e che siano effettuati da gruppi di lavoro differenti. Per quanto riguarda il numero di interventi effettuabili da parte di un gruppo di lavoro l'unico vincolo imposto è il seguente: un gruppo di lavoro non pu iniziare pi di 5 interventi al mese. Le informazioni sugli interventi devono essere gestite giornalmente: per ogni giorno di durata dell'intervento si deve riportare, oltre ad una descrizione del lavoro effettuato in tale data, la composizione della squadra di lavoro, specificando la causa di eventuali operatori assenti. Gli interventi sono distinti principalmente in interventi pianificati annualmente ed interventi straordinari. Ogni intervento pianificato deve iniziare e terminare entro una certa data; ha una durata stimata, ed un indicazione sullo stato di avanzamento dei lavori. Per gli interventi straordinari si deve invece riportare la causa dell'intervento, la data di inizio e di fine lavori. Un intervento straordinario pu essere segnalato da un gruppo di lavoro che durante un altro intervento rileva la necessità di un intervento straordinario. Periodicamente gli interventi pianificati vengono assegnati a gruppi di lavoro scelti tra quelli appartenenti all'area funzionale in cui è situato l'impianto interessato: questa allocazione avviene cercando di minimizzare gli spostamenti dei gruppi di lavoro. Ogni allocazione ha una data di inizio intervento, una data di fine intervento, la data effettiva in cui l'intervento è terminato e una descrizione.

B.2 Distribution Design

La fase di *Distribution Design* si suddivide in due sottofasi: *Frammentazione* e *Allocazione*. Per risolvere questo problema si fa riferimento a un modello

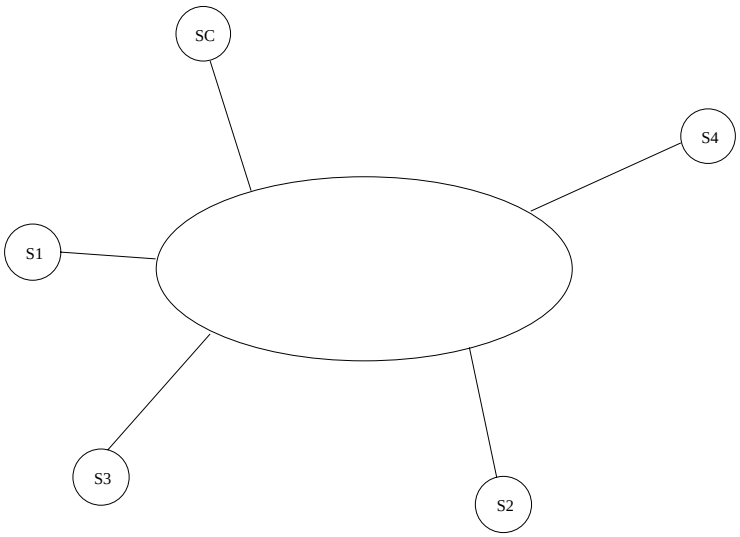


Figura B.1: Configurazione della rete di server

analitico detto *Database Allocation Problem* o DAP: una volta determinati i frammenti delle tabelle da replicare, si studierà quale sarà la loro migliore allocazione.

Dislocazione dei siti

Per applicare in modo significativo questo metodo, si assumerà che esistano una sede centrale e quattro sedi periferiche corrispondenti alle aree funzionali descritte nel testo precedente.

Descrizione della rete

Si assume che la rete sia a commutazione di pacchetto a 56 kbit/sec. Ogni frame ha dimensione pari a 4k (4096 byte); il tempo per trasmettere un frame risulta quindi $T_{pk} = 571$ msec (4*8/56). Protocollo di comunicazione: si trasmette un frame per iniziare un retrivial o un update e si ricevono n frames come risultato (1 pacchetto se il risultato è un acknowledgement).

Database locali

I blocchi di dati hanno dimensione pari a 4 kbyte e tempo d'accesso $T_{IO} =$ 20 msec.

