Università degli studi di
Napoli “Federico II”

Corso di laurea in Ingegneria Informatica

Analisi comparativa di protocolli per Transazioni Distribuite in ambiente WebServices

RELATORE:
Ch.mo Prof. Antonio d’Acierno

TESI DI LAUREA DI:
Elvio Vittozzi
matr.:041/1725

A.A. 2004-2005

A tutte le persone a cui voglio bene,
e che mi hanno sostenuto
nel lungo corso dei miei studi universitari.

INDICE BREVE

INDICE ESTESO

PREMESSA

Vorrei ringraziare molte persone. Zio Mario, per avermi infuso, già in tenera età, il piacere dello studio e della scienza, per avermi costantemente affiancato nei miei studi da ragazzo, e per aver costituito un insostituibile punto di riferimento nella mia formazione generale. La mia fidanzata, Luisa, per aver condiviso con me quest'esperienza universitaria, e per aver sempre creduto in me, costituendo una fonte di stimolo che mi ha portato a fare sempre del mio meglio. Un ringraziamento particolare va a mia mamma, che tanto ha dato senza nulla chiedere, per tutti i sacrifici che ha superato, non solo economici, i quali, pur sembrando dovuti, anche quando non lo sono, troppo spesso sono passati inosservati. Ricordo bene le tante mattine in cui, ancora buio, si svegliava prima di me, per prepararmi il caffè ed accompagnarmi alla stazione. Ricordo bene i tanti fine settimana, quando io tornavo a casa per i "rifornimenti", che lei trascorreva interamente a prepararmi tutto l'occorrente per il mio ritorno a Napoli. Questi, e tanti, tanti altri "piccoli" gesti, hanno contribuito ad alleggerire il "carico" aggiuntivo che, come ogni altro studente fuori sede, io e mia sorella abbiamo dovuto sostenere e superare. Un affettuoso ringraziamento a tutti loro, che spesso ho trascurato in quest'ultimo periodo, per aver accettato la mia "assenza".

Sempre presente, invece, è stato il professor Antonio d'Acierno, a cui va un ringraziamento altrettanto sentito per la sua disponibilità, i suoi consigli, la modalità con cui si è posto nei miei confronti, e, non meno importante, per avermi proposto una tesi molto interessante. Ho seguito il suo corso di Sistemi Informativi, e l'ho subito apprezzato, sia come insegnante, chiaro e scrupoloso, che come persona, divertente e generosa. Il lettore penserà che questi siano i classici complimenti, solo dovuti, che ogni studente fa sempre al relatore della propria tesi, a meno, ovviamente, di qualche eccezione. Fortunatamente, il mio caso, è proprio fra queste.

Vorrei ringraziare anche gli amici che ho conosciuto durante il soggiorno "napoletano", e che hanno contribuito, non poco, ad alleviare i dispiaceri per la lontananza dal mio tanto amato paese.

Con questo lavoro concludo un lungo, importante, e sofferto capitolo della mia vita, e un altro ne comincio. Non mi rammarico di nulla, rifarei tutto, poiché penso che al di la del nostro volere, della nostra convenienza, e delle nostre scelte, esista un disegno più ampio che governa la nostra esistenza. Alcuni Lo chiamano destino, caso; io Lo chiamo in altro modo. Un grande ringraziamento lo devo, infine, soprattutto a Lui, per avermi dato la possibilità di cimentarmi in questa affascinante esperienza.

PREFAZIONE

Ho impiegato molto tempo per portare a termine questo lavoro di tesi, e ciò è accaduto per svariati motivi. Il primo problema è stato quello di trovare la giusta documentazione. Non è stato facile, poiché l'argomento trattato è ancora oggi in continuo "fermento", e così è possibile imbattersi in molti documenti lunghi, complessi, ma dal contenuto particolarmente instabile, e frequentemente di dubbia professionalità. Solo dopo diverse settimane di ricerche sul Web, sono riuscito finalmente a trovare i riferimenti corretti, tra cui i documenti originali di tutte le Specifiche.

Il secondo problema è stato quello della comprensione, aggravato dalla notevole lunghezza di tutti i documenti esaminati (ad esempio, solo BTP è lungo circa 200 pagine), poiché quasi sempre ho avuto a che fare con documenti tecnici, rivolti a persone con una cultura molto specifica. Purtroppo, seppure il corso di Ingegneria dia molte nozioni fondamentali, è sempre necessario un grande salto di conoscenze per capire ciò che viene utilizzato nella pratica, soprattutto in ambito informatico. Per capire le Specifiche, era propedeutica, infatti, un'adeguata comprensione della tecnologia dei WebServices, la quale, basandosi su molteplici Standard, richiede a sua volta la comprensione di questi ultimi. SOAP, WSDL, UDDI, WS-ReliableMessaging, WS-Security, sono solo alcuni esempi. Del resto, capire tutte queste Specifiche, consente di inquadrare il problema solo nel particolare contesto dei WebServices. Poiché volevo proporre una soluzione "alternativa", per i motivi che dirò di qui a breve, non solo era necessario considerare, senza eccezioni, tutte le soluzioni già proposte, ma si richiedeva anche una minima conoscenza di quanto già esistente in altri contesti. Così, ho cercato di colmare, per quanto possibile, la mia ignoranza sulle tecnologie alternative e/o complementari ai WebServices, quali JavaBeans, JSP, Applet, EJB, JTA etc.

Capito il problema e le soluzioni disponibili, notai che le Specifiche valutate proponevano modelli Transazionali molto differenti da quello che io avevo immaginato. Così, forse preso dall'entusiasmo, decisi di realizzare subito un modello *ex-novo*, secondo quella che era la mia idea, e che, intuitivamente, ritenevo fosse l'idea migliore.

Tuttavia, su consiglio del professore, abbandonai quel lavoro, poiché, giustamente, la realizzazione di un nuovo modello avrebbe dovuto essere opportunamente motivata, e ripresi ad analizzare in maniera dettagliata tutte le varie proposte, per descriverle una ad una.

Qui incontrai il terzo problema: come descrivere un documento tecnico, quale è un documento di Specifica, mantenendo la leggibilità, ma senza perdere informazioni, potendo avere, ognuna di esse, un peso importante in seguito? Inoltre, pur affrontando lo stesso problema, le Specifiche sono tra loro molto differenti, almeno nella forma. Decisi, così, non solo di sacrificare parte della leggibilità e della scorrevolezza del testo (per le quali mi scuso con il lettore), fornendo una descrizione il più possibile vicina ai documenti originali e priva di mie considerazioni, ma decisi anche di rappresentare in UML i modelli proposti dalle Specifiche, in maniera da avere un riferimento comune e uniforme.

Grazie a questa analisi dettagliata, potei riconoscere che esistevano molte analogie, ma anche altrettante differenze. Tuttavia, non era semplice capire i motivi da cui tali differenze, apparentemente insignificanti, derivavano; tanto più che queste dovevano giustificare la decisione, presa, tra gli altri, da Microsoft, IBM, Bea Systems e Sun Microsystems, di sviluppare un proprio lavoro, dissociandosi dal comitato dell'OASIS con cui avevano collaborato per diverso tempo alla stesura di BTP. A questo punto, il mio obiettivo primario era cambiato: confrontare le soluzioni. Ma rispetto a cosa? Tutte le soluzioni proposte risolvono il problema, nella sua generalità, per il quale sono state sviluppate.

Questo ha rappresentato la quarta grande difficoltà. Fortunatamente, dopo qualche pomeriggio di ricerca, trovai gli scenari da cui erano partite tutte le Specifiche, ovvero gli scenari di BTP (ricordo che BTP rappresenta una radice comune per tutti le successive Proposte). Grazie a questi avrei potuto risalire ai requisiti del problema, e trovati i requisiti, sarebbe stato possibile confrontare le soluzioni. Qui la difficoltà maggiore è stata proprio quella di individuare i singoli requisiti, poiché questi sono tra loro fittamente correlati, e spesso contrastanti. Decisi, così, di individuarli separatamente, sia dal punto di vista del richiedente, che dal punto di vista di ogni servizio esecutore, e per ragionare con una certa linearità, adottai dei grafi "causa-effetto" che poi, nella forma finale, sono stati proposti in "1.4 Analisi e specifica dei

Requisiti". A questo punto dovevo solo valutare le Specifiche rispetto ai requisiti individuati, ma, l'ambiguità presente in esse, forse con la partecipazione della mia non perfetta conoscenza dell'inglese, non mi agevolava il compito. Tuttavia, in seguito a qualche rompicapo, il confronto era riuscito e, con grande sorpresa, potei constatare che il "mio modello", seppure migliorabile dall'esperienza acquisita, rappresentava ancora, a mio modesto parere, la soluzione migliore. Per questioni di tempo, non è stato possibile formalizzarlo nella presente tesi: basti pensare che i sei editori di BTP, con l'aiuto di circa trenta professionisti del settore, e con tutta la loro esperienza, hanno impiegato quasi un anno per realizzare la prima versione della Specifica. Fortunatamente, ciò non rappresenta una mancanza considerevole, poiché il suo valore è rappresentato essenzialmente dalle idee in esso presenti, e queste sono tutte deducibili direttamente dalle varie osservazioni fornite nel corso dell'intero testo, dalle critiche portate alle Specifiche trattate, e dai "consigli" proposti in "3.6 Possibili soluzioni e miglioramenti" per migliorare dette Specifiche.

INTRODUZIONE

Dall'avvento di Internet ai nostri giorni, abbiamo assistito a notevoli progressi, e talune volte a vere e proprie rivoluzioni.

Inizialmente esisteva solo HTML, siti statici, e poca flessibilità. Il Web veniva utilizzato essenzialmente come un grande "libro" per la pubblicazione di informazioni da condividere, sotto forma di *ipertesto*, alle quali l'utente poteva accedere in maniera semplice e interattiva, senza i vincoli di una rigida struttura fisica sequenziale.

Successivamente, le crescenti necessità di fornire informazioni aggiornate, e dunque di manutenzione, hanno portato ad una veloce integrazione dell'ambiente Web con quello delle basi di dati. Sono nati, così, i siti dinamici, ovvero delle vere e proprie applicazioni Web, in cui le pagine vengono generate in-linea sulla base delle richieste utente, e in funzione della logica applicativa. Inoltre, sono stati definiti opportuni meccanismi per supportare il concetto di *sessione*, e superare le limitazioni dovute alla proprietà di *connectionless* del protocollo http, usato da HTML. La tecnologia che riuniva il mondo Web e quello delle basi di dati per la costruzione di siti dinamici era CGI (Common Gateway Interface). Tale tecnologia, tuttora ancora utilizzata, è stata resa obsoleta dalle moderne ASP, PHP, JSP, JavaScript, Applet, Servelet, EJB, le quali hanno consentito, tra l'altro, di ridistribuire il carico applicativo tra client e server, eliminando o attenuando i difetti di performance legati all'adozione obbligata del modello *Thin Client*. Ciò, ovviamente, è stato reso possibile dal continuo incremento delle capacità di calcolo dei comuni personal computer utilizzati come client, che hanno raggiunto potenze elaborative talvolta superiori a quelle delle postazioni server.

Così, grazie alle nuove possibilità offerte dall'ambiente Web, sono nate e proliferate nuove tipologie di applicazioni, prima fra tutte quella ormai diffusa dell' *e-commerce*.

Contemporaneamente sono andate affermandosi tecnologie per la *distribuzione di contenuto*, tutte estensioni del noto modello di Remote Procedure Call (RPC). Esempi ne sono CORBA, IIOP, DCOM, Java RMI. Queste sono particolarmente indicate per realizzare applicazioni distribuite che prevedono la collaborazione di servizi

appartenenti alla stessa organizzazione, e, pertanto, spesso collocati all'interno di una singola sottorete, comunemente denominata *intranet*.

Recentemente, l'attenzione per l'integrazione dei servizi è stata spostata dall'interno di una intranet ad una *extranet*, ovvero ad una rete che riunisce più intranet (anche solo virtualmente), in cui coagiscono servizi appartenenti ad organizzazioni distinte ma che interagiscono strettamente. Ad esempio, una extranet può collegare un'azienda con i propri fornitori, un'agenzia di viaggio con i tour operators, oppure banche diverse ai fini di operazioni interbancarie. In tal caso si parla di relazioni "business to business", o multi-dominio, per distinguerle dalle relazioni intra-dominio, che in una intranet avvengono all'interno di un singolo dominio business.

Ancora più recentemente, le potenzialità offerte dalle collaborazioni all'interno di una extranet sono state concepite come il primo passo verso una collaborazione globale, tra servizi appartenenti a qualsiasi organizzazione presente sul Web. Tuttavia è noto che organizzazioni differenti spesso utilizzano tecnologie diverse, e l'integrazione diventa sovente un problema che richiede una costosa risoluzione. Così, ad esempio, se un'azienda decide di cambiare fornitori, si trova quasi sempre in notevoli difficoltà.

La crescente esigenza di dinamicità delle aziende, alle quali si richiedono adattamenti rapidissimi ad ogni nuova situazione di mercato, ha incontrato ultimamente il supporto dei produttori di software: nasce una nuova tecnologia, nota come "Web Services". Essa si proprone come uno standard *aperto*, proprio per regolare le interazioni multi dominio, ed ormai è supportata da tutti i maggiori produttori software mondiali, quali Microsoft, Sun, Oracle, IBM, per citarne solo alcuni.

La sua fama è dovuta essenzialmente a due fondamentali caratteristiche:

- Interoperabilità[1]: permette la collaborazione di servizi indipendententemente dalle macchine e dai linguaggi di programmazione sottostanti, in maniera semplice ed economica.
- Architettura orientata ai servizi[2]: tale collaborazione si esplica attraverso l'architettura SOA (acronimo di Services Oriented Architecture), la quale,

[1] L'interoperabilità è stata ottenuta adottando protocolli di comunicazione a livello di Applicazione dello stack ISO/OSI, e utilizzando *standard aperti*, quali SOAP, UDDI, WSDL e XML, comunemente accettati da tutti i produttori.

[2] Il modello architetturale SOA rispecchia pienamente il modello di programmazione tradizionale, fatto di chiamate a funzioni e procedure, esteso ad un ambiente distribuito.

completamente in accordo alla nozione di *incapsulamento* tanto famosa in ambito Object Oriented, consente che ogni organizzazione possa esportare all'esterno la propria logica business, e che altre organizzazioni possano utilizzarla per realizzare ulteriori servizi, senza tuttavia dover conoscere nulla sulla logica interna dei servizi utilizzati.

In tal modo è anche possibile realizzare applicazioni distribuite seguendo il modello di programmazione classico, dimenticando le notevoli difficoltà introdotte dai precedenti modelli di sviluppo Web.

Purtroppo, data la scarsa maturità di questa neonata tecnologia, esistono delle aree non ancora perfettamente definite, e una di queste riguarda il supporto per le Transazioni Distribuite.

Il concetto di Transazione rappresenta notoriamente una unità elementare di lavoro, svolta da una applicazione, avente particolari caratteristiche di correttezza, robustezza e isolamento. Le tecniche per supportarlo in ambito distribuito sono disponibili ormai da molti anni, e sono fornite dalla quasi totalità dei DBMS commerciali. Esse nascono per essere utilizzate in un ambiente caratterizzato dalla presenza di una rete veloce e affidabile: una rete intranet, o ad essa assimilabile per caratteristiche di velocità e affidabilità. Una rete di questo tipo, coincide spesso con la rete locale di un'azienda, ovvero con quello che abbiamo indicato come singolo *dominio business*, per evidenziarne le caratteristiche di autonomia e indipendenza del controllo.

Tali tecniche possono ancora essere utilizzate nel caso di interazioni multi dominio? quando il mezzo di comunicazione sia lento ed inaffidabile come Internet e quando un controllo centralizzato, per ovvi motivi di autonomia delle parti in gioco, non possa esistere?

La risposta non è semplice. Come vedremo, il problema è particolarmente complesso, e alla sua intrinseca complessità, si aggiunge la necessità di dover trovare una soluzione generale, capace di estendere l'interoperabilità dei Web Services anche ad un ambiente transazionale. In altri termini, si aggiunge la necessità di un nuovo standard.

Essendo il supporto per Transazioni Distribuite un elemento chiave per la realizzazione di moltissime relazioni business to business, i maggiori produttori software mondiali, negli ultimi anni, si sono dati da fare per sopperire a tale mancanza,

e seppure le prime Specifiche sull'argomento siano disponibili già dal "lontano" 2001, non esiste ancora nessun risultato definitivo. Nello stesso periodo di tempo, considerando che siamo quasi nel 2006, la Microsoft ha sfornato almeno altre due versioni del suo sistema operativo, la Intel è passata da processori con velocità dei centinaia di MegaHertz a velocità dell'ordine dei GigaHertz, le telecomunicazioni sono state rivoluzionate dalla diffusione delle trasmissioni a banda larga. Come mai non è stato prodotto ancora nessun risultato concreto per supportare Transazioni Business in ambiente WebServices?

L'ambizioso scopo della presente tesi è proprio quello di capire a fondo il problema, descrivere le soluzioni attualmente proposte dai Produttori, e analizzarne criticamente eventuali pregi e difetti.

DEFINIZIONI, ACRONIMI E ABBREVIAZIONI

Per comodità del lettore, in questa sede riportiamo le definizioni dei termini, con un significato particolare e di comprensione non immediata, più frequentemente utilizzati nel corso del presente testo.

Agente software:	Uno o più componenti software che, insieme, svolgono attivamente una precisa funzionalità, nell'ambito di una più ampia applicazione.
Agente:	Con la lettera maiuscola, utilizzato in luogo di "agente software"
Arruolamento:	L'atto con cui un servizio viene "iscritto" come partecipante in una transazione.
Attività Business:	Sinonimo di *transazione business*.
Attore:	Una delle parti coinvolte nell'esecuzione di una transazione. Può coincidere, a seconda del contesto, con una persona fisica o con un sistema software.
Binding:	Regole che definiscono la rappresentazione, la codifica e la trasmissione dei messaggi su un particolare protocollo di Trasporto.
Cliente:	Ruolo dell'*attore* che avvia una transazione.
Dominio Business:	Rappresenta l'insieme degli attori, gestiti da una singola organizzazione, utilizzati per ottenere un certo *business*. Solitamente questi attori sono confinati all'interno di una intranet o di una extanet.
Dominio di fiducia:	Rappresenta un insieme di attori tra i quali sono possibili, e volute, comunicazioni con particolari caratteristiche di

	sicurezza (confidenzialità, integrità, autenticità, etc.).
Esecutore:	Ruolo di ogni *attore* che esegue una delle operazioni coinvolte in una transazione.
Implementazione:	Con la lettera maiuscola, e in riferimento ad una certa Specifica, indica una generica implementazione software conforme alla Specifica in questione.
Macchina:	Elaboratore elettronico su cui risiede un *attore*.
Modalità di inversione:	La tecnica impiegata per riportare il sistema ad uno stato consistente in seguito all'Abort di una transazione.
Operazione transazionale:	Un'operazione, relativa ad una singola *richiesta applicativa*, eseguita come parte di una transazione business.
Partecipante:	Sinonimo di *esecutore*.
Peer:	Indica una delle due entità che partecipano ad una relazione biunivoca.
Preparato:	Aggettivo utilizzato per indicare lo stato di un Attore che, partecipando ad un protocollo di coordinazione, dichiara di poter raggiungere uno stato finale *consistente* con la decisione del coordinatore.
Protocollo di Trasporto:	Rappresenta il protocollo utilizzato da una Implementazione per trasportare i messaggi attraverso la rete sottostante. Nota: non va interpretato come "un protocollo del livello del Trasporto" definito dal modello ISO/OSI.
Richiedente:	Sinonimo di *cliente*.
Richiesta applicativa:	Un messaggio inviato dal *richiedente* ad un *esecutore* per richiedere un certo lavoro come parte di un'Attività.
Sicurezza:	Con la lettera maiuscola, è utilizzato, per indicare un insieme di caratteristiche generiche che consentano, in qualche maniera, una comunicazione (scambio di

	messaggi) sicura.
Specifica:	Con la lettera maiuscola, indica un “documento di specifica”.
Transazione Business:	Transazione distribuita multi dominio, ovvero che richiede interazioni tra *domini business* differenti.
Transazione distribuita:	Esistono almeno due esecutori.
Transazione locale:	Esiste un solo esecutore, e risiede sulla stessa macchina del richiedente.
Transazione remota:	Esiste un solo esecutore, ma risiede su una macchina distinta dal richiedente.
AT	Acronimo di Atomic Transaction.
BA	Acronimo di Business Activity
WS-Tx	Abbreviazione di WS-Transactions
ACID-Tx	Abbreviazione di ACID-Transactions
BP	Acronimo di Business Process

In ambito BTP

Coesione:	Sinonimo utilizzato in luogo di “transazione coesiva”.
Inferiore:	Un Attore nel ruolo di Inferior.
Partecipante:	Un Inferior semplice, ovvero che non si comporti da coordinatore rispetto ad altri inferiori.
Relazione Business:	Una qualsiasi informazione di stato condivisa e distribuita tra le parti, che sia soggetta a vincoli contrattuali accettati dalle parti stesse.
Superiore:	Un Attore nel ruolo di Superior.
Transazione atomica:	Rappresenta una *transazione business* di tipo Atom, ovvero avente la proprietà dell’Atomicità.
Transazione Business:	Un cambiamento **consistente** nello stato di una *relazione*

	business tra due o più parti.
Transazione coesiva:	Rappresenta una *transazione business* di tipo Cohesion, ovvero avente la proprietà dell'Atomicità rilassata.

In ambito WS-Transactions

Attività distribuita:	Un'unità computazionale costituita da più tasks distribuiti su nodi elaborativi distinti.
Attività:	Con la lettera maiuscola, viene usato come sinonimo di "attività distribuita".
Coordinazione:	Usato impropriamente al posto di "attività di coordinazione", in riferimento alle attività con cui un servizio coordinatore esegue un protocollo di coordinamento.

In ambito WS-CAF

Partecipante:	Nei modelli definiti da WS-CF e WS-TXM il termine viene utilizzato con il significato consueto (definito in ambito generale); nel modello definito da WS-CTX, invece, indica una qualsiasi entità che in qualche maniera sia coinvolta in un'Attività.
Attività:	come in WS-Transactions.
Coordinazione:	Con la lettera maiuscola, indica l'atto del coordinatore di "disseminare" informazioni ad un certo numero di partecipanti.
Attività Coordinata	Un particolare tipo di Attività, all'interno della quale sono eseguite delle Coordinazioni.

OVERVIEW

La restante parte della presente tesi è organizzata in tre sezioni principali:

- **Il problema**: fornisce un background al lettore, introducendo il concetto di transazione, spiegando le problematiche inerenti le Transazioni in generale e le Transazioni Distribuite in particolare, e descrivendo le tecnologie comunemente adottate per supportare tale concetto. Analizza, inoltre, alcuni scenari di casi d'uso per Transazioni Business, e definisce i requisiti di un *middleware* di supporto per Transazioni Distribuite in ambiente Web Services.
- **Soluzioni Attuali**: descrive le proposte fornite attualmente dalle maggiori case produttrici, tra cui BEA Systems, Hewlett-Packard, Sybase, Choreology, Sun Microsystems, Microsoft, IBM, Oracle Corporation, Arjuna Technologies, Fujitsu, IONA Technologies, Intel.
- **Analisi e Confronto delle Soluzioni**: analizza e confronta le varie proposte evidenziandone pregi e difetti, e suggerisce delle tecniche per colmarne le lacune.

Seppure nella prima sezione della tesi vengano fornite molte nozioni basilari, data la vastità degli argomenti coinvolti, per una comprensione adeguata di quanto seguirà, si richiede al lettore una buona familiarità con i sistemi transazionali in generale, l'UML, i Web Services, e con i concetti basilari delle reti di calcolatori.

1 IL PROBLEMA

Quando si pensa ad una transazione, si immagina quasi sempre la classica operazione bancaria di giroconto: un passaggio di denaro da un certo conto corrente ad un altro. Del resto, questo è anche l'esempio che viene riportato nella maggior parte dei testi per introdurre l'argomento, in quanto, molto semplicemente, permette di descrivere il concetto stesso di transazione inquadrandolo nel contesto "transazionale" per eccellenza: necessità di massima sicurezza, correttezza e affidabilità. Si potrebbe essere portati a pensare, così, che sia un concetto moderno.

Al contrario, il termine "transazione" è stato utilizzato sin dall'antichità, e conserva intatto il suo significato fino ai nostri giorni. Infatti, nella lingua italiana, il termine viene ancora così definito[3]:

> "*Transazione: dal tardo latino transactionem, da transactus = transatto, 1.Accomodamento, soluzione di compromesso; 2.Contratto col quale le parti, facendosi reciproche concessioni, pongono fine ad una lite già cominciata o prevengono una lite che può sorgere tra loro; 3.Operazione commerciale di compravendita; 4.Ciascuno dei fenomeni elementari che vengono trattati in una elaborazione da un sistema per l'elaborazione dei dati.*"

In altre parole, nel concetto di transazione si ritrovano quelli di *accomodamento di contratto,* eventualmente specializzato in un'operazione commerciale di compravendita, e di *prevenzione di lite*. Inoltre, si deduce che una transazione coinvolge sempre due o più parti. Dunque, una transazione deve consentire l'esecuzione di un contratto tra parti distinte, evitando l'insorgere di eventuali situazioni ambigue, che possano portare a controversie tra le parti stesse.

Quanto detto è certamente vero per l'operazione di giroconto, ma esistono anche altri tipi di operazioni, in contesti meno vincolanti di quello bancario, cui il concetto di transazione si addice parimenti.

[3] Definizione presa da Riferimenti[14]

Ad esempio[4], immaginiamo di trovarci all'interno della nostra auto quando, ascoltando la radio, sentiamo che il nostro cantante preferito si esibirà in concerto nella nostra città. Così, prendiamo subito il cellulare e chiamiamo l'agenzia per prenotare due posti. Dall'altro lato ci risponde l'addetto alle prenotazioni, che purtroppo ci informa di un guasto al sistema informatico. Ci dice di ritentare più tardi. Noi non potremo sapere quando il sistema sarà ripristinato, e così saremmo tentati di richiamare subito: la cosa migliore sarebbe stata che l'addetto si fosse segnato la nostra prenotazione, per registrarla non appena il sistema fosse stato ripristinato. Tuttavia, anche in tal caso, avremmo fiducia che l'addetto effettui la prenotazione, e non se ne dimentichi?

Dopo mezzora richiamiamo, e l'addetto ci dice che il sistema ha ripreso a funzionare. Ci dice anche che sono rimasti solo due posti, entrambi in ultima fila (contemporaneamente l'addetto blocca i due posti, per evitare che da altre agenzie questi possano essere utilizzati mentre stiamo per prendere una decisione). Noi ci fermiamo qualche attimo a pensare, accettiamo, e infine, proprio quando stiamo per comunicare il codice della nostra carta di credito, la linea telefonica cade. Cosa è successo? Abbiamo effettuato la prenotazione? Non lo sappiamo e dobbiamo richiamare. Supponendo che ci risponda un altro operatore, dobbiamo rieffettuare l'operazione rischiando di pagare per 4 biglietti? (L'operatore non può dirci telefonicamente, per problemi di privacy, se la prenotazione precedente è andata a buon fine).

Oggi, questo tipo di operazioni, sono sempre più frequentemente condotte via Web. L'utente interagisce direttamente con un'applicazione piuttosto che con un operatore umano, e molti più utenti possono effettuare le stesse operazioni contemporaneamente, con un conseguente abbattimento dei costi di gestione del servizio. Tuttavia, anche in questo caso, possono parimenti presentarsi tutti i problemi menzionati precedentemente.

Nell'esempio, inoltre, il sito, come l'operatore, funge da intermediario verso un singolo servizio. Grazie alla tecnologia dei Web Services, invece, si potranno sviluppare semplicemente applicazioni che coinvolgono molti più servizi, ma, ovviamente, i problemi aumenteranno di conseguenza.

[4] Esempio tratto da Riferimenti[30].

Quando la composizione dei servizi potrà avvenire con livelli di sicurezza, correttezza e affidabilità vicini a quelli con cui viene condotta l'operazione bancaria, costituirà un elemento chiave per lo sviluppo e la crescita di molte aziende che operano sul Web. Tuttavia, rispetto all'operazione bancaria, pur presentandosi molte caratteristiche e requisiti in comune, si presentano importanti differenze. Essa prevede, infatti, la cooperazione di servizi lascamente accoppiati, ovvero per i quali esista una forte separazione nello spazio, nel tempo, nella proprietà e nel controllo. Non a caso si parla comunemente di Transazioni Business, e non può esistere Business se non tra servizi di proprietà autonoma.

Nel prossimo paragrafo tratteremo un noto tipo di transazioni, dette transazioni ACID, poichè godono di un insieme di proprietà ben definite. Nel successivo, descriveremo le tecnologie adoperate dai DBMS (Data Base Management System) per supportarle, concentrando la nostra attenzione al caso delle transazioni Distribuite. Nel paragrafo "1.3 Analisi di casi d'uso per Transazioni Business", cercheremo invece di caratterizzare le Transazioni Business, valutando una serie di scenari dai quali poter estrarre dei requisiti comuni. Infine, nel paragrafo "1.4 Analisi e specifica dei Requisiti", analizzeremo e specificheremo i requisiti che, secondo noi, dovrebbero essere soddisfatti da qualsiasi Modello per la coordinazione transazionale, sul quale possa basarsi un *middleware* di supporto per Transazioni in ambiente WebServices.

Nel seguito, per poter distinguere i ruoli delle *parti* coinvolte nell'esecuzione di una transazione, utilizzeremo le seguenti due definizioni:

- *Richiedente (o Cliente)*: ruolo dell'attore che avvia la transazione.
- *Esecutore (o Partecipante)*: ruolo di ogni attore che esegue almeno una delle operazioni costituenti la transazione.

Ad esempio, in una semplice transazione di compra-vendita *on-line*, potremo avere tre soli attori: il cliente, il servizio di pagamento del cliente, e il servizio di vendita. Il cliente fungerà da richiedente, mentre i due servizi da esecutori.

1.1 Transazioni ACID

L'ultima definizione di Transazione data nella precedente introduzione - nell'ordine la quarta - conferisce un significato più attuale al termine, specializzandolo in un contesto per l'elaborazione dei dati, nel quale il *contratto* si esplica nell'esecuzione di una operazione come indivisibile ("fenomeni elementari").

Riunendo i quattro significati in ambiente informatico, specializzando i primi tre e generalizzando il quarto, si ottiene come risultato la definizione[5]:

"Transazione: identifica una unità elementare di lavoro svolta da una applicazione, cui si vogliono associare particolari caratteristiche di correttezza, robustezza e isolamento."

In accordo con quest'ultima definizione, e con particolare riferimento all'ambito delle basi di dati, gli studi[6] di Haerder e Reuter, pubblicati nel 1983, portarono alla definizione di specifiche caratteristiche, cui dovrebbe sottostare una transazione per essere considerata tale, oggi note con il nome di caratteristiche **ACID**[7]:

- **Atomicità**: una transazione deve essere eseguita come se fosse una singola operazione, ovvero deve avere la proprietà del "tutto o niente".
- **Consistenza**: la transazione non deve mai lasciare la base-dati violando i vincoli di integrità su di essa definiti.
- **Isolamento**: in un contesto concorrente, ogni transazione deve essere eseguita come se fosse l'unica operazione in esecuzione.
- **Durabilità (Persistenza)**: i dati modificati da una transazione, nel rispetto delle tre proprietà precedenti, non devono essere persi in caso di problemi di qualsiasi tipo, quali problemi hardware o software al sistema, interruzione di alimentazione, etc.

[5] Definizione tratta da Riferimenti[1 p300]
[6] Riferimenti[15]
[7] Secondo le definizioni date, atomicità e isolamento sono proprietà dell'operazione, mentre consistenza e persistenza sono proprietà dei dati.

Tali proprietà, nella pratica, stabiliscono quale sia il *contratto*, tra l'utente e il DBMS, nel contesto delle basi di dati. Stabiliscono, infatti, che le operazioni interne ad una transazione devono essere eseguite come un'unica operazione indivisibile (atomicità), nel rispetto dei vincoli stabiliti in precedenza sulla base di dati (consistenza), senza l'influenza di altre eventuali transazioni eseguite contemporaneamente (isolamento), e con un risultato duraturo nel tempo (persistenza).

Detto *contratto*, tuttavia, può essere esteso convenientemente anche a contesti differenti da quello per il quale è stato concepito.

Supponiamo, ad esempio, che un commerciante stia trattando contemporaneamente la vendita di un prodotto con due acquirenti. Se non ci fosse atomicità, la vendita potrebbe risultare nella sola consegna del bene o nel solo pagamento dello stesso (con conseguente lite e scioglimento del contratto tra le parti).

Se non ci fosse isolamento, pur essendoci atomicità, potrebbe accadere che mentre il primo acquirente stia per pagare il bene, questo venga dichiarato dal commerciante come disponibile anche al secondo acquirente, pur potendolo ricevere uno solo di loro. Pertanto, il secondo acquirente constaterebbe la mancanza di serietà del commerciante, che promette un bene di cui non dispone. Anche in presenza di un solo acquirente, se ci fosse atomicità, ma non consistenza, potrebbe accadere che questo paghi per un oggetto promesso non più disponibile, avendo il negoziante una visione *inconsistente* delle proprie scorte di magazzino.

Infine, nel caso in cui non ci fosse persistenza, ovvero in caso di commerciante smemorato, potrebbe accadere che, in seguito ad un vuoto di memoria, questi pensi che l'acquirente non abbia pagato l'oggetto, ma che lo abbia rubato.

L'esempio, nella sua banalità, mostra chiaramente come il rispetto delle proprietà ACID sia un requisito applicabile in molti casi reali, anche quando questi non coinvolgano nessun mezzo informatico. Il rispetto di tali proprietà, infatti, assicura l'esecuzione di una transazione da tutta una serie di anomalie che potrebbero altrimenti verificarsi.

In particolare la mancanza di isolamento, in ambienti concorrenti, può produrre le seguenti anomalie[8] (la classificazione si riferisce all'ambito delle basi di dati, ma i concetti espressi sono estendibili facilmente ad altri ambiti):

- **Perdita di update**: se due transazioni tentano di aggiornare contemporaneamente lo stesso dato, l'aggiornamento prodotto da una delle due potrebbe andare perso.
- **Letture sporche**: una transazione potrebbe leggere un dato modificato da una transazione che effettuerà l'abort.
- **Letture inconsistenti**: se la stessa transazione effettua due letture consecutive dello stesso dato, potrebbe ottenere valori diversi, poiché un'altra transazione potrebbe modificarlo nel mezzo.
- **Aggiornamenti fantasma**: una transazione che legga più dati potrebbe erroneamente concludere che questi non rispettino complessivamente un particolare vincolo loro imposto, poiché tale verifica può essere intrapresa solo dopo la lettura di tutti i dati, e un'altra transazione potrebbe modificarne alcuni nel mezzo, seppure nel rispetto di detto vincolo.

Tali anomalie possono, inoltre, facilmente tradursi a loro volta in mancanza di consistenza. Ad esempio, in seguito ad una *perdita di update*, eventuali altri dati modificati dalla stessa transazione potrebbero soddisfare "virtualmente" dei vincoli di integrità, che altrimenti non sarebbero stati soddisfatti.

Si noti che la caratteristica di isolamento, nelle ipotesi di esecuzione seriale delle transazioni, risulta automaticamente soddisfatta, poiché, come già osservato, essa equivale proprio a richiedere che l'esecuzione concorrente delle transazioni sia equivalente ad un'esecuzione seriale.

[8] Esempi concreti in cui si verificano tali anomalie possono essere trovati in Riferimenti[1 p304-306].

1.2 Tecnologie dei DBMS

Prima di introdurre brevemente le tecniche comunemente utilizzate dai DBMS per garantire le proprietà ACID, è necessario distinguere[9] le transazioni in:

- **locali**: coinvolgono un solo *esecutore*, e questo risiede sulla stessa macchina del *richiedente*.
- **remote**: coinvolgono un solo *esecutore*, ma questo risiede su una macchina distinta da quella del *richiedente*.
- **distribuite**: coinvolgono almeno due *esecutori*.[10]

La classificazione data presuppone che ogni operazione eseguita da un singolo esecutore faccia riferimento a dati che siano locali alla *macchina* dell'esecutore stesso. Nell'ambito delle basi di dati l'esecutore corrisponde al DBMS, e le transazioni sono composte da operazioni elementari comunemente denominate query. Una query, solitamente, viene espressa in SQL, che è un linguaggio standard, ma sono possibili anche linguaggi proprietari, o varianti dell'SQL detti *dialetti*.

Per quanto riguarda le ***transazioni locali***, esse costituiscono il caso d'uso più semplice di un DBMS transazionale, essendoci nello specifico due soli attori, l'utente (l'applicazione utente) e il DBMS, rispettivamente nei ruoli di *richiedente* ed *esecutore*.

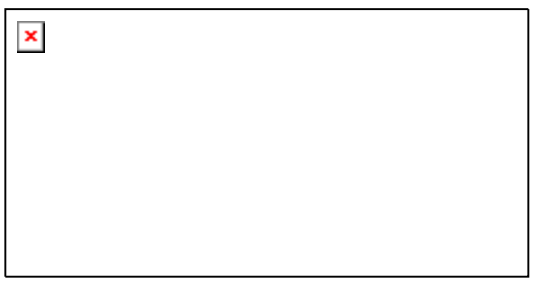

Figura 1 Transazione Locale

[9] Tale classificazione è stata adottata per la prima volta dall'IBM, e successivamente accettata da molti altri costruttori (confronta Riferimenti[**1** p376]). Essa faceva riferimento all'ambito più ristretto dei Data Base, e nel presente testo è stata estesa ad un ambito informatico più generico.

[10] La definizione vale a prescindere dalla collocazione dei due servizi, che potrebbe essere anche sulla stessa macchina. Si richiede solo che essi siano eseguiti in maniera autonoma: ad esempio, sulla stessa macchina, i due servizi potranno essere eseguiti in multitasking.

Ad esempio, l'operazione bancaria di giroconto può essere interpretata come una transazione di questo tipo, nell'ipotesi che i due conti correnti riguardanti l'operazione siano mantenuti sulla stessa base-dati, e nel caso in cui l'operatore interagisca direttamente con questa.

Per garantire le proprietà ACID in relazione a dette transazioni, i DBMS odierni utilizzano svariate tecniche, che seppure nella pratica risultano particolarmente complesse per questioni di efficienza, almeno concettualmente sono molto semplici.

L'atomicità viene garantita marcando i confini delle operazioni della transazione, con *commit* finale, e memorizzando lo stato antecedente al nuovo, in modo tale che, in caso di *abort*, sia possibile ripristinare la situazione precedente. La memorizzazione dello stato precedente avviene in maniera persistente e adottando particolari accorgimenti nelle scritture, affinchè l'atomicità possa essere garantita anche in caso di *failures* di sistema.

La consistenza viene garantita verificando che i vincoli di integrità non siano violati da una nuova operazione, nel qual caso l'esecuzione della transazione viene interrotta (abort), e viene ripristinato lo stato precedente.

L'isolamento viene garantito sfruttando i risultati della *Teoria della Concorrenza*, che nella pratica si traducono nel noto protocollo di Two Phase Locking (2PL)[11], con la sua variante Strict, o nel metodo del Time Stamping (TS)[12]. Quest'ultimo, tuttavia, non viene utilizzato, poiché presenta l'inconveniente del così detto *effetto domino:* in caso di abort di una transazione, tutte le altre transazioni che abbiano effettuato *letture sporche* dovrebbero essere *abortite* a loro volta. I sistemi commerciali definiscono inoltre diversi livelli di isolamento, almeno rispetto alle operazioni di lettura[13]: una transazione può ad esempio accettare *letture sporche*, se vuole incrementare le proprie performance di accesso concorrente.

[11] Viene spiegato esaustivamente in Riferimenti[**1** p309]. Un inconveniente che deriva dall'adozione di tale protocollo è la possibilità che si verifichino *deadlock*. Per ovviare al problema possono essere impiegate delle tecniche di prevenzione e/o di rilevamento ed eliminazione. Queste ultime solitamente usano dei semplici *timeout* che, quando scaduti, producono l'abort di una delle transazioni *bloccate*.
[12] Spiegato in Riferimenti[**1** p313].
[13] Per le operazioni di scrittura non è possibile *rilassare* i requisiti di isolamento per una singola transazione, poiché questo avrebbe ripercussioni anche sulle transazioni che non richiedano alcun rilassamento in lettura.

Infine, la persistenza dei risultati viene garantita impiegando tecniche di backup e più supporti di memorizzazione permanente. Inoltre, per non penalizzare le performance, viene utilizzata la RAM e le scritture vengono effettuate su un più rapido *file di log* sequenziale, piuttosto che direttamente nella loro collocazione definitiva. Evidentemente, affinché ciò sia possibile, vengono utilizzati particolari meccanismi per garantire l'*allineamento* dei dati tra i vari supporti di memorizzazione utilizzati.

Per le ***transazioni remote*** le tecniche dette continuano ad essere tutte valide. In tal caso sorge, tuttavia, un problema: essendo richiedente ed esecutore locati su macchine distinte, è necessario qualche meccanismo per assicurare che il *richiedente* sia messo a conoscenza dell'esito della sua richiesta. Il problema si presenta per due motivi: possibilità di caduta delle *macchine* su cui risiedono il richiedente o l'esecutore; errori di comunicazione e perdita di messaggi. Ad esempio, se il richiedente sospetta che la richiesta non sia pervenuta all'esecutore per qualche ragione, non può reinviarla, poiché non può distinguere tale situazione dalla perdita del messaggio di risposta. In tal caso una rispedizione risulterebbe in due messaggi di richiesta. Inoltre è possibile che il messaggio di richiesta arrivi in ritardo, ad esempio per presenza di congestioni sulla rete di comunicazione.

Per ovviare a questi inconvenienti, vengono impiegati dei protocolli detti *protocolli di completamento*, poiché assicurano il completamento, appunto, della transazione (evento che coincide con la comunicazione dell'esito al *richiedente*).

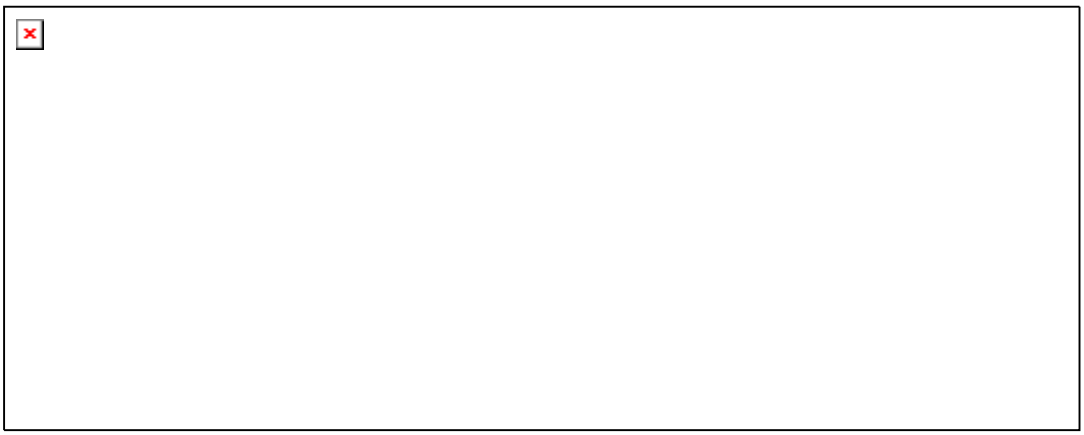

Figura 2 Transazione Remota

Si noti che, per transazioni locali e remote, spesso il non rispetto della proprietà di atomicità, isolamento o persistenza, implica il non rispetto della consistenza. Ad

esempio, ciò accade quando l'atomicità corrisponde ad un vincolo (un insieme di vincoli) di integrità definito su una singola base di dati. Per tali transazioni, tuttavia, essendo i sistemi di controllo e verifica delle varie proprietà separati, la consistenza può continuare ad essere assicurata anche in mancanza delle altre proprietà.

Un esempio di transazione remota è certamente quello della prenotazione del concerto via Web, interpretando come richiedente l'intera applicazione (trascurando la suddivisione in parte client e server) con cui interagisce l'utente, e come esecutore il DBMS della base dati su cui vengono memorizzati i biglietti prenotati. Un altro esempio è quello dell'operazione bancaria di giroconto, assumendo che l'operatore richieda l'operazione attraverso un terminale collegato via rete al server del DBMS.

I problemi di inaffidabilità del mezzo di comunicazione, assieme a possibili problemi delle macchine coinvolte in una transazione, vengono notevolmente amplificati per le ***transazioni distribuite***. Questo è il caso, ad esempio, ancora dell'operazione bancaria di giroconto, ma nell'ipotesi che i due conti corrente interessati dall'operazione siano mantenuti su due basi-dati gestite da DBMS distinti.

Per questo tipo di transazioni, affinché possa essere garantita ancora l'atomicità a dispetto di *failures* di sistema e/o di comunicazione, devono essere utilizzati dei meccanismi particolari. Essi sono detti *Protocolli di Commit*, poiché consentono ai partecipanti della transazione di raggiungere una decisione comune di commit (o di abort). In particolare viene utilizzato il protocollo *Two Phase Commit* (2PC)[14], eventualmente nelle sue varianti a tre e a quattro fasi, che, pur presentando alcune limitazioni, risulta nella pratica molto efficiente, e viene comunemente adottato. Tali protocolli hanno necessità di mantenere alcune ulteriori informazioni nel *file di log*, affinché, in caso di *failures*, sia possibile completare la sequenza dei messaggi prevista, nel rispetto della proprietà che vogliono garantire.

[14] Spiegato esaustivamente in Riferimenti[1 p384].