Tabelle interessate alla frammentazione

Si riporta ora lo schema relazionale finale per quelle tabelle interessate alla distribuzione, estratte dallo schema relazionale globale. Tra parentesi si riporta la diensione in byte dei campi di ogni tabella e con l'asterisco si indicano le FOREIGN KEY (relative solo a queste tabelle).

AREA(nome(50),coord_zona(10))

GRUPPO(codice(10),descrizione,(50) tipologia(10),cod_portatile(10),nome_area$(50)_*$)

INTERVENTO(codice(10),tipo_lavori(10),data_fine(6),data_inizio(6),cod_imp$(10)_*$,tipologia(10))

IMPIANTO(codice(10),coord(10),nome(50),nome_area$(50)_*$,descrizione(50),cod_rete(10),tipo(10))

COMPETENZE(competenze(10),cod_gruppo$(10)_*$)

B.2.1 Query e volume dei dati

- Query

  Si riportano in questa tabella la frequenza giornaliera e il sito su cui ogni transazione viene eseguita (SC = Sede Centrale, Reg = sede Regionale).

Transazione	Freq.(day)	Sito
Q1	100	SC
Q2	20	Reg
Q3	50	Reg
Q4	100	SC
Q5	50	SC
Q6	5	Reg
Q7	50	Reg

- Volume dei dati

Relazione	Cardinalità	Lunghezza (byte)	Tuple/ pag	NP	NP/Reg (*)
AREA	4	60	68	1	1
GRUPPO	50	130	31	2	1
INTERVENTO	100000	52	78	1270	318
IMPIANTO	30	190	21	2	1
COPETENZE	5000	20	204	25	7

(*) Si ipotizza una distribuzione uniforme delle tuple su tutti i siti periferici.

B.2.2 Transazioni in SQL

Q1: Selezione dei Gruppi di Lavoro di una certa tipologia

```
SELECT GdiL.*
FROM GRUPPO as GdiL
WHERE Tiplogia = $Tipologia
```

Q2: Selezione di un certo Impianto

```
SELECT Imp.*
FROM IMPIANTO as Imp
WHERE Cod = $COd
```

Q3: Selezione degli Impianti su cui si sta effettuando un certo tipo d'intervento

```
SELECT Imp.*
FROM   IMPIANTO as Imp, INTERVENTO as INT
WHERE  Imp.Cod = Int.cod_Impianto
AND    Int.tipo_lavori = $tipo_lavori
```

Q4: Selezione delle competenze di un certo Gruppo di Lavoro

```
SELECT Cp.*
FROM   COMPETENZE as Cp
WHERE  Cod_grup = $cod_grup
```

Q5: Selezione degli Interventi eseguiti su impianti di una certa zona

```
SELECT Int.*
FROM   INTERVENTO as Int, IMPIANTO as Imp, AREA as Ar
WHERE  Int.cod_impianto = Imp.cod
AND    Imp.nome_area = Ar.nome
AND    c_z = $c_z
```

Q6: Inserimento di un nuovo Gruppo di Lavoro

```
INSERT into GRUPPO
VALUES ($codice,$descrizione,...)
```

Q7: Modifica della tipologia di un Gruppo di Lavoro appartenente all'area funzionale di un certo impianto

```
UPDATE GRUPPO
SET    Tipologia = $Tipologia
WHERE  nome_area = (
                     SELECT  IMPIANTO.nome_area
                     FROM    IMPIANTO
                     WHERE   cod = $cod
                   )
```

Nota: la Query Q7 comporta una scrittura sulla tabella "GRUPPO" e una lettura sulla tabella "IMPIANTO".

B.2.3 Frammentazione orizzontale

Poiché le query selezionano o modificano intere righe delle tabelle, non viene effettuata alcuna frammentazione verticale.