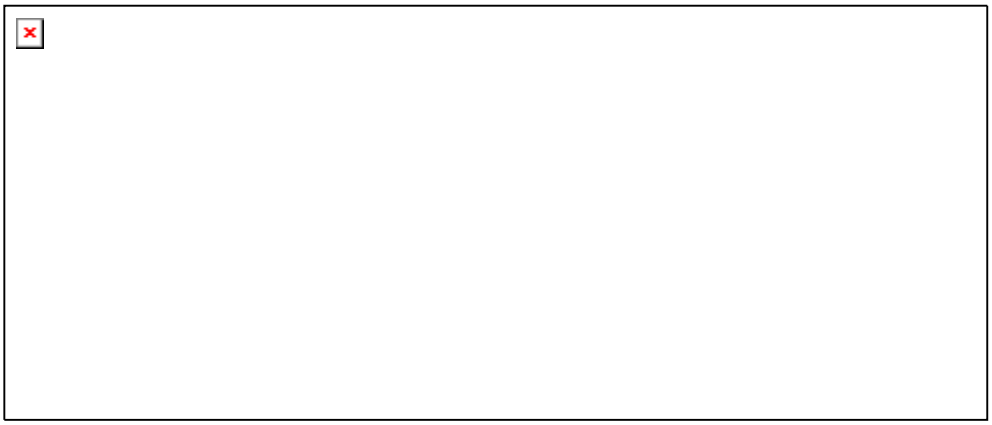

Figura 3 Transazione Distribuita

Le tecniche per garantire la *consistenza*, invece, non variano, essendo questa una caratteristica locale per ogni singola base di dati. Si noti, infatti, che la consistenza, così come precedentemente definito nelle proprietà ACID, fa riferimento esplicitamente ai dati e indirettamente alla transazione. Tuttavia, mentre per transazioni locali e remote esiste una netta distinzione tra i meccanismi che garantiscono atomicità e consistenza, per le *transazioni distribuite*, a volte, l'atomicità risulta essere l'unico "mezzo" a disposizione per imporre dei vincoli di integrità distribuiti, che non potrebbero essere altrimenti verificati. Così, considerando la base-dati unione delle singole basi di dati coinvolte nella transazione, il termine "consistenza", nel senso di *consistenza globale*, viene talvolta utilizzato in luogo di "atomicità".

La Persistenza, assieme alle performance di accesso concorrente, può essere ottenuta anch'essa ancora con le stesse tecniche adottate per le transazioni locali e remote, in quanto la persistenza locale su ogni *esecutore* garantisce automaticamente la persistenza globale a livello di transazione.

Lo stesso discorso non vale invece, in generale, per l'isolamento: l'isolamento delle operazioni su ogni singola macchina è solo condizione necessaria per l'isolamento delle transazioni distribuite. Fortunatamente, almeno nel caso del metodo utilizzato nella pratica, ovvero il 2PL, la teoria della concorrenza fornisce una proprietà[15] affinché detta condizione risulti pure sufficiente:

> *Se ogni esecutore utilizza il metodo di locking a due fasi e svolge l'azione di commit in un istante in cui tutte le sotto-transazioni ai vari nodi*

[15] Confronta Riferimenti[1 p380].

detengono tutte le risorse, le transazioni saranno isolate (schedule serializzabili).

Tale proprietà risulta in pratica automaticamente soddisfatta utilizzando il 2PL per l'isolamento e il 2PC per l'atomicità. L'adozione del 2PL, come per le transazioni locali e remote, può ancora comportare la possibilità di *deadlock*, nell'ambito specifico detti *deadlock distribuiti*, e questi possono ancora essere risolti mediante la "tecnica dei timeout".

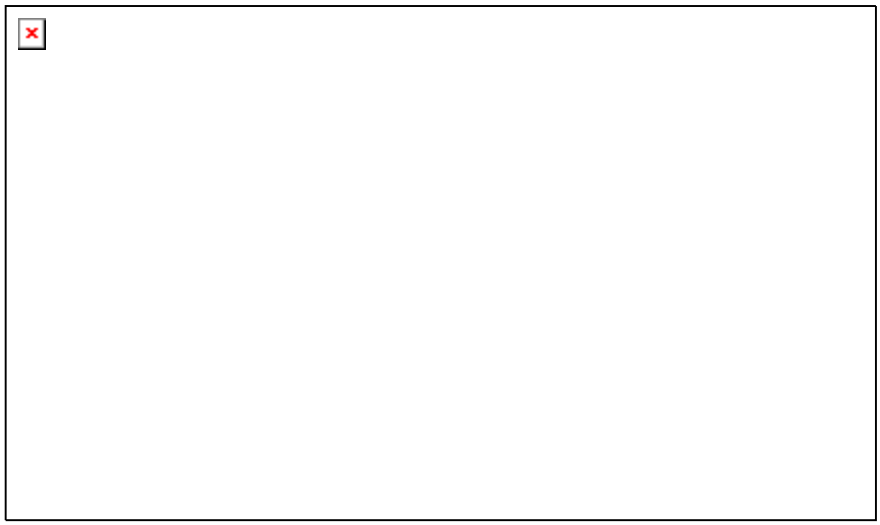

Figura 4 Esempio di *deadlock distribuito*

In Figura 4 viene mostrato un semplice esempio di *deadlock distribuito*. Sia Richiedente1 che Richiedente2 necessitano delle stesse risorse, denominate Resource1 e Resource2. Osservando la sequenza dei messaggi, mostrata nel Collaboration Diagram, con cui le risorse vengono richieste, si può capire facilmente come si determini una situazione di stallo, poiché nessuno dei due richiedenti potrà continuare la propria esecuzione. Fortunatamente, tale situazione può essere facilmente verificata mediante un timer opportunamente settato, scaduto il quale potrà essere *abortita* una delle due transazioni per consentire all'altra di proseguire. Ovviamente, se le due risorse in questione appartenessero allo stesso esecutore, e se fosse nota in anticipo la composizione delle transazioni (ovvero le risorse richieste da ognuna), il problema potrebbe essere evitato del tutto, semplicemente imponendo che le risorse siano assegnate sempre nello stesso ordine. Negli altri casi, la "tecnica dei timeout" risulta, comunque, molto efficiente, purchè i timer presso i vari servizi esecutori scadano in

tempi differenti (condizione praticamente sempre verificata), affinché si eviti che le transazioni in stallo siano *uccise* contemporaneamente, e dunque che il problema possa subito ripresentarsi.

1.2.1 Two Phase Commit e transazioni distribuite

Essendo il protocollo 2PC il punto di partenza per qualsiasi discussione che riguardi modelli di coordinamento per transazioni distribuite, lo descriveremo approfonditamente. Il 2PC altro non è che un Protocollo di Commit, ovvero avente l'obiettivo di raggiungere una decisione consistente tra tutti i partecipanti ad una transazione. A tal proposito vengono scambiati una serie di messaggi con una precisa semantica tra il *richiedente* della transazione e gli *esecutori*. Il richiedente avvia un agente software detto Transaction Manager (TM), il quale si preoccupa dell'esecuzione del protocollo con le controparti avviate dai singoli esecutori, dette Resource Manager (RM). Si sviluppa in due fasi:

- Nella prima fase il TM richiede a tutti gli RM_i di prepararsi per l'esecuzione della loro operazione, ma senza eseguirla realmente, e raccoglie da questi i messaggi di risposta.
- Nella seconda fase, il TM analizza tutte le risposte, e nel caso in cui tutti gli RM_i siano pronti per l'esecuzione, invia loro una decisione globale di *commit*, con cui comunica di eseguire realmente i cambiamenti. Nel caso in cui un solo RM_i abbia dichiarato di non essere pronto, oppure non abbia risposto entro un tempo massimo, il TM comunica a tutti gli RM_i una decisione globale di *abort*. In caso di commit, successivamente il TM attende un messaggio di riscontro dagli RM_i, per accertarsi che tutto sia andato come previsto.

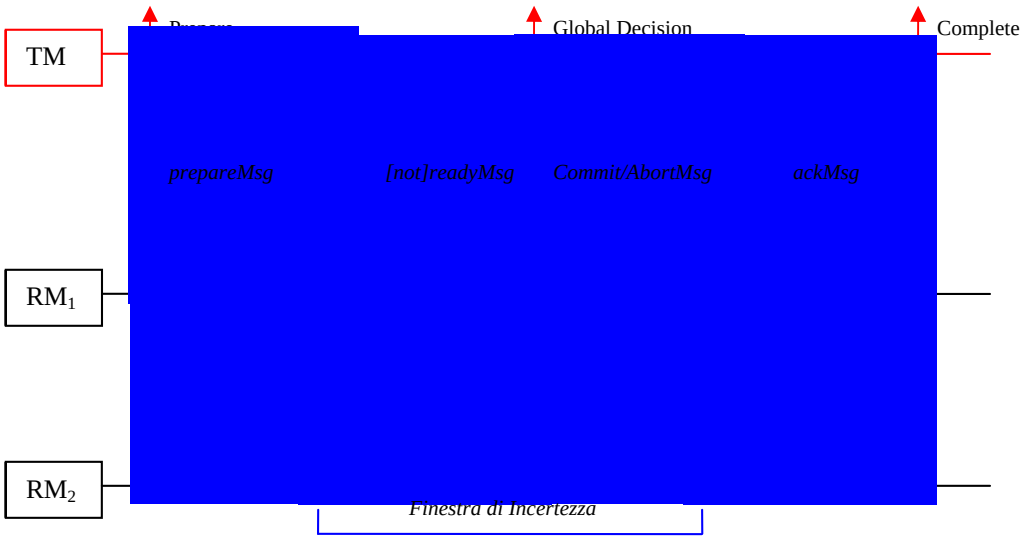

Figura 5 Protocollo 2PC

In questo modo, purchè tutti i partecipanti eseguano nella seconda fase ciò che hanno promesso nella prima, la transazione risulterà atomica.

Il lettore noti la duplice garanzia fornita dal protocollo: da una parte assicura l'atomicità della transazione, dall'altra assicura ai partecipanti che, una volta terminato il protocollo, essi non avranno più a che fare con detta transazione, consentendo loro di poter raggiungere uno stato stabile e consistente. Questa seconda garanzia, come si intuisce, è un presupposto all'isolamento. Si immagini, infatti, cosa potrebbe accadere se, in seguito ad una transazione che termini con Commit, la mancata esecuzione di una delle operazioni componenti dovesse comportare l'annullamento di tutte le altre operazioni eseguite correttamente: qualsiasi sia il protocollo utilizzato per l'isolamento, questa caratteristica non potrebbe più essere garantita.

Si vuole ora sottolineare come il 2PC nulla imponga sull'isolamento, poiché ha come unico obiettivo quello di permettere il raggiungimento di un risultato atomico. Tuttavia, la maggior parte dei sistemi distribuiti lo utilizzano congiuntamente allo Strict 2PL poiché, come osservato nel capitolo precedente, nella pratica non vi sono tecniche alternative, e questo potrebbe indurre a qualche confusione.

La confusione, inoltre, è incentivata dal fatto che l'utilizzo congiunto dei due noti protocolli avvenga nonostante gli innumerevoli problemi, altrettanto famosi, che questo comporta. In Figura 5 è stata mostrata la cosiddetta *finestra di incertezza*: l'intervallo di

tempo che passa da quando un RM_i comunica di essere *pronto* per il commit[16], a quando riceverà la decisione globale del TM.

In tale intervallo l'RM_i deve necessariamente rimanere in attesa della risposta dal TM poiché, non potendo in nessun modo distinguere la perdita del suo messaggio di *pronto* dalla perdita del messaggio contenente la decisione del TM, rischierebbe (nel secondo caso) di prendere una decisione locale differente da quella del TM, violando così lo scopo stesso del protocollo.

Il problema, nel caso in cui l'isolamento sia gestito mediante lo Strict 2PL, viene ulteriormente amplificato dal fatto che le risorse bloccate debbano rimanere tali per tutto il tempo della finestra di incertezza, e potenzialmente, in caso di caduta del TM[17], per un tempo indefinito. I sistemi commerciali odierni, sia per minimizzare la probabilità di inconvenienti, sia per limitare la degradazione delle performance "concorrenti" conseguente al *locking* delle risorse, utilizzano l'unica strategia possibile: cercano di ridurre al minimo la durata della finestra di incertezza. A tal proposito, dopo l'invio dei messaggi di *prepare*, il TM setta opportunamente un timer. Se non dovesse ricevere qualche risposta entro il tempo prefissato, deciderà per un Abort globale, affinchè almeno gli RM_i senza problemi possano liberare le proprie risorse. Il TM dovrebbe, comunque, rimanere attivo per gli eventuali RM_i dai quali non aveva ottenuto risposte in tempo poiché, quando questi potranno nuovamente comunicare, vorranno certamente conoscere l'esito della transazione.

In maniera analoga il TM, dopo aver inviato una decisione globale, dovrà rimanere in attesa di tutti i messaggi di riscontro, poiché non potrà distinguere la perdita di uno di questi dalla perdita del messaggio in cui comunicava la propria decisione globale al corrispondente RM_i.[18] Se, infatti, fosse la sua decisione ad essersi persa, e se non aspettasse il messaggio di riscontro dall'RM_i , quest'ultimo rimarrebbe bloccato indefinitivamente per quanto detto in precedenza. Pertanto, all'inizio della seconda fase, il TM solitamente setta un ulteriore timer, e nel caso non riceva risposte entro il tempo prefissato, ripete la seconda fase. Alternativamente, la ripetizione della seconda fase del protocollo potrebbe essere richiesta direttamente da un RM_i con un opportuno

[16] Più precisamente, da quando l'RM_i memorizza il record di Prepared.
[17] La caduta del TM nella *finestra di incertezza* produrrebbe un blocco permanente delle risorse dell'RM. Per rispondere a questo problema sono nati i protocolli di commit a 3 e a 4 fasi Riferimenti[1 p391].
[18] Per tale motivo sono stati sviluppati dei particolari protocolli di recovery.

messaggio, senza aspettare nessuna scadenza: ad esempio, un RM_i *caduto* può richiedere l'esito della transazione non appena ripristinato.

In Figura 5 vengono mostrati anche i records che devono essere memorizzati persistentemente dal TM e dagli RM_i affinché l'atomicità sia garantita anche in caso di *failures*. Il TM dovrà memorizzare:

- il record di Prepare, contenente anche gli identificativi degli RM_i, prima di avviare la fase omonima;
- il record contenente la decisione globale;
- il record di Complete, alla fine del protocollo

Un RM_i dovrà memorizzare, dopo aver raggiunto uno stato stabile, il record di Ready, assieme all'identificativo del TM, e il record contente la decisione finale di Commit o di Abort. Solo dopo aver memorizzato stabilmente il record di Ready, il partecipante potrà inviare il messaggio di Prepared, e solo dopo aver memorizzato stabilmente il record di GlobalDecision, il TM potrà inviare la decisione globale.

1.2.1.1 Ottimizzazioni

Un'ottimizzazione del protocollo consiste nell'adozione del modello dell' *Abort Presunto*[19], che consente di minimizzare la quantità di informazioni da mantenere persistentemente. Esso si basa, infatti, sul comportamento di default espresso dalla regola[20]:

> *Ad ogni richiesta di recovery, da parte di un partecipante in dubbio su una transazione della quale il coordinatore non abbia informazioni, quest'ultimo restituirà la decisione di Abort.*

In base a tale regola, è sufficiente che il TM memorizzi solo la decisione globale nel caso in cui questa sia di Commit[21], e che un RM_i memorizzi solo i record di Ready e di

[19] Tale modello è descritto in Riferimenti[1 p390]
[20] Confronta Riferimenti[1 p390]
[21] Il record di Complete non è essenziale, infatti, nel caso di caduta del TM, richiede semplicemente la ripetizione della seconda fase, ma ciò solo in caso di eventuale richiesta di un RMi.

Commit. La maggior parte dei sistemi scrivono anche gli altri record, ma con primitive asincrone, in maniera da velocizzare le operazioni.[22]

Si noti che, utilizzando tale modello, nel caso in cui il TM cada prima di aver preso la decisione globale, dovranno essere gli RM_i ad interrogarlo con una richiesta di *remote-recovery* nel caso siano rimasti bloccati a seguito della prima fase: la risposta del TM dovrà essere quella di default, ovvero di Abort globale, poiché esso non troverà il record di Prepare. Utilizzando il protocollo non ottimizzato, invece, la prima fase potrebbe essere ripetuta dal TM in seguito alla sua *ripresa*.

Un'ulteriore ottimizzazione prevede che un partecipante, nel caso in cui scopra durante l'esecuzione che non ha effettuato alcuna operazione di scrittura, possa inviare un messaggio di *read-only* in risposta al messaggio di *prepare*. In tal caso, infatti, il partecipante potrà essere ignorato nella seconda fase del protocollo.

1.2.1.2 Standard X-Open DTP

I DBMS commerciali, oltre ad implementare interfacce proprietarie del 2PC per permettere la realizzazione di applicazioni relazionali omogenee, solitamente implementano anche le interfacce previste dall'architettura[23] standard X-Open DTP (Distributed Transaction Processing), affinché sia possibile realizzare applicazioni relazionali eterogenee, ovvero distribuite su DBMS di venditori differenti.

Il protocollo X-Open altro non è che una formalizzazione del protocollo 2PC appena descritto. Esso definisce l'interfaccia tra client e TM, chiamata TM-interface, e l'interfaccia tra TM ed RM, chiamata XA-interface. Utilizza le ottimizzazioni di *Abort Presunto* e di *read-only*, e, come caratteristica aggiuntiva a quelle precedentemente descritte, prevede la possibilità di *decisioni euristiche*. Quest'ultima caratteristica è stata introdotta affinché, in presenza di guasti, sia possibile continuare l'evoluzione della transazione sotto il controllo di un operatore. Tuttavia, la sua introduzione può

[22] I records obbligatori vengono scritti, invece, in maniera *sincrona*, ovvero con una primitiva di *force*.
[23] L'architettura X-Open DTP è descritta sommariamente anche in Riferimenti[1 p394].

compromettere l'atomicità, e il protocollo garantisce che tale eventualità sia, in ogni caso, segnalata ai processi client.

Ad esempio, quando un RM sia bloccato in seguito alla caduta del TM, un operatore può imporre una decisione euristica (solitamente di abort) per liberare risorse. Una volta ripristinato il TM, questo guiderà la procedura di ripristino, che avverrà come segue:

- Il TM contatta l'RM, inviando un messaggio di *xa_recover*.
- L'RM consulta il suo log e indica le transazioni:
  - in dubbio: sono le transazioni rimaste appese in seguito all'invio di un messaggio di Prepared;
  - portate a termine con un commit euristico;
  - portate a termine con un abort euristico.
- Il TM comunica l'esito effettivo delle transazioni in dubbio, e verifica eventuali decisioni euristiche discordanti con la propria decisione. In quest'ultimo caso notifica il client dell'inconsistenza. In seguito invia un messaggio di *xa_forget* all'RM con cui gli indica che può dimenticare le transazioni terminate con decisioni euristiche, poiché queste devono essere ricordate fino alla ricezione di tale messaggio, proprio per garantire la segnalazione di risultati inconsistenti.

1.3 Analisi di casi d'uso per Transazioni Business

Cosa si intende esattamente per Transazione Business? Quali caratteristiche deve soddisfare questo tipo di transazioni? Sono delle transazioni ACID?

Per rispondere a queste domande, analizzeremo possibili scenari che descrivono interazioni tra servizi web che necessitano di coordinamento transazionale. Più specificatamente, valuteremo gli scenari raccolti in "*Use Cases for the Business Transaction Protocol*"[24], ovvero quegli stessi scenari dai quali sono partiti gli autori di BTP[25], una delle prime Specifiche sull'argomento, essendo questi particolarmente significativi, e largamente utilizzati anche in altri documenti sull'argomento e nelle Specifiche nate in seguito. Essi sono stati raccolti dal comitato per lo sviluppo di BTP, e pertanto, nella descrizione originale, fanno talvolta riferimento, seppure implicitamente, alla particolare soluzione da loro proposta. Nel seguito riproporremo detti scenari in forma abbreviata, cercando di *filtrarli* dalla soluzione in essi *inglobata*: cercheremo, cioè, di isolare il "cosa", per poi valutare separatamente il "come".

Per tutti gli scenari, inoltre, cercheremo di individuare i requisiti osservandoli separatamente dal punto di vista dei due *ruoli* coinvolti: questo ci aiuterà a capire come, nelle Transazioni Business, spesso siano richiesti requisiti tra loro in conflitto.

Nel seguito di questo testo utilizzeremo i termini "*transazione business*", "*attività business*", o più semplicemente "transazione" (quando il suo tipo risulti chiaro dal contesto), come sinonimi, per indicare interazioni multi dominio di tipo transazionale. Essi saranno utilizzati, nella valutazione degli scenari, come termini generici e imprecisati, indicanti semplicemente un insieme di operazioni, o attività, tra loro correlate in qualche maniera. Il nostro scopo, a breve termine, sarà proprio quello di fornirne un significato preciso.

[24] Riferimenti[2].
[25] *Business Transaction Protocol* è il risultato della collaborazione di un vasto gruppo di produttori organizzati in un comitato tecnico all'interno dell' OASIS. E' specificato in Riferimenti[3].

Utilizzeremo ancora il termine *operazione transazionale*, o più brevemente *operazione*, per indicare una singola operazione componente una transazione business, seppure ora non si parli più di *query*, bensì di "invocazione di metodi".

Assumeremo inoltre che, per garantire l'isolamento, sia necessaria qualche forma di *locking*, così come diffusamente accettato[26].

1.3.1 Arranging a Night-Out[27]

Supponiamo di trovarci in un futuro non lontano, quando molti servizi business saranno ormai disponibili anche come servizi Web. Stanchi dell'ultima settimana di lavoro, vorremmo organizzare la serata con la nostra consorte per una cena nel suo ristorante preferito, seguita da uno spettacolo a teatro. Così, prendiamo il nostro fedele PDA, e facciamo un giro per il Web. Purtroppo scopriamo che il teatro più vicino non effettua spettacoli utili, e che pertanto dovremmo spostarci fuori città. Non vogliamo utilizzare la nostra auto, e, consapevoli del tardo orario a cui finirà lo spettacolo, decidiamo di pernottare fuori. In definitiva vorremmo effettuare la prenotazione di: (t1) un taxi, (t2) un tavolo al ristorante, (t3) due posti a teatro, (t4) una camera di albergo.

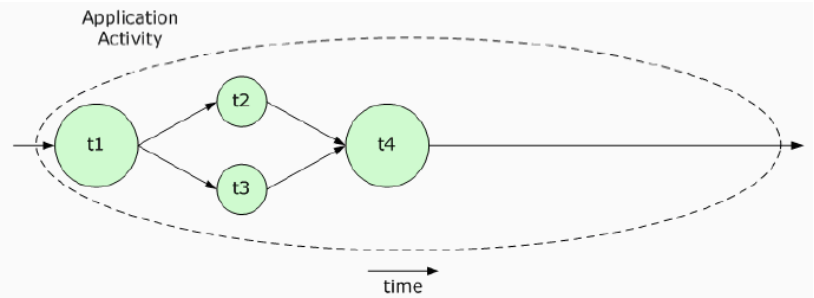

Figura 6 Flusso delle attività

Tuttavia, nel caso in cui una delle prenotazioni tra ristorante, teatro o albergo dovesse fallire, ci accontenteremmo di un posto al cinema e una cena consegnata a domicilio,

[26] Ricordando quanto detto nel capitolo "1.2 Tecnologie dei DBMS" relativamente alle tecniche per garantire l'isolamento, l'unico protocollo utilizzabile nella pratica consiste nel protocollo Strict 2PL.
[27] Caso d'uso trattato anche in Riferimenti[31, 10, 28, 29]

che vorremmo prenotare conservando la prenotazione per il taxi già ottenuta in precedenza mediante t1.

Valutazione

Come nella descrizione originale, anche nella versione qui riproposta non si fa alcun riferimento ad un servizio di credito *on-line*, e pertanto potrebbe sembrare uno scenario poco realistico. Tuttavia, è possibile interpretare le varie prenotazioni effettuate dal cliente come sostitutive delle classiche prenotazioni telefoniche. E' noto, infatti, che queste vadano intese come *prenotazioni di cortesia*, poiché non pongono nessun vincolo stringente alle parti.