- **AREA(Q5)**: non si genera alcuna frammentazione.
- **GRUPPO(Q1,Q6,Q7)**: si generano 4 frammenti, G1, G2, G3, G4 in base all'area di apparteneza.
- **INTERVENTO(Q3,Q5)**: non si genera alcuna frammentazione.
- **IMPIANTO(Q2,Q3,Q5,Q7)**: si generano 4 frammenti, I1,I2, I3, I4 in base all'area d'appartenenza.
- **COMPETENZE(Q4)**: si raggruppa per gruppi di lavoro e quindi si frammenta per aree funzionali.

B.2.4 Allocazione dei frammenti

Si ipotizza ora che per le transazioni di retrivial si trasmettano solo le tuple necessarie (risultanti da una query) e non di tutto il blocco che le contiene. La tabella rappresenta, per ogni query, il numero di tuple coinvolte, il numero di blocchi (tra parentesi si riporta il valore in caso di frammentazione) e il numero di pacchetti necessari per trasmettere le tuple.

- Volume degli accessi e delle tramissioni

Query	Tuple/Trans	Blocchi/Trans	Pacchetti/Trans
Q1	50 GRUP	2(1)	2
Q2	1 IMP	1	1
Q3	(i) 834 INT	1270	11
	1 IMP	1	1
Q4	100 COMP	1	1
Q5	(ii) 334 INT	1270	5
	2 AREA	1	1
	10 IMP	1	1
Q6	1 GRUP(w)	1	1
	1 AREA(r)	1	1
Q7	1 GRUP(w)	1	1
	1 IMP(r)	1	1

(i) Si suppone che gli interventi (100000) siano equamente divisi come "tipo lavori" in Interventi di manutenzione, riparazione, aggiornamento e sostituzione.
(ii) Si ipotizza che vi siano 10 impianti per zona (8 per area) e che ogni zona comprenda in media 2 aree.

B.2.5 Costo delle transazioni

Si riportano ora i costi delle transazioni (moltiplicati per le loro frequenze). Per i costi di I/O (AC_i) si considerano le alternative (tra parentesi se diversa) di frammentare per area oppure no.
I costi di trasmissione (TC_I) si valutano secondo il protocollo già descritto, il T_{PKT}, e un ritardo di propagazione medio tra sede centrale e sedi periferiche di 9.2 msec. Si farà anche uso delle seguente formula:

$$AC_i = \sum_S \sum_F (u_{ij} + r_{ij}) \cdot sel_j(Q_i) \cdot x_{ij} \cdot LPK_k$$

Dove S sono i siti, F i frammenti; u_{ij} e r_{ij} valgono rispettivamente 2 e 1 in caso di update e lettura, 0 in caso contrario.
$sel_j(Q_i)$ è il fattore si selettività della query Q_i nel sito j-simo;
x_{ij} vale 1 sse il frammento j-simo è presente nel sito k-simo;
LPK_K è il costo totale di accesso al sito k-simo.

Per quanto riguarda TC_i, si ipotizza di eseguire una trasmissione iniziale della query nel sito più il ritorno o di un ACK (1 solo blocco) oppure, se interessano i dati, di N pacchetti. Si ha:

- tempo per trasmettere un pacchetto: 571 msec
- ritardo di propagazione medio: 9.2 msec
- tempo totale di trasmissione di un pacchetto: 580.2 msec

Query	**Sito**	**Relazione**	AC_i**(sec/day)**	TC_i**(sec/day) (messaggi + dati)**
Q1	SC	GRUP	4(2)	58.02+116.04
Q2	Reg	IMP	0.4	11.604+11.604
Q3	Reg	INT	1270	29.01+319.11
		IMP	1	29.01+29.01
Q4	SC	COMP	2	58.02+58.02
Q5	SC	INT	1270	29.01+145.05
		AREA	1	29.01+29.01
		IMP	1	29.01+29.01
Q6	Reg	GRUPPO	0.1	2.9+2.9
		AREA	0.1	29.01+29.01
Q7	Reg	GRUPPO	1	29.01+29.01
		IMP	1	29.01+29.01

Per maggior chiarezza esplicito i calcoli per la quey Q1:

- Senza frammentazione:

$$AC_1(GRUP) = 2 \cdot 20/1000 \cdot = 4sec$$

- Con frammentazione:

$$AC_1(GRUP) = 1 \cdot 20/1000 \cdot = 2sec$$
$$T_{messaggio} = 1 \cdot 580.2/1000 \cdot 100 = 58.02$$
$$TC_1(GRUP) = 1 \cdot 580.2/1000 \cdot 100 = 58.02$$

B.2.6 Allocazione non ridondante

In base al volume degli accessi si riporta uno schema di allocazione non ridondante considerando intere relazioni (non frammenti).
Nella tabella per ogni relazioni si riportano i costi d'accesso (AC_i) relativi alla regione o alla sede centrale (in base all'origine della query).

Relazione	**SC**	**Reg**	**Scelta**
GRUPPO	4	1.1	SC
INTERVENTO	1270	1270	SC
AREA	1	0.1	SC
IMPIANTO	1	2.4	Reg
COMPETENZE	2	0	SC

Si può notare come l'unica tabella frammentata sia IMPIANTO; ora si valuterà la possibilità di avere o meno delle repliche per tutte le relazioni ad esclusione della tabella COMPETENZE.

B.2.7 Copie multiple

Per ogni tabella replicabile si valuteranno i costi e i benefici in base alle query interessate.

1. **AREA**

Transazione	**Sito**	$AC_i + TC_i$ **(solo in SC)**	$AC_i + TC_i$ **(anche nelle Reg)**
Q5	SC	1358.03	1358.03

- Senza replica:
  Si accede alle tabelle INT e AREA localmente alla sede centrale (1270+1);
  accedo in remoto alla tabella IMP (già frammentata), (29.01) e ritrasmetto i dati (29.01);
  accedo localmente a IMP (29.01);
- Con replica:
  La replica della tabella AREA non modifica i costi totali, quindi non si duplica la tabella.

2. **GRUPPO**

Transazione	**Sito**	$AC_i + TC_i$ **(solo in SC)**	$AC_i + TC_i$ **(anche nelle Reg)**
Q6	Reg	11.8	6
Q7	Reg	59.02	2

Hp: la tabella AREA non è replicata.

Query Q6

- Senza replica:
  Si accede in remoto a SC (2.9*2);
  accedo localmente alle tabelle GRUPPO e AREA (0.1+0.1);
  si ritrasmettono i dati da SC (2.9*2).
- Con replica:
  Si accede in remoto alla tabella AREA;
  accedo localmente in SC e poi nel sito regionale (0.1+0.1);
  si ritrasmettono i dati da SC (2.9*2).

Query Q7

- Senza replica:
  Si accede in remoto e poi in locale alle tabelle GRUPPO (29.01+1);
  accedo localmente a IMP (1);
  si ritrasmettono i dati da SC (29.01).
- Con replica:
  accedo localmente alle tabelle GRUPPO e IMP (1+1). si ritrasmettono i dati da SC (2.9*2).

Si replica la tabella GRUPPO (infatti i benefici sono maggiori dei costi).

3. **INTERVENTO**

Transazione	**Sito**	$AC_i + TC_i$ **(solo in SC)**	$AC_i + TC_i$ **(anche nelle Reg)**
Q3	Reg	1909.22	1271

- Senza replica:
  Si accede in remoto a INT su SC (319.11);
  accedo localmente a INT e poi si ritrasmettono i dati (1270+319.11);
  si accede localmente a IMP (1).
- Con replica:
  accedo localmente alle tabelle INT e IMP (1270+1).

Si replica la tabella INTERVENTO (i benefici sono maggiori dei costi).

4. **IMPIANTO**

Transazione	**Sito**	$AC_i + TC_i$ **(solo in SC)**	$AC_i + TC_i$ **(anche nelle Reg)**
Q2	Reg	0.4	0.4
Q3	Reg	1271	1271
Q5	SC	1620.12	1562.1
Q7	Reg	58.02	1271

Query Q2

- Senza replica:
  accedo localmente in SC alla tabella IMP (0.4);
- Con replica:
  accedo localmente in SC alla tabella IMP (0.4);

Query Q3

- Senza replica:
  accedo localmente alle tabelle INT e IMP (1270+1);
- Con replica:
  accedo localmente alle tabelle INT e IMP (1270+1).