Dal punto di vista del *richiedente*, la composizione della transazione business deve poter essere determinata *dinamicamente*, nel senso che nuove operazioni devono poter essere aggiunte in funzione delle risposte alle operazioni correnti: nel caso specifico, le prenotazioni di cinema e "cena a casa" sono soggette alle risposte ottenute mediante t2, t3 e t4. Inoltre, operazioni correnti devono poter essere *annullate* selettivamente in caso di indisponibilità di qualche servizio esecutore: nel caso in cui una tra le prenotazioni corrispondenti a t2, t3 e t4 non sia possibile, le rimanenti due prenotazioni, tra le tre, devono essere annullate.

Si noti, in Figura 6, che le due operazioni t2 e t3 sono state poste tra loro in parallelo, mentre sono in serie rispetto a t1, e t4. Tale disposizione rispecchia semplicemente la dinamica di composizione operata dal cliente, peraltro del tutto arbitraria, per la quale viene avviata prima t1, in funzione del risultato di questa vengono successivamente avviate t2 e t3, e così via.

Essendo le prenotazioni non vincolanti, la caratteristica di atomicità non è essenziale, seppure per l'utente possa essere preferibile che essa sia rispettata per l'insieme composto dalle prenotazioni confermate: t1, t2, t3, e t4, oppure, eventualmente, t1, "cinema" e "cena a casa". Tuttavia, anche nel caso in cui l'atomicità non sia rispettata, l'utente vorrà certamente essere messo al corrente di eventuali prenotazioni non riuscite poiché, ad esempio, non vorrà rischiare di prendere il taxi per arrivare a teatro, senza avere un posto riservato per quest'ultimo. Evidentemente, la possibilità di risultato non atomico costringerebbe comunque l'utente ad operazioni aggiuntive. Ad esempio,

supponendo che dopo la conferma della prima transazione l'albergo non abbia confermato la prenotazione, ma che lo abbiano fatto gli altri servizi, l'utente potrebbe cercare un altro albergo in zona. Ma se anche gli altri alberghi registrassero il "tutto pieno", l'utente dovrebbe certamente disdire le prenotazioni già confermate relative a taxi, ristorante e teatro.

Prima di effettuare le prenotazioni, è lecito supporre che l'utente richieda i prezzi e controlli la disponibilità dei servizi prenotati, e sulla base di questi decida se prenotare o meno. Pertanto, anche per quanto riguarda l'isolamento, l'utente vorrebbe certamente prendere decisioni su dati validi. Del resto, almeno per i costi, il "contratto di prenotazione" tra utente e servizio dovrà necessariamente prevedere la consapevolezza, per il cliente, del prezzo di acquisto.

Dal punto di vista dei proprietari dei servizi *esecutori*, è importante garantire la massima disponibilità, e mantenere il controllo delle proprie risorse, pur andando incontro alle esigenze del cliente. Per assicurarsi il controllo delle risorse, un servizio esecutore dovrebbe poter revocare la propria partecipazione ad una transazione business in qualsiasi momento prima che questa sia conclusa. Tale possibilità, inoltre, sarebbe particolarmente utile nel caso in cui il servizio abbia qualche problema per il quale non possa più continuare nella transazione. Per incrementare la propria disponibilità, e dunque il numero di accessi concorrenti, essendo possibile la *composizione dinamica* da parte del cliente, e potendo questa durare un tempo anche molto lungo, un servizio esecutore potrebbe voler utilizzare una o più delle seguenti strategie:

- *Overbooking*: assegnamento delle risorse, prima della conferma finale, in maniera non esclusiva.
- Minimizzazione del periodo di *locking* delle risorse durante l'esecuzione della transazione business.

Per soddisfare le esigenze del cliente, un servizio potrebbe assicurare il rispetto dei vincoli necessari per garantire l'atomicità del risultato, oppure potrebbe garantire almeno che il cliente sia avvisato, anche in presenza di *failures*, nel caso in cui tali vincoli non siano rispettati,. In entrambi i casi, il servizio esecutore dovrà sicuramente mantenere informazioni persistentemente circa la transazione business, a seconda del particolare protocollo di coordinamento utilizzato.

Inoltre, per poter rispettare il "contratto di prenotazione", vorrà mantenere persistentemente anche i risultati delle prenotazioni, e vorrà che tali memorizzazioni avvengano in maniera consistente.

Per qualche servizio, tuttavia, rispettare i vincoli per fornire garanzie circa l'atomicità, potrebbe non essere accettabile. Ad esempio, il servizio per il taxi potrebbe utilizzare un operatore umano che, quando sia confermata una prenotazione, invii direttamente il taxi all'indirizzo richiesto, senza memorizzare il risultato finale. In questa ipotesi, seppure poco concreta, poiché non permetterebbe la posticipazione dell'utilizzo del taxi, il servizio potrebbe non voler memorizzare alcuna informazione, in maniera da economizzare la propria infrastruttura hardware. In tal caso, evidentemente, l'atomicità non potrebbe essere garantita in presenza di *failures*, ma, del resto, nel contesto in esame le prenotazioni sono solo di *cortesia*. Analogamente, qualche esecutore potrebbe garantire solo la persistenza del risultato finale (non anche l'atomicità), ed eventualmente soltanto per un periodo limitato di tempo: ad esempio, il servizio per la prenotazione del teatro potrebbe richiedere che il cliente passi a pagare e ritirare il biglietto almeno 24 ore prima dell'inizio dello spettacolo, e così manterrebbe la prenotazione solo fino alla scadenza detta.

Sia per il cliente che per i servizi esecutori, le operazioni dovrebbero avvenire in un contesto sicuro. Il livello di sicurezza richiesto dal cliente, tuttavia, potrebbe essere differente da quello offerto dal particolare servizio utilizzato, e servizi diversi potrebbero utilizzare diversi protocolli per la sicurezza. Nello scenario, per il cliente sarà importante la propria privacy, mentre per i servizi esecutori sarà essenziale che le richieste provengano da utenti legittimi. Ad esempio, il ristorante non vorrà mantenere tavoli riservati inutilmente, e per questo motivo potrebbe utilizzare politiche differenti a seconda che le prenotazioni provengano da clienti di fiducia o da clienti occasionali. Per questi ultimi, potrebbe richiedere una caparra di anticipo.

Chiaramente, nel caso in cui un servizio richieda un pagamento anticipato, anche solo in forma di caparra, questo potrà essere effettuato "di persona", deteriorando così i benefici per il cliente, che vorrebbe effettuare tutte le operazioni da casa, oppure potrà avvenire ancora on-line, ad esempio mediante carta di credito, o con lo stessa tecnica che discuteremo nello scenario "1.3.6 Micro-paying". Tuttavia, con l'introduzione di

meccanismi di pagamento on-line, sia la sicurezza che la garanzia di atomicità sarebbero per il cliente requisiti inderogabili.

1.3.2 Home entertainement system[28]

E' arrivato il momento di rinnovare il nostro, ormai obsoleto, sistema di intrattenimento domestico. Vogliamo acquistare dei nuovi TV, DVD, hi-fi, e video registratore, comodamente seduti a casa, utilizzando il Web, per ottenere i prezzi migliori e per disporre di una maggiore scelta, in quanto, per non suscitare l'ira di nostra moglie, dovremo comprare tutti i componenti con una spesa contenuta, e nel rispetto dell'arredamento del salotto.

Per farlo utilizziamo un'applicazione che ci permette di contattare i servizi Web dei venditori di interesse. Durante la visione dei siti per la TV, prenoteremo momentaneamente, con riserva, un certo numero di differenti televisori, diciamolo insieme A, che corrispondono ai nostri requisiti (attività t1). Prima che t1 sia terminata, sceglieremo un sottoinsieme B[29] di A, al quale siamo interessati per l'acquisto, e concederemo il nostro numero di conto bancario affinché il venditore possa controllare che abbiamo sufficiente disponibilità economica.

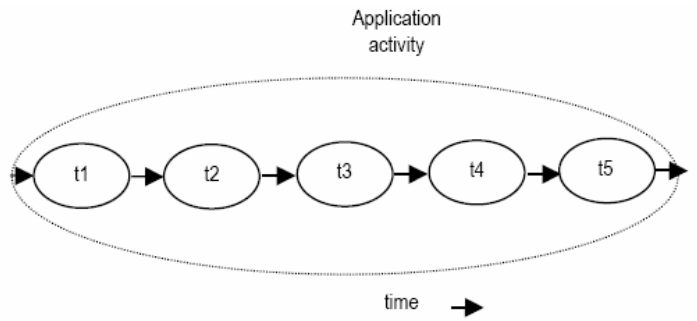

Figura 7 Flusso delle attività

[28] Caso d'uso trattato anche in Riferimenti[10, 29]
[29] Normalmente l'insieme B sarà costituito da un singolo televisore.

Quando t1 sarà terminata, tutte le riserve ottenute, oltre quelle su B, potranno essere rilasciate, mentre quelle su B dovranno essere mantenute. Nelle successive attività t2, t3, e t4, effettueremo operazioni simili, ma su oggetti differenti.

Quando inizierà t5, avremo una lista di oggetti riservati a nostro nome, e potremo decidere se confermare l'intero acquisto.

Valutazione

A differenza dello scenario precedente, in questo si fa esplicitamente riferimento ad un servizio di credito on-line. Tuttavia, esso non viene utilizzato per un pagamento, bensì per assicurare il venditore sulla disponibilità economica del cliente. Pertanto, l'atomicità può ancora essere un requisito rilassabile.

In realtà, è facile verificare che valgono anche tutte le altre considerazioni già fatte per lo scenario precedente. In questo, tuttavia, risulta maggiormente evidente un problema che consegue dall'adozione del modello della Composizione Dinamica: la liberazione anticipata delle risorse riservate, e non più necessarie, avviene ad opera del cliente, sicchè le performance di concorrenza dei servizi sarebbero soggette alla sua "educazione". Tale caratteristica non è certamente desiderabile per i servizi esecutori, che vorrebbero invece esaudire il massimo numero di richieste per incrementare i profitti. Di conseguenza, la possibilità di revoca della partecipazione in qualsiasi momento prima della conclusione della transazione, e l'adozione di tecniche per incrementare la Disponibilità, risultano ora requisiti maggiormente importanti.

Strettamente legato alla possibilità di Composizione Dinamica del cliente, introduciamo ora un ulteriore requisito. Poiché è sempre possibile che l'applicazione utilizzata dal cliente abbia qualche problema, è desiderabile per un servizio esecutore poter stabilire se una prolungata attesa prima della *conferma*, sia conseguente ad una maggiore durata della composizione del cliente, piuttosto che alla presenza di eventuali problemi. Allo scopo sarebbero molto utili meccanismi standard che consentano ad un servizio esecutore di interrogare il richiedente sul suo stato corrente, e/o meccanismi che consentano al richiedente di indicare anticipatamente una stima della durata delle operazioni di composizione, in modo tale che ogni servizio esecutore possa capire quando sia il momento di supporre circa l'esistenza di eventuali problemi.

E' doveroso sottolineare che, a differenza dello scenario precedente, le "prenotazioni" on-line riguardano ora l'acquisto di beni materiali: potrebbero dunque ancora essere intese come di *cortesia* (non vincolanti), ma sarebbe più appropriato inquadrarle nell'ambito delle vendite per corrispondenza. Questo tipo di azioni, almeno nella maggior parte dei paesi occidentali, sono soggette per legge al *diritto di recesso* del cliente, che può essere esercitato anche dopo il ritiro della merce. Pertanto, difficilmente il cliente accetterebbe richieste di pagamento anticipato. Inoltre, è lecito supporre che la maggior parte dei servizi Web che offrono questo tipo di servizi, mettano a disposizione delle procedure particolari, ancora esposte via Web, che consentono l'annullamento di un precedente ordinativo, prima del ritiro dei corrispondenti beni. In tal caso, la garanzia atomicità, o la garanzia di notifica per risultati non atomici, sono effettivamente requisiti di secondo piano, mentre non lo è il requisito sulla sicurezza.

1.3.3 Arranging a meeting[30]

Consideriamo un'applicazione per determinare la data di incontro tra un gruppo di persone. Ognuna di queste possiede un'agenda personale, che è esposta come servizio Web, e ogni slot (giornata) in tali agende è riservabile separatamente. L'applicazione inizia comunicando ad ogni membro del gruppo la necessità di un prossimo incontro, e successivamente raccoglie da ognuno un insieme di date preferibili. Una volta raccolte, le informazioni vengono analizzate per determinare l'insieme dei giorni accettabili per l'incontro. L'insieme così determinato viene comunicato a tutti, affinché questi possano specificare ulteriori preferenze. Il processo continua fino a quando l'insieme non si riduca ad un singolo giorno, quello dell'incontro. Per ridurre la quantità di lavoro che dovrebbe essere intrapreso dall'applicazione in caso di *failures*, e per gestire gli accessi concorrenti, è desiderabile che il software di supporto assicuri che ogni singolo *round* di questo protocollo sia eseguito come una transazione, ovvero nel rispetto delle proprietà di atomicità e isolamento. Inoltre, per non limitare l'accesso concorrente agli slots delle agende personali, che potrebbero essere utilizzate contemporaneamente da altre

[30] Trattato anche in Riferimenti[29]

applicazioni, si vorrebbe che solo le riserve sul nuovo insieme, di volta in volta calcolato, siano mantenute durante il *round* successivo.

Valutazione

In questo scenario il *richiedente*, seppure non esplicitamente definito, coincide con l'utente che richiede l'incontro (o con la sua applicazione). Le operazioni transazionali coincidono, invece, con le richieste di riserva degli slots, che rappresentano le giornate sulle singole agende gestite dai servizi esecutori.

L'applicazione client, durante la sua esecuzione, ha necessità di *cancellazioni* dinamiche: l'insieme dei giorni di volta in volta scartati.

Come evidenziato dall'autore dello scenario, si richiede l'atomicità di ogni singolo *round*, per minimizzare le azioni applicative in caso di *failures*. Pertanto, seppure non fosse garantita l'atomicità, dovrà essere certamente garantita la notifica di risultati non atomici. In ogni caso, la possibilità di rilassare l'atomicità, dipende strettamente dal fatto che l'applicazione debba garantire necessariamente la partecipazione di tutti gli individui contattati o meno.

L'isolamento è certamente necessario, affinché l'applicazione client possa prendere decisioni su dati validi. Lo stesso autore sottolinea che, per incrementare le performance di accesso concorrente, sarebbe desiderabile, in ogni round, liberare gli slots riservati e non più necessari. Anche in questo caso, la liberazione anticipata delle risorse (i giorni sulle agende) è dunque soggetta all' "educazione" del cliente. Tuttavia, nello specifico, il problema risulta meno significativo, in quanto lo scenario lascia presupporre che le operazioni avvengano in un contesto di fiducia, e così, la possibilità di revoca della partecipazione potrebbe non essere necessaria.

Ovviamente, tutte le operazioni dovrebbero avvenire in un ambiente sicuro, e ogni esecutore vorrebbe agire nel rispetto della Persistenza e della Consistenza dei dati.

1.3.4 Manifacturer-Supplier-Shipper[31]

Un'industria manifatturiera, per effettuare gli ordinativi di materia prima, utilizza il servizio Web del proprio fornitore di fiducia. Prima di confermare ogni ordinativo, si rivolge ad una compagnia di trasporto, anch'essa esposta come servizio Web, per assicurarsi che la merce possa essere consegnata entro il giorno stabilito. Solo in questo caso, sia l'ordinativo della merce che quello per il trasporto vengono confermati.

Il fornitore della merce fornisce il proprio servizio anche ad altre industrie, e così non desidera perdere concorrenza *bloccando* il data-base degli ordinativi per l'intera durata di una transazione. Sarebbe disposto a "rinunciare" all'Isolamento, in cambio di una maggiore disponibilità del proprio servizio.

Oltre l'Atomicità, le parti coinvolte nella transazione richiedono che sia rispettata pure la Persistenza, affinché sia ricordato il risultato raggiunto di comune accordo.

Valutazione

Anche in questo scenario non viene utilizzato un servizio di pagamento on-line. Tuttavia, un ordinativo non potrà più essere interpretato come di cortesia, poiché ora rappresenta l'esecuzione del primo passo di un contratto che sarà certamente perfezionato.

Nel caso specifico, poiché si ragiona in un contesto di fiducia, dal punto di vista dell'industria manifatturiera, ovvero il cliente, la garanzia di atomicità non è un requisito essenziale, a patto che un eventuale risultato non atomico sia sempre segnalato, cosicchè sia possibile richiedere l'annullamento dell'altra operazione. Inoltre, l'annullamento dovrebbe essere richiesto in tempi brevi, per non rischiare di recare un danno al proprio partner.

Poiché la conferma di un ordinativo sarà certamente preceduta da "operazioni di lettura", ad esempio per conoscere prezzi, disponibilità, etc. il cliente vorrebbe anche l'Isolamento.

[31] Un esempio del tutto equivalente è presente in Riferimenti[3 p15].

Dal punto di vista dei partecipanti, oltre alla necessità di persistenza, l'autore sottolinea l'importanza delle performance di concorrenza. In particolare afferma che il fornitore delle materie prime vorrebbe fornire un servizio non isolato, in cambio di una maggiore disponibilità.

Per capire concretamente questa affermazione supponiamo che a seguito di un ordinativo, per semplicità, il servizio del fornitore effettui due sole operazioni:

- Registra localmente l'ordinativo con tutte le informazioni relative al cliente.
- Decrementa le variabili locali che indicano la disponibilità in magazzino delle corrispondenti merci ordinate.

Se si utilizzasse il modello transazionale classico, in cui, relativamente alle operazioni di scrittura, viene necessariamente utilizzato lo Strict 2PL, all'ordinativo di una certa quantità della merce X corrisponderebbe il *lock* della relativa *variabile di disponibilità*, e tale merce non potrebbe essere venduta ad altri acquirenti fino al termine della transazione in questione, quando detta variabile sarebbe riportata ad uno stato consistente.

Per incrementare la disponibilità, seppure non menzionato nella descrizione dello scenario da noi esposta, l'autore propone l'utilizzo del **Modello Compensativo**. Tralasciando per ora i dettagli relativi al Modello, che analizzeremo approfonditamente nel paragrafo "3.2 Interpretazioni dei Modelli Compensativi", esso prevede che l'ordinativo sia subito registrato, e le variabili di disponibilità subito decrementate. In caso di fallimento, prevede che sia invocata un'azione *compensatrice* per cancellare l'ordinativo dal database, e per incrementare le variabili di disponibilità. Queste ultime non dovranno essere *lockate* tra le due operazioni di decremento e incremento. Risulta dunque chiaro che, adottando tale modello, le transazioni non saranno tra loro isolate, poiché prima dell'eventuale azione compensatrice saranno certamente possibili *letture sporche*.

Nell'esempio, tuttavia, la mancanza di isolamento non rappresenta in effetti un problema poiché, al più, si tradurrebbe in un sotto utilizzo delle risorse disponibili. Infatti, assumendo che l'utente possa successivamente annullare un ordine fatto in precedenza inviando un contrordine, si otterrebbe semplicemente un incremento delle corrispondenti variabili di disponibilità, precedentemente decrementate. Se ora un

utente effettua le due operazioni dette, e nel mezzo un altro utente dovesse richiedere un ordinativo sulla stessa merce, quest'ultimo non sarebbe isolato dal primo, in quanto potrebbe ottenere una indisponibilità di merce, che invece sarebbe stata disponibile a seguito del contrordine del primo utente. Questa mancanza di isolamento, fortunatamente, non danneggerebbe direttamente il secondo cliente, che potrebbe rivolgersi altrove. Non danneggerebbe neanche il fornitore del servizio, che pur perdendo il cliente nella situazione detta, fornirebbe normalmente un servizio con una maggiore disponibilità, essendo maggiore il possibile numero di accessi concorrenti nell'unità di tempo.

I problemi potrebbero nascere, invece, nel caso in cui i dati riguardanti il magazzino siano utilizzati anche da altre applicazioni, che eventualmente potrebbero non accettare letture sporche. Ad esempio, un'applicazione di questo tipo potrebbe essere un sistema automatico che, al di sotto di una certa disponibilità per un dato prodotto, invii automaticamente un ordinativo al corrispondente produttore. Prima degli eventuali annullamenti da parte dei clienti, si potrebbe verificare una condizione transitoria che induca il sistema automatico ad effettuare l'ordinativo della merce "virtualmente" finita, con ovvi problemi di immagazzinamento. Il problema potrebbe essere risolto imponendo che il sistema automatico entri in azione solo quando tutte le transazioni utente siano completate, ovvero solo dopo che sia stato ripristinato momentaneamente l'isolamento.

La politica dell'isolamento *rilassato* appena esposta, può essere utilizzata, in realtà, da tutti quei servizi che si comportano in maniera analoga al nostro "servizio fornitore". Infatti, per essi, le regole stesse dell'applicazione possono minimizzare e/o evitare i problemi che derivano dal non soddisfacimento della proprietà in questione. Come osservato anche nell'esempio, tuttavia, in questi casi, per evitare anomalie, spesso dovrebbero essere utilizzati degli opportuni accorgimenti, quasi sempre dipendenti dal particolare servizio offerto.[32]

Prendendo spunto da questo scenario, introdurremo ora un altro importante requisito. Supponiamo che il servizio fornitore sia un intermediario, ovvero che, per esaudire i propri clienti, si rivolga ad altri servizi. In tal caso, ad un ordinativo del cliente, il

[32] Un approfondiremo sulla questione sarà dato in "3.1 Modelli tradizionali: analisi rispetto ad Atomicità e Isolamento".

servizio fornitore potrebbe reagire invocando nuove transazioni. Tali invocazioni non dovrebbero essere visibili al cliente, poiché costituiscono parte integrante della logica interna del servizio utilizzato, e tale logica dovrebbe poter cambiare senza richiedere necessariamente variazioni nella logica dei servizi richiedenti. La tecnica che permette questo tipo di intermediazione, facendo corrispondere ad un'operazione transazionale altre transazioni, è nota come **Interposizione**. Essa prevede che un esecutore possa assumere contemporaneamente anche il ruolo di richiedente. In tal caso esso viene detto **Subordinato**.

Seppure in generale le operazioni di un servizio esecutore possano avere una durata impredicibile, ad esempio perché richiedono l'intervento di un operatore umano, l'adozione del modello dell'Interposizione enfatizza tale possibilità. Pertanto, per il cliente, sarebbe desiderabile avere qualche meccanismo con cui poter stabilire se le operazioni dei servizi invocati siano realmente terminate, affinché possa richiedere la conferma della transazione nel momento appropriato, senza rischiare che questa si concluda con un insuccesso semplicemente perché uno, tra i servizi esecutori, non ha avuto il tempo necessario per l'elaborazione. A tale meccanismo, inoltre, dovrebbe esserne associato un altro, affinché il cliente possa conoscere lo stato corrente del servizio esecutore quando questo non completi i lavori entro un tempo stabilito, eventualmente stimato e comunicato in anticipo dal servizio esecutore stesso, e possa così distinguere l'eventualità che il servizio in questione richieda un tempo maggiore per le proprie operazioni, piuttosto che abbia avuto un qualche problema.

E' superfluo dire che le operazioni dovranno avvenire in un contesto Sicuro, e che i servizi coinvolti nella transazione, oltre che nel rispetto della Persistenza, vorranno certamente operare nel rispetto della Consistenza dei dati.

1.3.5 Financial Trading

Un agente finanziario vuole acquistare azioni ed opzioni (consistono nel pagare un premio che consente di diminuire le perdite in caso di discesa repentina delle prime) ai prezzi migliori. A tal proposito sollecita continuamente le offerte dai servizi Web di diversi *brokers* per ottenere il prezzo più basso per entrambe. Conferma le offerte migliori, e lascia che le offerte peggiori scadano per timeout.