Query Q5

- Senza replica:
  accedo localmente alla tabella AREA (1);
  si accede in remoto alle tabelle INT e IMP (145.05+29.01);
  si accede localmente alla tabelle INT e IMP (1270+1);
  si ritrasmettono i dati (145.05+29.01).

- Con replica:
  accedo localmente alle tabelle AREA e IMP (1+1);
  accedo in remoto alla tabella INT (145.05);
  si accede localmente alle tabelle INT (1270);
  si ritrasmettono i dati (145.05).

La query Q7 non cambia a seconda della replica.
Si replica la tabella IMPIANTI anche in SC (i benefici sono maggiori dei costi).

B.2.8 Schema di allocazione finale

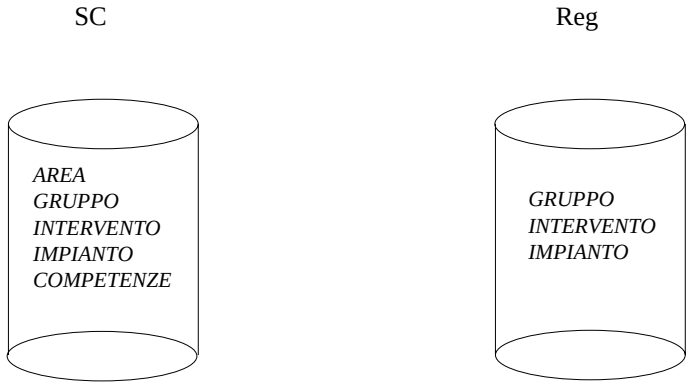


Figura B.2: Schema di allocazione finale

Abbiamo quindi deciso di replicare nei siti regionali le tabelle GRUPPO, INTERVENTO;
la tabella IMPIANTO, originariamente allocata nei siti regionali, viene duplicata nella sede centrale;
non si replicano le tabelle AREA e COMPETENZE.
Le altre tabelle quindi sono allocate nella sede centrale.

B.3 Implementazione del database su SQL Server 6.5

- Creazione dei devices e dei database:

```
use master
disk init name = 'tesinadev',
```

```
physname =  'c:\mssql\data\tesinadev.dat'
vdevno = 99,
size = 10240

create database Tesina on tesinadev = 15
log on tesinadev = 5

go



/* creazione del distribution db **/

use master

disk init name = 'distribution',
physname =  'c:\mssql\data\distribution.dat'
vdevno = 100,
size = 10240

disk init name = 'distriblog',
physname =  'c:\mssql\data\distriblog.dat'
vdevno = 100,
size = 10240

create database Tesina on distribution
log on distriblog

go
```

- Creazione tabelle:

```
CREATE TABLE AREA
( nome char(50),
  c_z char(10),
  CONSTRAINT primary1 PRIMARY KEY (nome),
  CONSTRAINT foreign1 FOREIGN KEY (c_z)
             REFERENCES ZONA (coordinate)
)
```

```
  [..]

CREATE TABLE GRUPPO
( codice char(10),
  descrizione char(50),
  tipologia char(10),
  cod_portatile char(10),
  nome_area char(50),
  CONSTRAINT primary2 PRIMARY KEY (codice),
  CONSTRAINT foreign2 FOREIGN KEY (tipologia)
             REFERENCES TIPOLOGIA (tipologia),
  CONSTRAINT foreign3 FOREIGN KEY (cod_portatile)
             REFERENCES PORTATILE (codice)
)

  [..]
```

- Settaggio registro di configurazione del sistema per il distribution db

```
xp_regwrite 'HKEY_LOCAL_MACHINE',
             'Software\Microsoft\MSSQLServer\replication',
            'DistributionDB','REG_SZ','distribution'

exec(``xp_regwrite 'HKEY_LOCAL_MACHINE',
    'Software\Microsoft\MSSQLServer\replication',
    'WorkingDirectory','REG_SZ','c:\mssql\repldata'')

run INTDIST.SQL
```