L'efficacia della strategia di acquisto consiste nel comperare dette azioni ed opzioni ad un certo prezzo, e necessariamente insieme, poiché il prezzo delle opzioni dipende sempre dal prezzo corrente delle corrispondenti azioni.

Valutazione

A differenza dei precedenti scenari, in questo, pur non essendo citato un servizio di pagamento on-line, i vincoli applicativi richiederanno certamente che un acquisto del cliente non sia revocabile. Dunque, la garanzia di atomicità rappresenta un requisito inderogabile.

Un ulteriore differenza riguarda il requisito sulla revoca di partecipazione di un partecipante: mentre in precedenza costituiva una garanzia di autonomia, ora è parte essenziale della logica business. Infatti, in seguito alle richieste del cliente, i partecipanti stabiliscono la durata delle proprie offerte, assegnandole un tempo massimo dopo il quale esse perdono automaticamente validità: si parla pertanto di *revoca automatica*. Tale requisito, nell'esempio in esame, assume un'importanza fondamentale, in quanto l'utente non vuole e non può perdere tempo ad effettuare *annullamenti*, e le offerte variano troppo rapidamente per poter essere revocate singolarmente presso ogni partecipante.

Inoltre, per i servizi esecutori (i servizi dei brokers), è indispensabile la possibilità di concedere le offerte in *overbooking*, mentre per i clienti è indispensabile ragionare su dati attuali. Un cliente, infatti, non vorrebbe fare scelte su offerte che, seppure non scadute per timeout, siano state già assegnate definitivamente ad un altro cliente. Per lo stesso motivo il cliente vorrà anche l'Isolamento delle proprie operazioni.

Le stesse considerazioni fatte per questo scenario, valgono, più in generale, per tutte quelle applicazioni che prevedano la vendita mediante aste on-line (si pensi al successo di e-bay). L'unica eccezione potrebbe riguardare la garanzia di atomicità, che sarà un requisito essenziale solo per le applicazioni in cui siano venduti beni soggetti ad un'alta variabilità del proprio valore nel tempo, e per i quali un'operazione di annullamento dell' acquisto non potrebbe essere accettabile (a meno del dovuto risarcimento).

Come sempre, Sicurezza, Persistenza e Consistenza sono requisiti scontati.

1.3.6 Micro-paying

L'utente effettua acquisti di beni a basso costo attraverso un'applicazione che utilizza due servizi web, quello del fornitore del bene, e quello che lo abilita all'acquisto mediante *token* (Micro-payment service). Il business è dato dall'alto numero di piccoli acquisti, poiché ognuno di essi coinvolge una somma di denaro troppo bassa per poter essere significativo singolarmente. Beni candidati ad essere venduti con *micro pagamenti* sono tutti quei beni che non richiedono uno spostamento fisico, ma per i quali lo scambio può essere effettuato interamente on-line: file mp3, software, notiziari, informazioni di borsa, etc.

Supponiamo, in particolare, che l'utente sia interessato all'acquisto di files mp3.

L'applicazione client contatta il servizio fornitore del bene, seleziona la sua canzone preferita, e ottiene in ritorno il prezzo e i dettagli dell'acquisto. A questo punto richiede un *token di pagamento* al proprio servizio di micro-payment, e lo consegna al servizio del fornitore: il token è inattivo. Il fornitore, verificata la validità del token, concederà un *token di acquisto*, anch'esso inattivo, all'applicazione client.

Se tutto è andato bene, si può passare all' ultima fase, che consiste nel rendere entrambi i token attivi. Così il servizio fornitore potrà ottenere il credito, e il servizio cliente potrà effettuare il download del file mp3. In termini transazionali questa equivale alla fase di Commit: se entrambe le parti effettueranno il commit, i token saranno resi attivi.

Questo è un esempio ordinario di uso delle transazioni, in cui usualmente si da poca importanza alla risoluzione delle condizioni di *failures*, dovute, ad esempio, a cadute di server o indisponibilità della rete.

Valutazione

Lo scenario prevede la partecipazione di due servizi esecutori, quello del fornitore del bene, e quello del servizio di micro-payment. Raggiunto il commit, il primo attiverà il token di acquisto, mentre il secondo il token di pagamento.

L'utente, oltre all'atomicità, vorrà l'isolamento. Questo non dovrebbe essere un problema per il servizio fornitore, in quanto è lecito supporre che i prezzi delle canzoni e i dettagli dell'acquisto siano dati abbastanza stabili. Sempre per quanto riguarda il fornitore, nel caso specifico, il risultato della transazione (l'attivazione del token di acquisto) dovrebbe essere memorizzato, e così non è superfluo parlare di Persistenza e Consistenza. Lo Scenario non aggiunge nessun requisito importante. Vuole semplicemente mostrare che esistono situazioni in cui, dal punto di vista di un *esecutore*, la disponibilità del proprio servizio è l'unica caratteristica essenziale, anche quando la transazione business preveda la partecipazione di un servizio di pagamento.

Questo, tuttavia, non è vero per il servizio di micro-payment, che vorrà fornire, invece, le dovute garanzie di atomicità e isolamento ai propri clienti, e vorrà operare nel rispetto delle caratteristiche di Persistenza e Consistenza.

Un altro concetto importante è che il business dipende dall'alto numero di vendite, pertanto l'architettura di supporto deve essere performante e scalabile.

Infine, come al solito, è necessario un contesto Sicuro in cui condurre le operazioni.

Si noti che la tecnica utilizzata in questo scenario potrebbe essere impiegata anche per applicazioni in cui lo scambio non preveda l'uso esclusivo del Web. Questo è il caso, ad esempio, per quei servizi che richiedono pagamenti anticipati online, eventualmente anche solo in forma di caparra.

1.4 Analisi e specifica dei Requisiti

In questo capitolo, basandoci sulle considerazioni riportate nella valutazione degli scenari appena presentati, e analizzando le caratteristiche del particolare ambiente in cui questi saranno eseguiti, cercheremo di specificare un insieme di requisiti generali che permettano lo svolgimento di ognuno di essi. Utilizzeremo i concetti esposti per il modello transazionale classico, presentato nel capitolo 1.2, che, implicitamente o esplicitamente, sarà il termine di riferimento per i nostri ragionamenti.

Per il momento, saranno utilizzati indistintamente i termini *coordinatore* e *meccanismo di coordinamento*, senza nulla imporre come requisito: il meccanismo di coordinamento potrà essere inglobato nei ruoli di richiedente e partecipante, oppure potrà essere un servizio a se stante. Ritratteremo la questione in "3.5 Problemi irrisolti".

Per riassumere graficamente alcuni dei concetti esposti, utilizzeremo degli Activity Diagrams con notazione informale, definita dal simbolismo riportato di seguito:

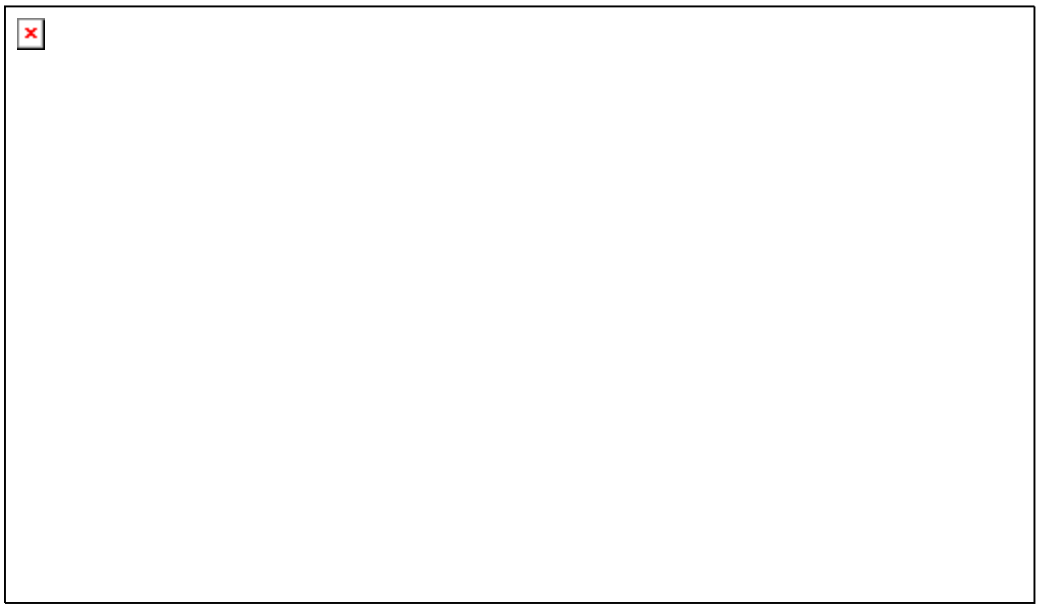

Figura 8 Simbolismi per gli Activity Diagrams

1.4.1 Ambiente di esecuzione

Prima di specificare i requisiti che dovrebbero essere soddisfatti da un *middleware* di supporto per transazioni in ambiente WebServices, cercheremo di caratterizzare meglio il contesto transazionale in cui dovremo ragionare.

Abbiamo precedentemente visto la tecnologia impiegata dai DBMS per supportare le Transazioni Distribuite tradizionali. Queste vengono utilizzate all'interno di un singolo dominio organizzativo, i cui confini, molto spesso, sono delimitati da una intranet. Una rete di questo tipo solitamente utilizza mezzi trasmissivi con ampia larghezza di banda, ed è circoscritta ad una regione di spazio relativamente piccola, cosicché la probabilità di *failures,* già abbastanza contenuta, risulta ulteriormente ridotta. Sono reti dimensionate in base all'utenza reale: difficilmente sono soggette a problemi di congestionamento. Il tutto si traduce in una rete molto veloce, e anche grazie a tale velocità, possono essere impiegati dei protocolli di trasporto più "pesanti", ma anche molto più affidabili. Questi ultimi sono denominati Connection Oriented poiché, utilizzando il concetto di connessione, garantiscono il giusto ordinamento di consegna per i messaggi, e la ritrasmissione in caso di errori, ovvero la consegna affidabile dei messaggi[33]. Inoltre, per problemi di Sicurezza, spesso una intranet viene isolata attraverso Firewall, ovvero dispositivi che limitano l'accesso esterno.

Aziende che utilizzano questo tipo di tecnologie difficilmente esporranno i propri DBMS all'esterno, se non attraverso opportune applicazioni utente, controllate ancora dall'interno: ad esempio, attraverso un sito Web. In altre parole, il richiedente, cioè l'applicazione, e i servizi esecutori, cioè i DBMS, sono gestiti dalla stessa società. Infine, un'ultima caratteristica importante per le transazioni tradizionali, riguarda la loro durata. Queste richiedono mediamente un'elaborazione breve, mentre, come si deduce degli scenari precedentemente proposti, le transazioni business avranno una durata media molto più lunga.

[33] Un messaggio sarà certamente consegnato privo di errori, oppure non sarà consegnato. Questo secondo caso può verificarsi se la connessione viene persa, ad esempio in seguito ad un malfunzionamento di qualche router di passaggio essenziale, o se il ricevente ha qualche problema.

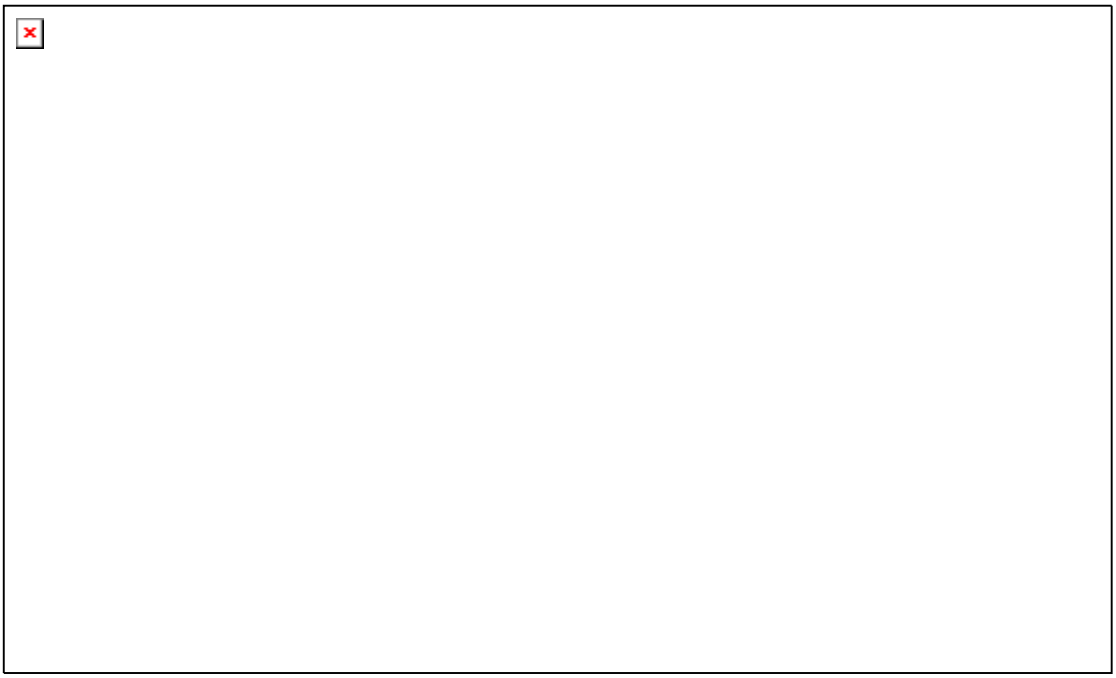

Figura 9 Caratteristiche principali dell'ambiente per transazioni distribuite tradizionali

Le caratteristiche appena menzionate, e riassunte nello schema di Figura 9, si conciliano perfettamente con le tecnologie utilizzate dai DBMS. Ad esempio, la velocità della rete consente una effettiva minimizzazione della *finestra di incertezza*, mentre la presenza di un gestore unico permette di rilassare l'atomicità, così come fa X-Open permettendo decisioni euristiche, semplicemente perché i problemi che ne derivano sono interni all'organizzazione stessa. Inoltre, la presenza di un controllo centralizzato consente, per le interazioni tra nodi interni al dominio, di garantire le proprietà di atomicità e isolamento in maniera semplice: basta utilizzare il 2PC nella coordinazione e imporre lo Strict 2PL su ogni nodo, tecniche ormai mature e ben conosciute.

Il contesto delle Transazioni Business è per molti versi, invece, l'opposto di quello appena descritto. Una delle due caratteristiche salienti dei Web Services, menzionata inizialmente, è, infatti, quella dell'interoperabilità, e la massima forma di interoperabilità si esplica proprio attraverso Internet e HTTP, in un contesto di servizi lascamente accoppiati e indipendenti.

Attraversare Internet significa attraversare potenzialmente qualsiasi tipo di rete, con tutto ciò che questo comporta: la condivisione della banda di trasmissione con molti altri utenti della rete potrebbe produrre una banda utile molto ristretta, e le perdite di

messaggi per congestioni o le ritrasmissioni dovute ad errori di comunicazione aggravano ulteriormente la situazione.

HTTP è, poi, un protocollo di richiesta-risposta, ovvero un protocollo di tipo Connectionless, caratteristica, questa, che male si addice alle stringenti necessità di affidabilità richieste in ambito transazionale. Fortunatamente, appoggiandosi su TCP, che è un protocollo orientato alla connessione, un messaggio HTTP arriverà certamente privo di errori, ma non vi è nessuna garanzia che arrivi[34], né tantomeno che i messaggi siano consegnati nello stesso ordine con cui sono stati inviati[35].

Infine, oltrepassare i confini della propria rete aziendale comporta evidentemente problemi di Sicurezza, che pur avendo una notevole rilevanza in generale, diventano cruciali nell'ambito specifico delle transazioni business.

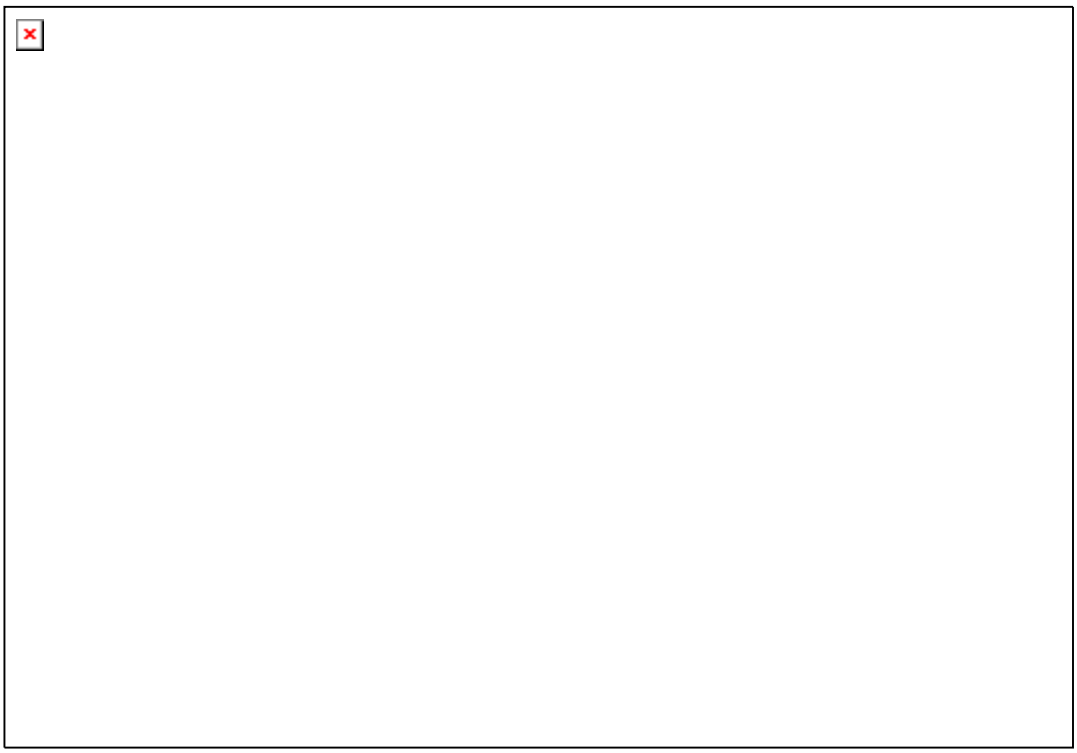

Figura 10 Caratteristiche essenziali dell'ambiente in cui si svolgono le Transazioni Business

Dunque, le transazioni business prevedono interazioni **multi-dominio**, ovvero tra sistemi appartenenti a domini business distinti (sistemi *lascamente accoppiati*), e si svolgono nell' ambiente dei Web Services, che usa come protocollo di trasporto

[34] Per gli stessi motivi per cui non vi è garanzia di consegna anche per un protocollo di trasporto orientato alla connessione.
[35] Per ogni richiesta viene infatti utilizzata una nuova connessione.

principale HTTP. Pertanto, l'ambiente di esecuzione può essere caratterizzato come segue:

- **Insicuro**: tali sistemi interagiscono tra di loro attraversando Internet, che è una rete altamente insicura.
- **Utilizza protocolli di Trasporto lenti ed inaffidabili**: il protocollo HTTP è intrinsecamente inaffidabile: messaggi possono perdersi sulla rete, oppure arrivare in un ordine diverso da quello con cui sono stati inviati. Inoltre, l'infrastruttura fisica sottostante Internet, non è ancora in grado di assicurare nessun livello di qualità, essendo di tipo *best effort* la politica globale per la consegna dei messaggi: nessuna garanzia sui tempi di consegna.

Infine, le interazioni avvengono tra sistemi…

- **Indipendenti**: sistemi autonomi nel controllo, nella gestione e nella proprietà. Questa è una caratteristica di base nel contesto in esame, e pertanto faremo riferimento ad essa come ad un principio, che chiameremo *Principio di Autonomia*. Una ulteriore caratteristica delle transazioni business, diretta conseguenza dell'accettazione di questo principio, riguarda la loro durata, che potrà essere molto lunga, e spesso impredicibile.

1.4.2 Requisiti Iniziali

In questo paragrafo saranno precisati (specificati) alcuni dei requisiti che sono stati dedotti nella valutazione degli scenari. Per distinguerli dai requisiti che introdurremo in seguito, li abbiamo denominati “iniziali”. Questi possono essere raggruppati, così come avevamo già fatto nella valutazione, a seconda che riguardino il servizio richiedente, i servizi esecutori, o entrambi.

1.4.2.1 Requisiti degli esecutori

Le caratteristiche peculiari di ogni partecipante ad una transazione business, sono tutte una diretta conseguenza del Principio di **Autonomia** imposto dalle interazioni multi dominio.

In generale, infatti, un partecipante non vorrà essere sottoposto a troppi vincoli che siano imposti dall’esterno del proprio dominio business di appartenenza, a meno che questi non siano strettamente necessari per l’esecuzione della propria logica business.

Nella maggioranza dei casi, un esecutore vorrà operare nel rispetto delle proprietà di Persistenza e Consistenza. Il fatto che il risultato di una transazione debba essere sempre persistente, è certamente un requisito indiscutibile, altrimenti si ammetterebbe la possibilità che l’effetto di una transazione sia superfluo: soltanto le transazioni di sola lettura non hanno un effetto. La transazione, inoltre, non dovrebbe violare i vincoli di integrità definiti sui dati, poiché tale evento, evidentemente, sarebbe indice di una cattiva applicazione o di un malfunzionamento. Inoltre, in accordo al Principio di Autonomia, un partecipante potrà decidere di rispettare o meno tali caratteristiche, a seconda delle proprie necessità, o meglio del “contratto applicativo” che dovrà rispettare nei confronti del cliente, infatti, come anticipato in precedenza, sia Persistenza che Consistenza sono proprietà locali all’esecutore. Per lo stesso motivo, non necessitano di alcun supporto transazionale.

Il supporto transazionale è, invece, necessario per garantire gli altri requisiti che discendono dal rispetto del Principio di Autonomia:

RP1. Disponibilità[36]

E' spesso un fattore determinante per la riuscita di alcune logiche business. Può dipendere da particolari scelte per la gestione degli accessi concorrenti, e dunque dalle particolari tecniche adottate per ottenere l'isolamento. Come abbiamo visto, alcuni servizi potrebbero voler rilassare l'isolamento anche rispetto alle operazioni di scrittura. Questo è possibile in particolari circostanze, quando le regole applicative minimizzano e "controllano" gli effetti collaterali. Modelli transazionali che rilassano l'isolamento e/o l'atomicità, sono comunemente detti Modelli Rilassati[37].

Per alcuni servizi può essere fondamentale la possibilità di concedere risorse in *overbooking*, e come vedremo in "3.3 Tecnica dell'Overbooking", questo non è sempre possibile senza rinunciare ad altri requsiiti, ma dipende dal particolare protocollo di coordinamento.

Tuttavia, la Disponibilità può anche dipendere da fattori esterni alle scelte implementative e alla logica propria del servizio. Sono, infatti, possibili diversi *attacchi* alla Disponibilità, con conseguente degradazione o completo annullamento delle performance.

Alcuni esempi[38] di attacchi alla Disponibilità sono:

PULL-Based Denial-of-Service Attack

Se il protocollo di coordinamento utilizza il paradigma di tipo *Pull*[39] per l'arruolamento di un partecipante con il coordinatore, un "malintenzionato" potrebbe riuscire ad arruolarsi, e subito dopo potrebbe chiudere la connessione. Così forzerebbe un abort della transazione ad insaputa del richiedente. In questo

[36] Nell'esempio di "1.3.6 Micro-paying" o in quello di "1.3.5 Financial Trading", la disponibilità è un requisito fondamentale.
[37] Vedi Riferimenti[35 p172 (Cap3)]
[38] Gli esempi sono tratti da Riferimenti[23 cap16].
[39] L'atto con cui un servizio viene "iscritto" come partecipante in una transazione viene comunemente detto *arruolamento*. Se l'atto viene compiuto del servizio stesso, si parla di paradigma di tipo PULL per l'arruolamento.

modo il coordinatore, e quindi i servizi esecutori, risulterebbero non utilizzabili anche per richieste legittime, e, dunque, sarebbero indisponibili.

Il problema può essere risolto richiedendo l'autenticazione nell'arruolamento, e verificando che l'arruolante abbia i permessi necessari.

PUSH-Based Denial-of-Service Attack

Se il protocollo di coordinamento utilizza il paradigma di tipo *Push*[40] per l'arruolamento di un partecipante con il coordinatore, è sempre possibile, per un "malintenzionato", scoprire che un particolare coordinatore mantiene delle informazioni persistenti per ogni transazione avviata e non completata. Il "malintenzionato" potrebbe allora avviare velocemente molteplici transazioni, aspettare che entrino in una fase in cui il coordinatore memorizzi dette informazioni (ad esempio dopo la fase di Prepare per il 2PC), e successivamente disconnettersi.

Alla lunga, il servizio coordinatore esaurirebbe tutte le risorse, e risulterebbe indisponibile per un uso legittimo.

Anche questo problema può essere risolto richiedendo l'autenticazione nell'arruolamento, e verificando che l'arruolante sia di fiducia.

RP2. Revoca di Partecipazione

La disponibilità a partecipare ad una transazione business, in accordo al Principio di Autonomia del partecipante, deve poter essere revocata in qualsiasi momento, a meno che non sia stato diversamente pattuito con il richiedente. Tale revoca deve poter avvenire anche in maniera *automatica*, ovvero senza l'intervento diretto, presso il richiedente, del partecipante in questione. Ad esempio, un partecipante deve potersi dichiarare disponibile solo per un certo periodo di tempo, trascorso il quale il richiedente saprà automaticamente che detta disponibilità è stata revocata.

[40] Se invece l'atto di iscrizione viene compiuto dal richiedente della transazione, si parla di paradigma di arruolamento di tipo PUSH.

RP3. Rilassamento dell'Atomicità

Un partecipante non deve necessariamente essere vincolato dall'Atomicità, ma deve poter fornire un servizio in cui tale caratteristica sia rilassata in maniera consona alle proprie possibilità ed esigenze. Ad esempio, un servizio potrebbe scegliere di non implementare la Persistenza per economizzare la propria infrastruttura, e per lo stesso motivo non vorrà garantire l'Atomicità.

RP4. Interposizione

Se il servizio di un partecipante sia implementato mediante la composizione transazionale di altri servizi, non si vuole che detta composizione sia esposta all'esterno, come avverrebbe utilizzando un modello transazionale "piatto". Questo requisito è in completo accordo non solo con il Principio di Autonomia, ma anche con il modello SOA e la nozione di *incapsulamento*, e l'introduzione del modello transazionale gerarchico, associato alla tecnica dell'Interposizione, lo soddisfa pienamente.

Tale tecnica, oltre a rispettare la nozione di incapsulamento e a permettere una migliore strutturazione del flusso di controllo per il trattamento degli errori[41], fornisce anche un altro importante vantaggio. Infatti, quando una transazione dovesse coinvolgere molti partecipanti, nel caso in cui ci fosse un solo coordinatore, questo sarebbe sovraccaricato dal dover comunicare con tutti. Il sovraccarico sarebbe ancora più marcato se la rete sottostante dovesse essere particolarmente inaffidabile, con un'elevata probabilità di ritrasmissione. In questi casi, se più partecipanti in uno stesso Dominio potessero essere raggruppati e fare riferimento ad un coordinatore locale che fungesse da "proxy" del coordinatore principale, si partizionerebbe il carico di lavoro sia tra i coordinatori che tra le sottoreti. In questo modo il traffico sulla rete "più inaffidabile" sarebbe minimizzato. Pertanto, l'adozione della tecnica dell'Interposizione, concorre anche nel soddisfacimento del requisito "RC3 Protocolli Efficienti".

Abbiamo già osservato che le transazioni business hanno una durata mediamente molto più lunga delle transazioni tradizionali. L'utilizzo della tecnica

[41] Essendo ogni transazione "figlia" gestita separatamente.

dell'Interposizione enfatizza questa caratteristica, poiché l'elaborazione di durata lunga e impredicibile di un servizio partecipante può dipendere dall'elaborazione, a sua volta di durata lunga e impredicibile, di altri servizi. Alcuni dei requisiti che introdurremo nel paragrafo "1.4.3 Risoluzione dei Conflitti" saranno una diretta conseguenza di questa notevole differenza.

1.4.2.2 Requisiti del richiedente

Oltre il Principio di Autonomia, sempre valido, i requisiti del richiedente possono essere riassunti come segue:

RR1. Composizione dinamica

Il richiedente deve poter comporre una transazione business dinamicamente, ovvero deve poter inserire nuove *operazioni* ed eliminarne altre prima di richiedere il completamento della transazione. Inoltre, tale composizione deve poter avere una **durata qualsiasi**, a seconda delle necessità applicative e/o di interazione con l'utente.

RR2. Atomicità

Per molte applicazioni l'atomicità è un requisito essenziale. Affinché essa sia garantita anche in caso di *failures*, di nodo e/o di comunicazione, non solo è necessario definire opportuni protocolli di coordinamento, ma è anche necessario definire un preciso modello per il Recovery.

RR3. Isolamento

Il richiedente vorrebbe che le proprie richieste siano esaudite come se fosse l'unico utente dei servizi. Del resto, la composizione di molteplici servizi distinti all'interno di transazioni, può produrre applicazioni la cui correttezza risulta difficilmente verificabile. In certi casi, per ottenere delle migliori performance di accesso concorrente, alcune applicazioni potrebbero accontentarsi di un isolamento rilassato per le operazioni di lettura, così come accade con le transazioni tradizionali. E' opportuno

sottolineare, per ottenere l'Isolamento Globale, che il protocollo di coordinazione renda applicabile la condizione necessaria riportata a pag32.

RR4. Segnalazione risultato non atomico

Nel caso in cui l'atomicità non possa essere rispettata, il richiedente vorrà almeno essere messo al corrente di eventuali operazioni non eseguite. Affinché ciò sia possibile anche in presenza di *failures*, come per l'atomicità, è necessario definire un preciso modello per il Recovery.

1.4.2.3 Requisiti comuni richiedente-partecipanti

I requisiti elencati nei due precedenti paragrafi, separatamente per richiedente ed esecutori, sono attribuibili al modello di coordinazione transazionale per servizi business.

Come conseguenza del particolare ambiente di esecuzione, determinato dall'associazione di Web Services, Internet e HTTP, esistono anche altri requisiti, questa volta più propriamente attribuibili al *middleware* di supporto transazionale. Essi sono:

RC1. Comunicazioni Sicure

La caratteristica di *insicurezza* dell'ambiente di esecuzione impedisce qualsiasi rapporto business e/o fiduciario. Pertanto, in un "macro contesto" insicuro quale è Internet, è necessario ricavare virtualmente dei ***Contesti di Fiducia***, all'interno dei quali sia possibile svolgere le attività business in tutta sicurezza. Come osservato precedentemente, tale requisito risulta essenziale per evitare *attacchi* alla Disponibilità. Tuttavia, sono possibili anche altri *attacchi* alla sicurezza, e qui di seguito vengono solo accennati degli esempi[42] particolarmente noti in letteratura.

[42] Gli esempi sono tratti da [23 cap16].

Transaction Corruption Attack

In caso di caduta del Transaction Manager, un "malintenzionato" potrebbe sostituirsi ad esso e così potrebbe, ad esempio, dirigere i partecipanti verso un risultato che comprometta l'atomicità della transazione.

Il problema può facilmente essere risolto ricordando l'identità del TM, e richiedendo l'autenticazione all'avvio di ogni nuova connessione.

Packet-Sniffing Attack

Se un "malintenzionato" riuscisse ad intercettare dei messaggi appartenenti ad una interazione business, potrebbe trarre delle informazioni utili per pianificare ulteriori attacchi. Ad esempio, se un campo di commento di un messaggio indicasse il nome e la versione del software utilizzato per il coordinamento, il "malintenzionato" potrebbe utilizzare dette informazioni per avvantaggiarsi di eventuali bugs, noti per tale versione software.

Il problema può essere evitato utilizzando la crittografia per tutti i messaggi, eventualmente con una chiave di sessione che velocizzi le operazioni di criptaggio-decriptaggio.

Man in the middle Attack

Se un "malintenzionato" riuscisse ad intercettare e alterare dei messaggi appartenenti ad una interazione business, potrebbe creare notevoli inconvenienti. Ad esempio, potrebbe sostituire un comando di COMMIT con un comando di ABORT.

Il problema può essere risolto utilizzando sia l'autenticazione che la crittografia di sessione.

RC2. Comunicazioni Affidabili

Nell'ambito delle transazioni business l'Affidabilità del protocollo di trasporto risulta una caratteristica essenziale. Ad esempio, se il messaggio contenente una richiesta

applicativa fosse riinviato una seconda volta, in seguito alla perdita del messaggio di risposta relativo alla prima richiesta, sarebbe interpretato correttamente come una rispedizione?

Inoltre, il protocollo di trasporto deve assicurare che i messaggi siano consegnati nello stesso ordine di invio, poiché in una conversazione alcuni messaggi potrebbero essere leciti solo se ricevuti nella corretta sequenza. Ad esempio, la conferma di un ordine è lecita solo dopo la comunicazione dell'ordine stesso.

RC3. Protocolli Efficienti

Per compensare la lentezza e l'inaffidabilità del mezzo di comunicazione devono essere definiti protocolli efficienti. Pertanto, dovrebbe essere minimizzato il numero di messaggi scambiati, soprattutto quelli che attraversano la rete più inaffidabile, ad esempio adottando la tecnica dell'Interposizione, e dovrebbe essere minimizzato il numero di Abort[43].

Tale caratteristica, in generale, risulta desiderabile per incrementare il livello di Disponibilità dei servizi, ma risulta essenziale per quelle applicazioni in cui lo stato è altamente variabile. Ad esempio, il successo dell'agente finanziario nello scenario "1.3.5 Financial Trading" dipende strettamente dalla tempestività delle sue azioni.

[43] Questa affermazione sarà più chiara in seguito.

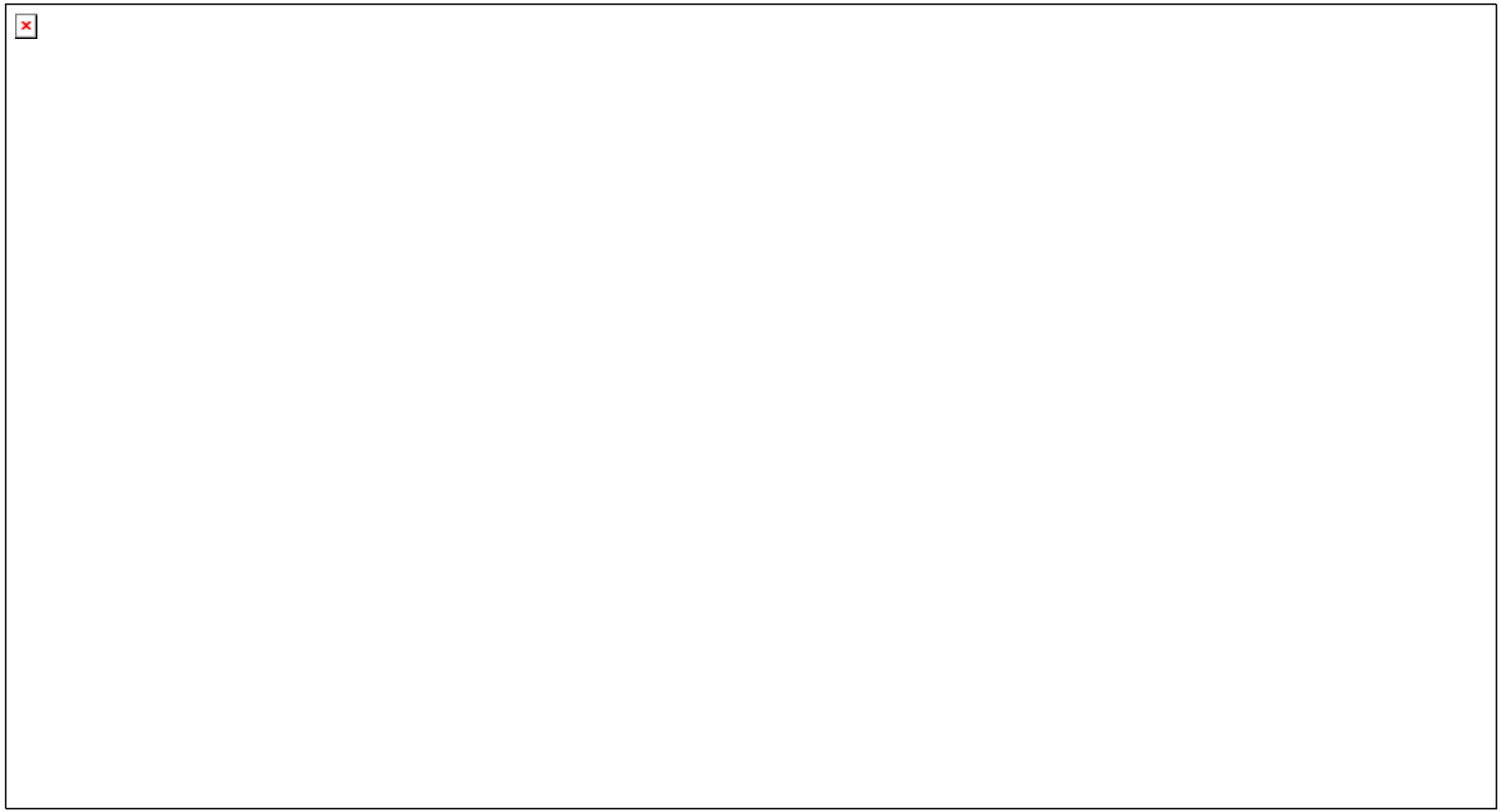

Figura 11 Requisiti Iniziali

1.4.2.4 Supporto per transazioni tradizionali

A questo punto dovrebbe risultare chiara la differenza concettuale tra una transazione distribuita di tipo tradizionale e una di tipo *business*. Infatti, le prime:

- hanno una semantica ben definita, così come prescritto dal soddisfacimento delle proprietà ACID;
- prevedono interazioni intra dominio, poiché sono utilizzate tra DBMS che hanno gestione, controllo e proprietà in comune;
- sono composte da operazioni che normalmente sono delle query;

mentre le transazioni business:

- non hanno una semantica ben definita, poiché potranno rilassare alcune delle proprietà ACID, a seconda delle esigenze dei vari servizi partecipanti;
- prevedono interazioni multi dominio, poiché saranno utilizzate tra servizi autonomi;
- saranno composte da invocazioni di metodi di servizi;

Per quanto riguarda le proprietà ACID, inoltre, cambia completamente il punto di vista dal quale osservarle. Infatti, seppure rappresentino sempre una garanzia di correttezza dell'operazione, per le transazioni tradizionali il punto di vista è unico, quello dell'utente del DBMS, mentre per le transazioni business il punto di vista può essere quello del richiedente, interessato ad Atomicità e Isolamento, o quello dell'esecutore, interessato alle altre due proprietà. Per inciso, si noti che il richiedente non ha interesse in tutte quelle proprietà che rappresentano un problema locale all'esecutore, quali sono appunto Consistenza e Persistenza, poiché, se necessarie al rispetto del "contratto applicativo", sarà interesse dell'esecutore stesso rispettarle.

Tuttavia, i due "mondi" non sono così distinti come potrebbe sembrare, e tra i due estremi sono possibili molte configurazioni intermedie:

1. E' possibile che due servizi business interagiscano all'interno dello stesso dominio, ad esempio perché la stessa organizzazione fornisce all'esterno due servizi, uno dei quali si basa sull'altro.

2. Inoltre, un servizio business solitamente utilizza dei data-base di *backend*, e seppure siano concepibili servizi che memorizzano i dati su file gestiti con tecniche proprie, nella maggior parte dei casi si vorrà continuare ad utilizzare un data-base.
3. E' anche possibile che una singola organizzazione abbia più sedi, e che voglia comporre servizi business utilizzando basi di dati collocate in sedi separate. Oppure, più semplicemente, è possibile che due organizzazioni distinte, ma strettamente collegate, decidano di condividere i propri data-base. In tal caso, pur appartenendo a domini distinti, le due società richiederanno supporto per transazioni classiche: l'invocazione di un metodo trasporterà ora una o più query.

Per la configurazione 1 non ci sono particolari problemi, poiché i requisiti che abbiamo sinora individuato si riferiscono ad un caso peggiore, e dunque, al massimo, ci saranno delle caratteristiche del *middleware* di supporto non utilizzate. Per le altre due configurazioni potrebbero invece sorgere dei problemi.

Affinché un servizio business possa utilizzare un data-base a valle, infatti, è necessario che il protocollo di coordinamento per Transazioni Business sia compatibile/integrabile, in qualche maniera, con le tecnologie disponibili per i DBMS, quali ODBC, JDBC, JTS, X-Open, etc. Ad esempio, in Figura 12, si è ipotizzato che il servizio business situato nel Dominio2 possa utilizzare una base dati attraverso lo standard X-Open, ma, affinché ciò sia possibile, deve essere dato un significato alle fasi del protocollo per Transazioni Business in relazione alle fasi del 2PC previsto dallo standard. Tale compatibilità sarebbe certamente assicurata se nel protocollo business esistessero due fasi (due insiemi di fasi) assimilabili alla fase di Prepare e di Commit rispettivamente, ma vedremo che non sempre è così.

Infine, per quanto riguarda la terza configurazione, è già disponibile lo standard X-Open DTP, ma questo non è specifico per WebServices, e così, anche per ottenere una integrabilità più semplice e flessibile dei servizi, sarebbe desiderabile uno standard specifico per WebServices, ad esempio ottenuto adattando opportunamente lo stesso X-Open. In questo modo, infatti, anche una base dati potrebbe essere esposta come un semplice servizio Web, ovvero tramite un servizio *proxy* per il DBMS "condiviso", con

tutti i pro che tale strategia comporta. D'ora in poi faremo riferimento al "modello per transazioni ACID per WebServices" chiamandolo brevemente "modello per transazioni ACID", o "modello dalla semantica ACID".

Nel Deployment Diagram in Figura 12 si è ipotizzato che il servizio business appartenente al Dominio2, oltre ad utilizzare una base di dati locale al suo dominio, utilizza anche un servizio proxy per accedere alla base di dati appartenente al Dominio3.

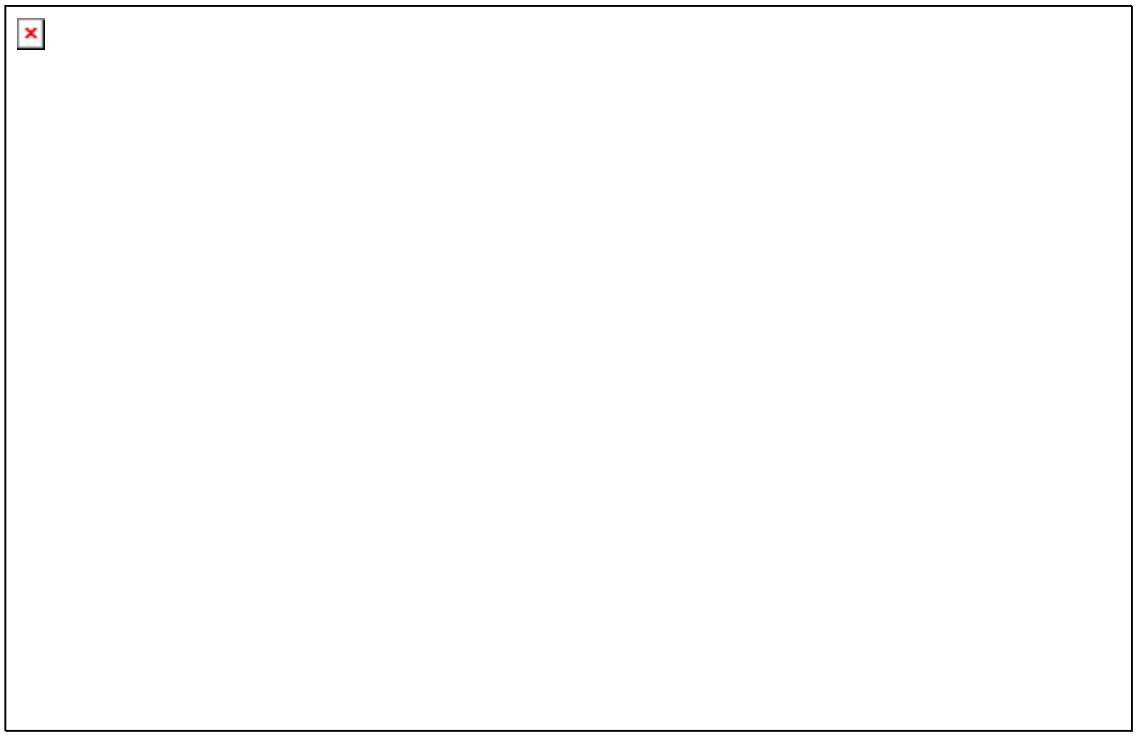

Figura 12 Esempi di integrazione tra servizi Web e DBMS

Come vedremo in seguito, tutte le Specifiche che discuteremo forniscono un modello per transazioni ACID, e non è un segreto che gran parte del successo dei protocolli transazionali per Web Services sarà dovuto proprio alla capacità di assicurare la compatibilità con la tecnologia dei DBMS.

Ma cosa si richiede esattamente ad un modello per transazioni ACID, costruito sopra la tecnologia dei WebServices?

L'ambiente di esecuzione per questo tipo di transazioni è rappresentato in Figura 13. Come per le transazioni business, abbiamo inizialmente a disposizione un trasporto inaffidabile, e se i servizi sono utilizzati attraverso Internet, sarà anche lento e insicuro. Tuttavia, un servizio di tipo ACID difficilmente parteciperà ad una relazione business,

se non indirettamente (ovvero come servizio di supporto ad un servizio business). Solitamente sarà utilizzato da altri servizi con i quali condivide la proprietà, oppure, se questo non è il caso, sarà comunque utilizzato in maniera statica, e non all'interno di transazioni composte dinamicamente. Pertanto, con riferimento a quanto discuteremo nel seguito, qualsiasi tipo di Contratto con il cliente potrà sempre essere, e sarà certamente, definito in sede separata (al di fuori del protocollo di coordinamento), e così, anche per relazioni multi dominio, possiamo continuare a ragionare come se ci fosse un unico proprietario.