- Configurazione del server per la pubblicazione del database

```
sp_serveroption PENTA,'dist',TRUE
go
sp_addpublisher PENTA
go
sp_addsubscriber TELMA
```

```
go
sp_dboption 'Tesina','published',TRUE
go
```

- Definizione della publication e del task di sincronizzazione

```
use msdb
go

declare @taskid int

exec sp_addtask 'grup_table_sync','sync','PENTA','sa','Tesina',1,4,1,
                8,1,0,0,19991231,090000,230000,00000000,160000,0,'',
                2,1,'',2,2,@newid = @taskid OUTPUT

use Tesina
go

exec sp_addpublication tes_pub,@taskid,@status=active
go
```

- Definizione degli articoli della publication

```
use Tesina
go

exec sp_addarticle tes_pub,gruppo_art,GRUPPO,GRUPPO,@cretion_script=NULL

exec sp_addarticle tes_pub,intervento_art,INTERVENTO,INTERVENTO,
                   @cretion_script=NULL

exec sp_addarticle tes_pub,impianto_art,IMPIANTO,IMPIANNTO,
                   @cretion_script=NULL
```

- Sottoscrizione degli articoli della publication

```
sp_addsubscription Tes_pub,gruppo_art,TELMA
go

sp_addsubscription Tes_pub,intervento_art,TELMA
go

sp_addsubscription Tes_pub,impianto_art,TELMA
go
```

Bibliografia

[1] W. Du, A. Elmagarmid, "Workflow Management: State of the Art vs. State of the Products", *HP Labs Technical Reports, Palo Alto, USA, 1998.*

[2] 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/.

[3] Microsoft Corporation, "Microsoft SQL Server 6.5 Administrators Guide: Microsoft Corporation", 1996.

[4] Microsoft Corporation, "Microsoft SQL Server 7.0 Administrators Guide: Microsoft Corporation", 1998.

[5] A. Sheth, D. Georgakopouloulos, S.M.M. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, A. Wolf, "Report from the NSF Workshop on Workflow and Process Automation in Information Systems", *Results of the NSF workshop on Workflow and Process Automation in Information Systems*, Georgia, 1996.

[6] "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/.

[7] Action Technology Inc., "Action Workflow Enterprise Series 3.0: Action Workflow Guide", 1993-1996.

[8] M. T. Ozsu and P. Valduriez, "Principles of Distributed Database Systems", *Prentice Hall International Editions*, New Jersey, 1991.

[9] H. Stark and L. Lachal, "Ovum Elatuates Workflow", *Ovum Evaluates*, Ovum Ltd., London, 1995.

[10] "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/.

[11] 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.*

[12] D. Georgakopoulous and M. Hornik, "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure", *Distribuited and Parallel Databases, Kluwer Academic Publisher, Boston, 1995.*

[13] R. Medina-Mora, T. Winograd, R. Flores, "ActionWorkflow as the Enterprise Integration Technology", *Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society, Vol. 16, No.2, 1993.*

[14] A. Sheth, "From Contemporary Workflow Process Automation to Adaptive and Dynamic Work Activity Coordination and Collaboration", *Large Scale Distributed Information System Lab, University of Georgia, 1996.*

[15] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modeling and Design", *Prentice Hall, 1991.*

[16] I. Jacobson, "Object-Oriented Software Engineering - A Use Case Driven Approch", *ACM Press, Addison-Wesley, 1992.*

[17] M. Amberg, "The Benefits of Business Process Modeling for Specifying Workflow-Oriented Application Systems", *Business Information Systems, University of Bamberg, Germany.*

[18] 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.*

[19] N. Fiorenzi, "Supporto alla gestione di dati distribuiti in applicazioni di workflow mediante tecnologia DDBMS: progetto e realizzazione",*Tesi di Laurea in Ingegneria Informatica, Università di Modena, Luglio 1998.*