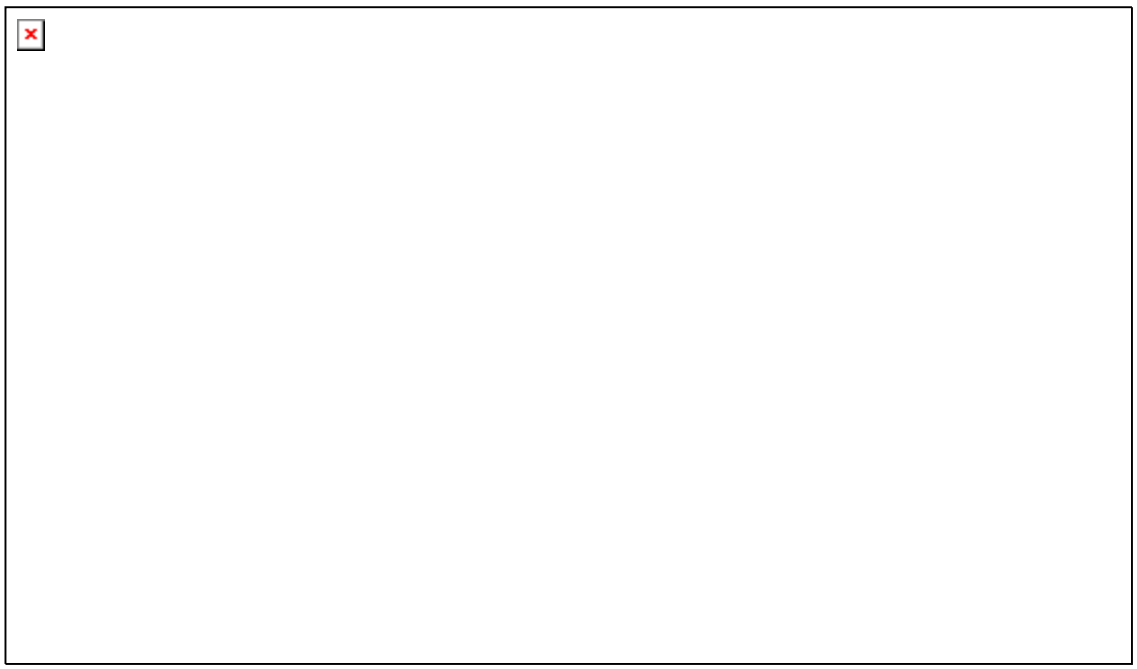

Figura 13 Ambiente di esecuzione per transazioni Web ACID

Di conseguenza, reso Affidabile e Sicuro il Trasporto, l'unica caratteristica che manca alle transazioni ACID per Web, affinché possano utilizzare le stesse tecnologie impiegate per le classiche transazioni distribuite, è la velocità del trasporto, che in relazione al protocollo 2PC, risulta cruciale per minimizzare la *finestra di incertezza*. Evidentemente, non si potrà trasformare Internet in una rete veloce, ovvero dove tutte le comunicazioni avvengano con un'ampia banda di trasmissione e dove la frequenza degli errori di trasmissione sia molto bassa, ma è possibile minimizzare l'inconveniente definendo protocolli efficienti, capaci di minimizzare quanto meno il numero di messaggi scambiati. A tal proposito abbiamo già osservato che l'adozione della tecnica dell'Interposizione può rappresentare un grande vantaggio.

Per tutto quanto detto, il middleware di supporto dovrà rispettare gli ulteriori due requisiti:

RP5. Compatibilità tra Protocollo Business e tecnologia dei data-base

La maggior parte dei servizi business utilizzano, e vorranno continuare ad utilizzare, dei data-base di *backend*. Il protocollo per transazioni business dovrebbe pertanto essere "compatibile" con la tecnologia utilizzata dai DBMS, e dovrebbe essere specificata chiaramente la relazione che lega la semantica di tale protocollo con la semantica delle transazioni ACID, ovvero come sia possibile integrare le due tecnologie.

RC4. Modello per *Transazioni ACID per Web*

Deve essere possibile la costruzione di servizi Web che, come partecipanti, fungano da semplici intermediari verso dei *database server*. Affinché ciò sia possibile, deve essere fornito un protocollo di coordinamento i cui messaggi siano facilmente "mappabili" con quelli previsti dai classici Resource Manager. Ad esempio, un servizio Web "intermediario" dovrebbe poter facilmente tradurre i messaggi del protocollo in questione, nei messaggi previsti dall'interfaccia XA dello standard X-Open. In altri termini, deve essere fornito un modello per transazioni ACID specifico per WebServices. Tale modello deve supportare Affidabilità e Sicurezza delle comunicazioni, e deve essere particolarmente efficiente per sopperire alla lentezza del trasporto imposta dall'ambiente Web.

Figura 14 Conflitti tra caratteristiche e requisiti Iniziali

1.4.3 Risoluzione dei Conflitti

La Figura 11 riassume i requisiti e le caratteristiche precedentemente esposti. Come risulta evidente, questi non sono tra loro sempre compatibili: esistono cioè dei conflitti.

Sono pertanto necessari degli opportuni meccanismi di "compromesso". In realtà, alcuni di essi sono già presenti implicitamente tra i requisiti in questione. Lo schema in Figura 11 può infatti essere riorganizzato come mostrato in Figura 14, dove la presenza di conflitti viene evidenziata dai rombi. In particolare la "revoca di partecipazione" e "l'overbooking delle risorse", possono essere considerati meccanismi di compromesso tra il requisito di Disponibilità dei servizi esecutori, e la caratteristica di "composizione di lunga durata" del cliente; il "rilassamento dell'Atomicità" e la "segnalazione di risultato non atomico" possono essere interpretati come un primo passo verso la risoluzione del conflitto tra il requisito di Atomicità del richiedente, e il Principio di Autonomia applicato agli esecutori; il "rilassamento dell'Isolamento" come un calmiere tra la il requisito di isolamento richiesto dal richiedente e quello di massima disponibilità richiesto dal partecipante.

Nel seguito di questo paragrafo specificheremo quali dovrebbero essere gli altri meccanismi necessari per risolvere i vari conflitti, e quelli che a loro volta derivano dall'introduzione di tali meccanismi. Questi saranno specificati ancora come requisiti, ovvero in una forma generale e tale che sia valida, a prescindere dal particolare protocollo di coordinamento.

RR5. Conoscenza dello "stato" di elaborazione dell'esecutore

L'autonomia di un servizio partecipante, in generale, si traduce in tempi di elaborazione completamente arbitrari e spesso impredicibili. Tale caratteristica viene esaltata dall'applicazione del concetto di Interposizione, per il quale l'elaborazione di un servizio dipende da quella di molti altri, che a sua volta potrà essere di durata arbitraria e impredicibile. Affinché possano essere minimizzati il numero di Abort, il richiedente dovrebbe essere al corrente dello stato di avanzamento di ogni partecipante, in maniera che possa attendere l'eventuale completamento delle operazioni prima di richiedere la

conferma della transazione. Inoltre, questo stesso meccanismo potrebbe essere utilizzato, sempre dal richiedente, per distinguere elaborazioni ancora in corso da eventuali malfunzionamenti del corrispondente servizio esecutore, e dunque per decidere come comportarsi di conseguenza (in accordo al Principio di Autonomia).

A tal proposito potrebbero essere previsti, ad esempio, un protocollo di interrogazione, una stima di durata comunicata inizialmente dal partecipante, un messaggio con cui il partecipante comunica autonomamente che ha completato la propria esecuzione.

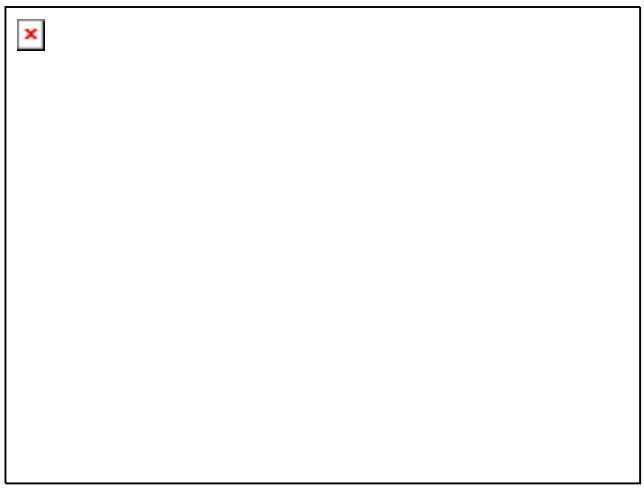

Figura 15

RR6. Overbooking con Notifica

Per minimizzare il numero di Abort, nel caso in cui un servizio utilizzi l'overbooking delle risorse, è necessario che il richiedente sia notificato circa le risorse assegnate definitivamente, ovvero circa le risorse la cui prenotazione non risulta ulteriormente valida. In questo modo si potranno evitare conferme per transazioni dall'esito negativo e scontato: il cliente richiederebbe l'eventuale annullamento solo quando strettamente necessario, e non, ad esempio, perché pensava che l'albergo fosse ancora disponibile quando non lo era più.

Ovviamente molti servizi, tra cui quelli di vendita on-line, vorranno applicare la tecnica dell'overbooking ad insaputa dell'utente. Ma esistono molte altre situazioni, ad esempio servizi che svolgono aste on-line, oppure servizi analoghi a quelli descritti nello scenario "1.3.5 Financial Trading", in cui è conveniente che l'utente ne sia consapevole, anche dal punto di vista dell'esecutore.

RP6. Conoscenza dello "stato" di composizione del richiedente

Essendo sempre valido il Principio di Autonomia del richiedente, e a maggior ragione essendo possibile la Composizione Dinamica, i tempi di composizione prima di un'eventuale conferma possono essere molto lunghi. Affinché un partecipante possa distinguere tale situazione da possibili malfunzionamenti del servizio richiedente, e conseguentemente possa prendere decisioni appropriate, è desiderabile che sia fornito un meccanismo con cui poter conoscere lo stato corrente del richiedente. In proposito potrebbero essere utilizzate tecniche analoghe a quelle citate per il requisito RR5.

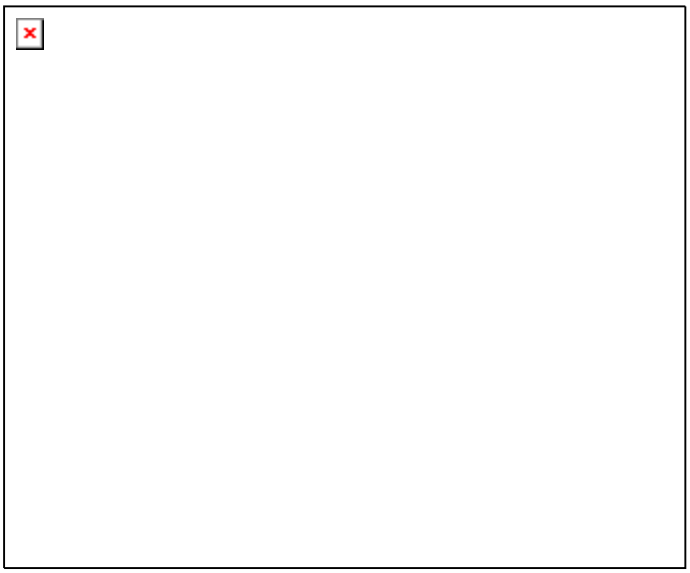

Figura 16

RC5. Recovery anche nella fase di Composizione

Detta "fase di composizione" la fase precedente alla conferma del richiedente, e data la caratteristica di lunga durata delle transazioni business, potrebbe risultare troppo oneroso, in caso di *failures*, ripetere tale fase dall'inizio. Del resto, anche il requisito sulla Composizione Dinamica ha come obiettivo quello di evitare la ricomposizione dall'inizio nel caso in cui un partecipante alla transazione non sia disponibile.

Sarebbe pertanto opportuno prevedere protocolli che consentano, o agevolino, il Recovery anche nella fase di Composizione.

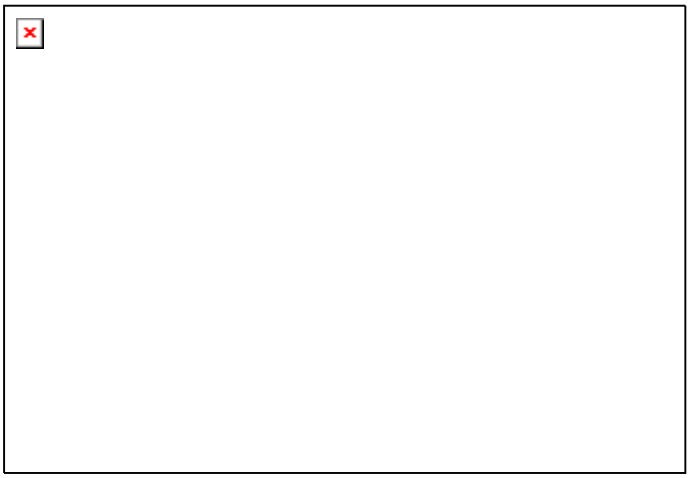

Figura 17

RC6. Molteplici modelli per la Sicurezza[44]

Eventualmente il requisito sulla Sicurezza ("RC1 **Comunicazioni Sicure**") deve poter essere rilassato, soprattutto quando collida con altri requisiti, quali, ad esempio, il requisito per un partecipante di fornire massima Disponibilità del proprio servizio. Un partecipante dovrebbe poter supportare un livello di sicurezza consono alle proprie esigenze; ad esempio, per poter avere un elevato livello di Disponibilità, deve poter richiedere solo l'autenticazione del cliente, e mediante uno specifico protocollo per l'autenticazione. Del resto, il cliente deve poter utilizzare i servizi che gli garantiscano il voluto livello di sicurezza, ad esempio secondo le proprie necessità di privacy, secondo l'importanza delle operazioni richieste (non rifiutabilità), etc. Pertanto, è necessario che il protocollo di coordinamento supporti molteplici modelli per la Sicurezza.

[44] Ad esempio, nello scenario di "1.3.6 Micro-paying", per fornire la massima disponibilità del servizio fornitore, il livello di sicurezza potrebbe essere minimo. Al contrario, nello scenario "1.3.5 Financial Trading", il livello di sicurezza sarà certamente molto elevato.

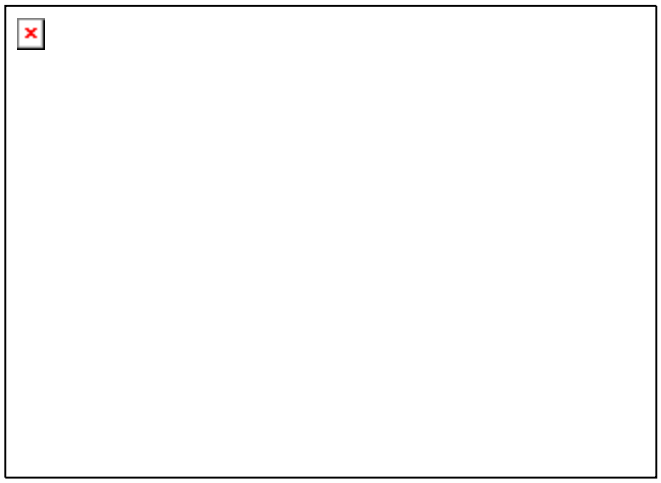

Figura 18

RR7. Contratti Vincolanti relativi all'Atomicità[45]

Affinché sia garantita l'Atomicità, non è sufficiente definire solo opportuni meccanismi di coordinamento e di Recovery, ma è anche necessario che i servizi esecutori rispettino determinati vincoli, solitamente dipendenti dal particolare protocollo utilizzato per garantirla. Richiedere "Garanzia di atomicità", significa proprio stabilire quali siano i vincoli da rispettare, ovvero il "contratto" che, una volta sottoscritto, i partecipanti si impegnano a rispettare.

Tuttavia, è possibile che i servizi esecutori spesso non vogliano fornire questo tipo di garanzia. In tal caso il richiedente potrebbe accontentarsi di essere messo sempre al corrente dell'eventualità che una transazione non avvenga in maniera atomica. Come per la garanzia di atomicità, anche nel caso in esame è necessario che i partecipanti rispettino dei vincoli dipendenti dal particolare protocollo di coordinamento adottato, che tuttavia potranno essere meno onerosi del caso precedente.

[45] Per un approfondimento vedi anche "1.4.4 Considerazioni" a pag87.

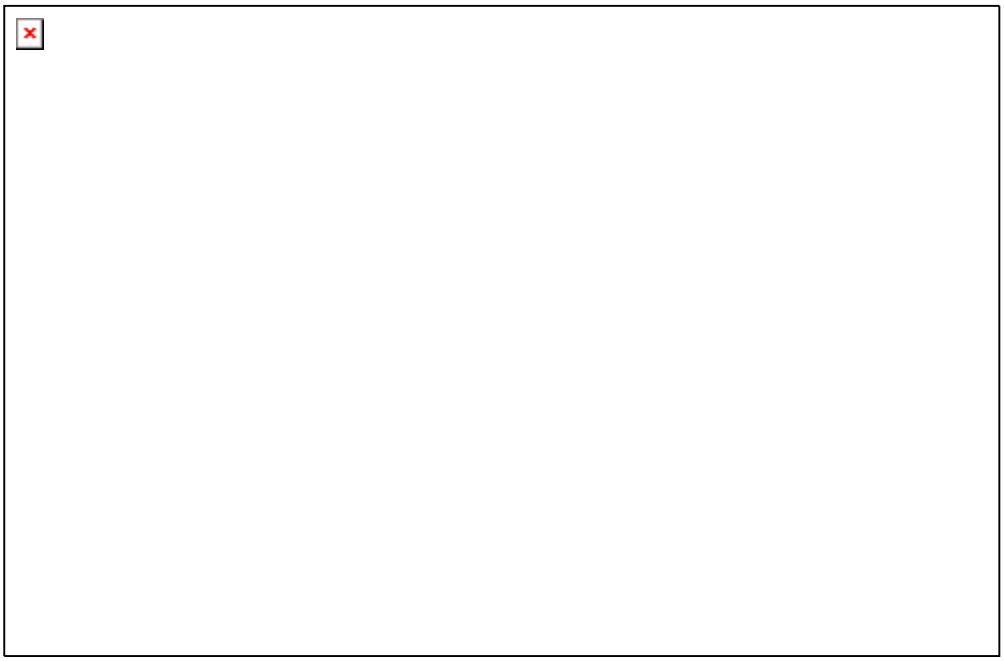

Figura 19

RR8. Procedura di Annullamento

In accordo al Principio di Autonomia, un utente può sempre comporre una transazione business mediante servizi che garantiscono atomicità e servizi che non forniscono questa garanzia. In tal caso, ovviamente, la transazione potrà sempre concludersi con un risultato non atomico. Eventualmente alcuni servizi, per andare incontro alle esigenze del cliente, potrebbero fornire volutamente una Procedura di Annullamento, ovvero una procedura mediante la quale sia possibile richiede l'annullamento di un'operazione precedentemente confermata, entro un certo tempo dalla conferma, cosicchè l'utente potrebbe recuperare in extremis. In altri casi, come accade in presenza di particolari normative legislative, quali, ad esempio, il *diritto di recesso* (vedi scenario "1.3.2 Home entertainement system"), fornire la possibilità di annullamento entro un certo tempo potrebbe, invece, essere di estrema convenienza per il servizio esecutore, che così potrebbe evitare inutili spese, quali, ad esempio, spese di spedizione per una merce che non sarà mai ritirata.

Seppure spesso i servizi vorranno fornire questo tipo di procedure separatamente dalle procedure di utilizzo ordinario, anche per scoraggiarne il loro utilizzo, in molti altri casi potrebbe essere desiderabile un meccanismo standard che ne consenta un utilizzo in maniera automatica.

RR9. Contratti Vincolanti relativi all'Isolamento

Come per la garanzia di Atomicità, anche per avere garanzia di isolamento non è sufficiente definire solo opportuni meccanismi, ma è anche necessario che i servizi esecutori rispettino i vincoli da essi imposti. Pertanto, anche relativamente all'isolamento, è opportuno che siano definiti specifici Contratti.

RR10. QoS a Runtime (A-I-S-Overbooking-etc.)

Secondo quanto sinora specificato, e in accordo al Principio di Autonomia, ogni partecipante potrà fornire diversi livelli di "Quality of Service" (QoS). Infatti, come discende dalla discussione dei requisiti trattati finora, un partecipante potrà fornire "garanzia di atomicità", "garanzia di segnalazione per risultato non atomico", "atomicità e Procedura di Annullamento", potrà supportare un certo modello per la sicurezza, potrà concedere risorse in Overbooking o in "Overbooking con notifica", e potrà supportare differenti livelli di isolamento.

Il richiedente può venire a conoscenza del livello di QoS fornito da un particolare servizio in molti modi, compresi quelli tradizionali (cartaceo, telefonico, dal sito web della compagnia, etc). Tuttavia, affinché sia possibile una composizione automatica dei servizi, oppure, più semplicemente, affinché sia possibile realizzare applicazioni interattive rispetto alla composizione dei servizi, ovvero applicazioni in cui l'utente possa scegliere i servizi da comporre dinamicamente in maniera rapida ed efficiente, sarebbe desiderabile un meccanismo standard utilizzabile a tempo di esecuzione (run-time).

Inoltre, essendo possibile che un servizio esecutore supporti più livelli di QoS contemporaneamente, ad esempio in funzione dei costi che il cliente vorrà sostenere, si rende necessario un meccanismo standard con cui il richiedente possa indicare quale tipo di servizio voglia utilizzare.

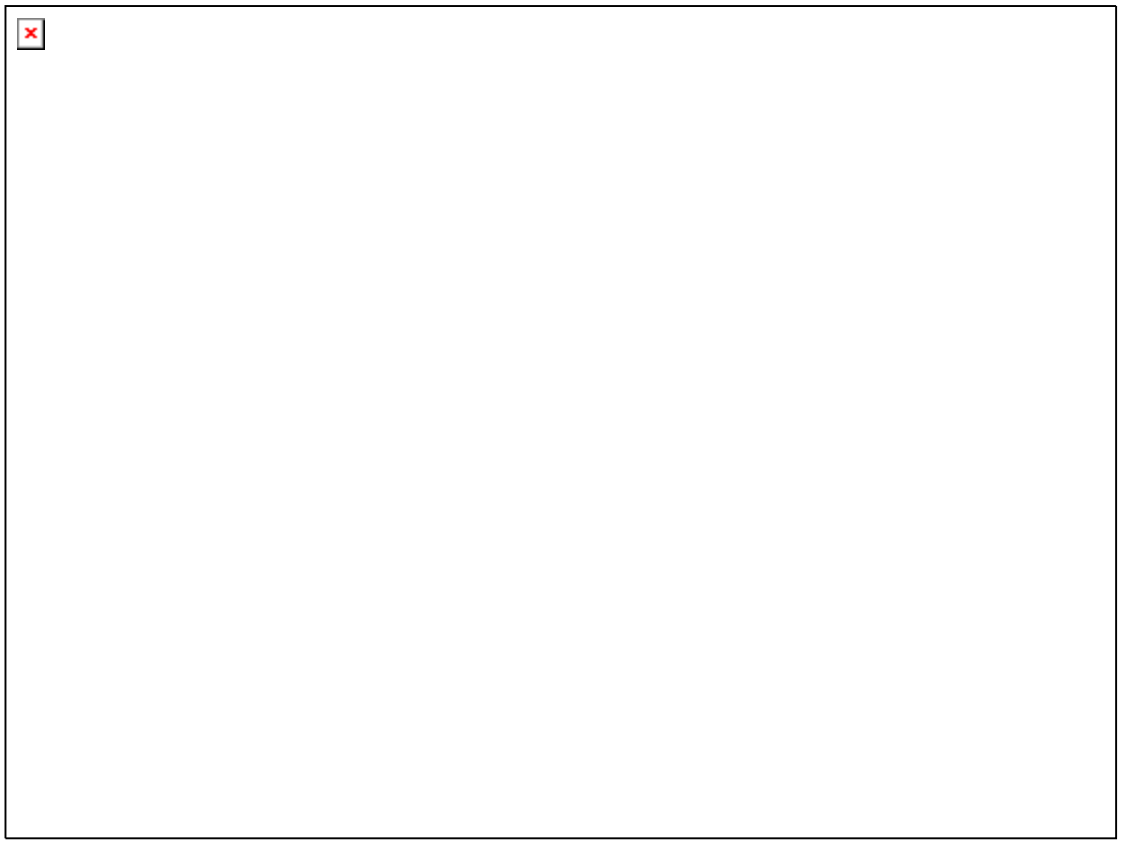

Figura 20

1.4.4 Considerazioni

Relazione tra Composizione Dinamica e Atomicità

Il requisito di *composizione dinamica* potrebbe essere confuso con il rilassamento dell'atomicità. Tuttavia, come approfondiremo meglio in "3.1 Modelli tradizionali: analisi rispetto ad Atomicità e Isolamento", la garanzia di atomicità dipende dal particolare protocollo di coordinamento adottato. Tale garanzia è, infatti, strettamente legata alla possibilità di revoca della partecipazione concessa ad un esecutore, in particolare, ai vincoli imposti dal protocollo di coordinamento sul momento in cui tali revoche sono consentite. La Composizione Dinamica permette invece di variare le operazioni (i partecipanti) dinamicamente prima di confermare, ma una volta che la transazione sia confermata con certe operazioni (certi partecipanti), l'atomicità potrà essere rispettata o meno.

Modelli Transazionali Estesi

La tecnica dell'Interposizione adotta il modello delle **Transazioni Innestate** (Nested Transactions). Quest'ultimo sostituisce al modello transazionale *piatto* quello *gerarchico*, concedendo che un'operazione transazionale possa essere essa stessa una nuova transazione. Si parla così di Modello Transazionale Esteso[46]. Tuttavia esiste una leggera differenza. Il modello delle Transazioni Innestate rappresenta infatti un modello formale per correlare una transazione "padre" a delle transazioni "figlie", e richiede che tutte completino con lo stesso risultato.

Ad esempio, nello scenario "1.3.3 Arranging a meeting" abbiamo applicato il concetto di *composizione dinamica*, grazie al quale è possibile considerare un'unica transazione in cui gli slots *lockati* vengono rilasciati via via che la composizione avanza, semplicemente annullando le corrispondenti operazioni che li hanno determinati. In questo modo, solo la composizione finale, costituita dall'ultima operazione (indicante lo

[46] Esistono diversi Modelli Transazionali Estesi (vedi Riferimenti[35]).

slot dell'incontro), dovrebbe essere confermata. Alternativamente, utilizzando le Transazioni Innestate, si potrebbe associare una sotto-transazione ad ogni rounds, al termine del quale risulterebbe assegnato (e lockato) un insieme di slots. La sotto-transazione successiva richiederebbe l'assegnamento di un sotto insieme degli slots assegnati dalla sotto-transazione precedente, e così via. In questo modo, ad ogni passo, l'insieme degli slots iniziale verrebbe ridotto, purchè ogni servizio esecutore, alla fine di una sotto-transazione, si preoccupi del rilascio degli slots non confermati: ciò è possibile proprio perché il servizio esecutore è consapevole che le varie sotto-transazioni, facendo parte di un'unica transazione principale, sono tra loro correlate. Il processo si ripeterebbe fino a quando non sia richiesto il completamento con successo della transazione principale. In caso contrario, tutti gli slots precedentemente assegnati dovrebbero essere rilasciati.

La tecnica dell'Interposizione, invece, non fa riferimento al risultato delle transazioni, che dunque potrà essere comune o meno, ma alla modalità di coordinamento: ogni transazione "figlia" ha un proprio coordinatore, e tutti i coordinatori relativi a transazioni "figlie" sono a loro volta coordinati dal coordinatore della transazione "padre". Generalizzando questo concetto, e assumendo che il coordinatore sia "parte" del richiedente, si può affermare che la tecnica dell'Interposizione prevede la sovrapposizione dei ruoli di richiedente ed esecutore.

Interposizione e *deadlock strutturali*

L'introduzione della tecnica dell'Interposizione comporta un problema. Nel *modello piatto* abbiamo infatti visto che, utilizzando il locking per l'isolamento, sono possibili deadlock distribuiti. Questi possono essere considerati come eventi occasionali, poiché dipendono dal fatto che almeno due transazioni, che siano eseguite contemporaneamente, richiedano le stesse risorse e in ordine inverso. Con l'Interposizione e il modello gerarchico sono possibili anche deadlock distribuiti di tipo *permanente*. Questi, a differenza dei primi, possono verificarsi anche durante l'esecuzione di una singola transazione. In proposito si osservi l'esempio mostrato in Figura 21.

Il Subordinato8, per svolgere il proprio lavoro come esecutore all'interno della transazione, richiede una risorsa già bloccata dal richiedente della stessa transazione. Chiaramente, abortire la transazione come si fa per i deadlock occasionali non ha senso, poiché il problema è strutturale, e riguarda una singola transazione, che dunque non potrà mai completare con successo.

L'adozione della tecnica dell'Interposizione in un ambiente multi dominio può sempre portare ad una composizione che presenti il problema, e questo in maniera del tutto trasparente all'utente: ciò è dovuto al fatto che ad ogni livello, nell'albero della transazione, la composizione dei servizi avviene, in accordo alla proprietà di autonomia del controllo di ognuno di essi, senza conoscere come questi siano composti a loro volta (incapsulamento). Per applicazioni intra dominio il problema può ancora presentarsi ma, essendo in tal caso l'Interposizione applicata tra servizi dal comportamento noto, sarebbe da considerarsi più propriamente come un errore di programmazione.

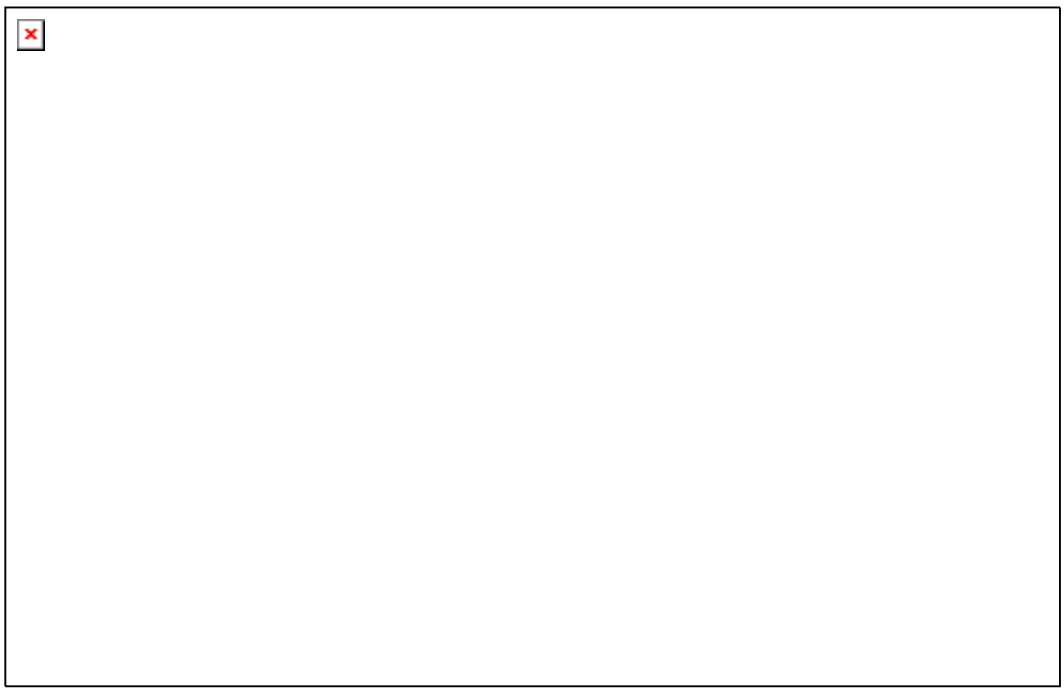

Figura 21 Esempio di deadlock *permanente*

Fortunatamente, per individuare questo problema è possibile adoperare una tecnica molto semplice ed elegante: basta attribuire ad ogni transazione un identificatore univoco, che sarà ereditato anche da ogni sotto-transazione, e un timeout. Quando scada

il timer, è sufficiente che il servizio controlli semplicemente l'eventualità che esistano delle risorse su cui siano bloccati richiedenti appartenenti alla stessa transazione. Se questo è il caso, saremo certamente in presenza di un deadlock permanente. Evidentemente, una volta individuato il problema, serve anche un meccanismo per segnalarlo al richiedente (compositore) della transazione.

Applicabilità della Procedura di Annullamento

La possibilità, per l'utente, di annullare in extremis un'operazione già confermata, può essere conveniente in molte situazioni. Tuttavia, per un partecipante, questo tipo di procedura sarà applicabile convenientemente solo in alcuni casi, in particolare per applicazioni in cui le risorse business abbiano un valore poco variabile. Per applicazioni dal valore delle risorse fortemente variabile, infatti, è molto probabile che ci siano delle "perdite", e così, qualsiasi protocollo venga utilizzato, difficilmente un fornitore di servizio che abbia riservato (bloccato) delle risorse per una prenotazione, concederà all'utente la possibilità di annullare detta prenotazione a "costo zero". Per evitare questi svantaggi, è necessario utilizzare delle tecniche di Overbooking, con cui le risorse possano essere prenotate, ma senza riserve.

Caratteristiche di una buona Specifica e *Contratti*

Le Specifiche che prenderemo in esame nella seconda parte della tesi, prima di essere la "soluzione" ad un problema, sono dei documenti di specifica, e come tali dovrebbero essere corretti, consistenti, non ambigui, completi … ovvero dovrebbero soddisfare tutte le proprietà di un buon documento di Software Requirements Specification (SRS)[47]. Una cattiva Specifica solitamente produce cattive Implementazioni, e dunque può inficiare la stessa validità della soluzione in essa proposta. Tanto più che le Specifiche in questione si rivolgono ad un target molto esigente: quello dei servizi business.

In generale, una relazione business tra servizi indipendenti nella gestione, nel controllo e soprattutto nella proprietà, richiede la definizione preliminare di un

[47] così come dettato dalle buone norme dell'Ingegneria del Software (le proprietà di un buon SRS sono state formalizzate dall'IEEE)

Contratto, diciamolo *contratto business*, dipendente dal particolare servizio offerto, e che le parti dovranno sottoscrivere per realizzare il business in questione.

Tuttavia, nell'ambito di una transazione business, affinché questa possa avere determinate caratteristiche nel suo complesso, è necessario definire anche un altro tipo di Contratto, questa volta indipendentemente dalla particolare logica applicativa. Infatti, come abbiamo già osservato, per ottenere un risultato atomico, o per avere "garanzia di segnalazione per risultato non atomico", non è necessario solo definire un appropriato protocollo di coordinamento, che potrebbe sempre non essere rispettato, ma anche dei vincoli ben precisi.

Ad esempio, assumendo che per l'atomicità sia utilizzato il 2PC precedentemente descritto, il Contratto dovrebbe fornire risposte precise a domande del tipo:

- dopo un *prepared*, per quanto tempo deve rimanere vincolato il partecipante?
- cosa deve accadere nel caso in cui tale tempo non sia o non possa essere rispettato? ad esempio, il partecipante deve accertarsi che un messaggio di *cancelled* sia pervenuto al *richiedente*?
- se il messaggio di riscontro per il *cancelled* non dovesse arrivare entro un certo tempo, il partecipante può ritenersi esonerato dal Contratto?
- un partecipante che invia un *prepared*, dichiara che potrà confermare la transazione per quanto tempo?
- cosa deve accadere nel momento in cui un partecipante "cada", dopo aver inviato un *prepared*, per un tempo maggiore a quello per cui è disposto ad aspettare il richiedente (o il coordinatore) ?

Solo con un Contratto preciso, in caso di contenziosi, potranno essere intraprese eventuali azioni legali contro parteners scorretti, e questo a prescindere dal modello per la Sicurezza: a cosa serve la non rifiutabilità, se poi non si può attribuire alcun obbligo al significato di un messaggio non rifiutabile?

I contratti business potrebbero essere definiti di volta in volta, all'occorrenza, tra richiedente ed esecutore, ma, solitamente, sono predefiniti da quest'ultimo, e mostrati al richiedente on-line subito prima di consentire l'utilizzo del servizio (si pensi ai tanto

diffusi servizi di e-commerce). Così il richiedente dovrà semplicemente decidere se accettarli o meno.

Per quanto riguarda i contratti relativi al protocollo di coordinamento, che d'ora in avanti chiameremo per semplicità *contratti di protocollo*, si potrebbe pensare che sia applicabile lo stesso meccanismo, specialmente quando sia comunque necessario stabilire un contratto business.

Tuttavia, esistono delle importanti differenze tra i due tipi di contratto:

- Un contratto business dipende dalla particolare relazione business, variabile con l'applicazione e i servizi, mentre un contratto di protocollo dipende strettamente dal protocollo di coordinamento. Anzi, affinché un certo contratto di protocollo possa essere definito, è spesso necessario che il protocollo di coordinamento metta a disposizione opportuni meccanismi.
- Il mancato rispetto di un contratto business invalida l'accordo tra le sole due parti coinvolte nella rispettiva relazione business, mentre il non rispetto di un contratto di protocollo, in special modo quando riguarda l'atomicità, compromette il risultato di un'intera transazione. In questo caso può coinvolgere, dunque, anche i servizi che il contratto lo hanno rispettato. Ad esempio, una transazione richiede della merce al servizio di un fornitore e il trasporto ad un altro servizio: nel caso in cui l'ordinativo della merce non vada a buon fine, il cliente non vorrà sostenere il costo per un trasporto a vuoto.

Se, dunque, si lasciasse la definizione del contratto di protocollo ai servizi, non solo si richiederebbe ai rispettivi gestori o proprietari una perfetta e completa conoscenza del protocollo di coordinamento, e di conseguenza ai gestori o proprietari dei servizi richiedenti che dovranno interpretarli, ma si permetterebbe anche un proliferare di molteplici contratti, il cui utilizzo congiunto difficilmente potrà dare certezze al richiedente. Operando in questo modo, ne risulterebbe penalizzata l'interoperabilità, ovvero quella stessa caratteristica che rappresenta l'obiettivo ultimo di qualsiasi standard di coordinamento nell'ambito in questione.

E' dunque necessaria la definizione precisa di pochi contratti di protocollo, definiti una volta per tutte in maniera standard, e come suggerisce la stretta dipendenza tra contratti e protocolli, tale definizione dovrebbe coinvolgere gli autori di questi ultimi. Pertanto, è

lecito supporre che essi siano definiti come parte della Specifica del protocollo di coordinamento, essendo a quest'ultimo strettamente legati, o che siano definiti in un documento separato. L'importante è che siano definiti.

Evidentemente, potranno esistere più contratti, o più varianti di uno stesso contratto, ognuno associato ad un particolare livello di QoS.

In conclusione, assumendo che detti Contratti siano definiti all'interno della Specifica per il protocollo di coordinamento, questa dovrà essere valutata sotto diversi aspetti, e con la consapevolezza che ognuno di essi concorrerà parimenti alla bontà della soluzione finale:

- come **soluzione** particolare del problema: dovrà rispettare i requisiti precisati nei precedenti paragrafi, e dovrebbe farlo fornendo una soluzione semplice, ovvero con una complessità proporzionata al problema che va a risolvere.
- come **documento di specifica**: dovrà avere le caratteristiche di un buon documento di specifica, quali correttezza, completezza, non ambiguità, etc.
- come "**base contrattuale"** tra le Parti relazionate: dovrebbe stabilire diversi Contratti, in funzione dei differenti livelli di QoS supportabili da ogni servizio.

1.4.5 Ricapitolando

E’ necessario un **modello di coordinazione transazionale** per servizi Web che fissi la sintassi e la semantica dei messaggi scambiati. Tale coordinazione deve avvenire nel rispetto dei requisiti del richiedente e dei singoli partecipanti sopra definiti. In particolare, compatibilmente con gli altri requisiti, deve garantire la massima autonomia di partecipante e richiedente, supportando contemporaneamente diversi livelli di Qualità del Servizio (QoS) e definendo diversi contratti di protocollo. Deve inoltre sopperire alle caratteristiche del trasporto di inaffidabilità, fornendo protocolli robusti, e di lentezza, fornendo protocolli efficienti che riducano al minimo l’overhead introdotto (ottimizzazioni, minimo numero di messaggi scambiati, Interposizione, etc).

Infine, deve supportare l’esecuzione di transazioni in ambiente sicuro e, non meno importante, deve fornire pieno supporto per **Transazioni ACID** tradizionali.

In questo modo sarà possibile comporre servizi Web in maniera affidabile, e con strutture anche molto più complesse di quella mostrata nel Deployment Diagram in Figura 22.

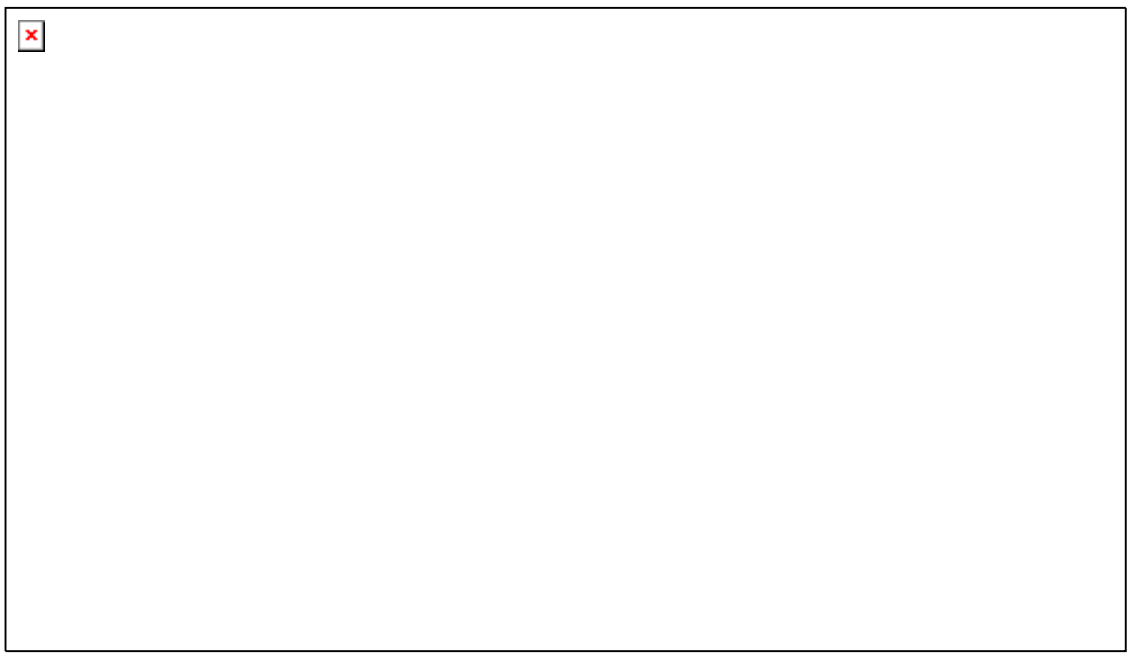

Figura 22 Esempio di Composizione di servizi Web

I requisiti precedentemente esposti non sono suddivisibili in requisiti essenziali e desiderabili, poiché non esiste un unico *committente*, e ogni requisito può essere di un tipo o dell'altro a seconda dei casi. Tuttavia, vale la pena sottolineare che esiste un principio, prima ancora di quello dell'Autonomia, che indiscutibilmente va considerato come essenziale: l'**Interoperabilità**. Un middleware di supporto che non rispetti questo principio in ambiente Web Services non ha nessuna ragione di esistere. Questo è maggiormente vero per le interazioni multi dominio, ma è vero anche per quelle intra dominio, poiché sempre più spesso capita che servizi utilizzati internamente vengano, prima o poi, esposti all'esterno (come accade oggi passando dalle intranet alle extranet). Del resto, quello dell'interoperabilità, è uno dei principi cardine su cui si fonda il successo stesso dei WebServices. Questo è il motivo per cui è assolutamente necessario parlare di **Standard**.

2 SOLUZIONI ATTUALI

Qualche anno dopo l'introduzione di TIP, uno dei primi lavori prodotti per supportare Transazioni Distribuite su Internet, l'OASIS iniziò lo sviluppo di **BTP**. Quest'ultimo, nell'anno 2001, quando iniziarono formalmente i lavori per la sua stesura, già si rivolgeva esplicitamente all'ambiente dei Web Services, seppure non in maniera esclusiva. Molti produttori hanno contribuito alla sua realizzazione, forse troppi. Tra questi ricordiamo BEA Systems, Hewlett-Packard, Sybase, Choreology, IONA, Oracle e Sun Microsystems. Alcuni di essi hanno infatti deciso in seguito di proporre delle alternative proprie, e nella versione attuale[48] della Specifica vengono citati solo autori dei gruppi BEA Systems e Choreology.

Più o meno nello stesso periodo venne sviluppata da autori del gruppo Intel, come membri W3C, la Specifica di **THP**. Non rappresenta un'alternativa alle altre Specifiche, bensì un prodotto complementare, da utilizzare congiuntamente ad esse.

Poco dopo, iniziò lo sviluppo di **WS-Transactions**, una Specifica composta da tre sotto-Specifiche. Tra i maggiori suoi contribuenti ricordiamo Microsoft, IBM e BEA Systems.

Data la perenne controversia tra il colosso Microsoft e quello della Sun Microsystems, non poteva mancare nello scenario una ulteriore proposta. Così, più di recente rispetto a BTP e WS-Transactions, un gruppo composto da Sun Microsystems, Oracle Corporation, Arjuna Technologies, Fujitsu e IONA Technologies, ha realizzato una nuova Specifica denominata **WS-CAF**. Essa viene proposta come un super set delle precedenti Specifiche, BTP e WS-Transactions, e come quest'ultima, è composta da tre sotto-Specifiche distinte.

Quelle elencate costituiscono tuttora le soluzioni proposte dai maggiori produttori mondiali per supportare Transazioni Distribuite in ambito dei Web Services. Le descriveremo e analizzeremo nel seguito di questo lavoro di tesi.

[48] "Business Transaction Protocol Version 1.0.9.5 BTP 1.1 Working Draft 05, 9-11-2004".

Nell'elenco abbiamo citato anche la Specifica "Transaction Internet Protocol" (TIP)[49], prodotta nel "lontano" 1998. Essa ha avuto uno scarso successo, probabilmente per due motivi principali. Il primo riguarda la soluzione proposta: TIP utilizza esclusivamente un protocollo 2PC tradizionale, e questo, nella community dei WebServices, viene ritenuto poco interessante. Il secondo motivo riguarda il target a cui si rivolge la Specifica: TIP non si rivolge all'ambiente dei Web Services, ma, in maniera generica, alle Transazioni Distribuite su Internet. In effetti, anche le soluzioni proposte dalle altre Specifiche, in qualche maniera, utilizzano un protocollo 2PC, ma, a differenza di TIP, lo inseriscono nel più ampio contesto delle Transazioni Business, e tutte si rivolgono esplicitamente ai servizi Web.

Per gli stessi motivi, abbiamo deciso di escluderlo dalla presente tesi. In ogni caso, non avrebbe dato un contributo significativo al nostro lavoro, sia perchè il protocollo 2PC è stato già ampiamente discusso nella sezione "1 Il problema", e sia perchè tale protocollo sarà ulteriormente trattato all'interno della descrizione delle altre Specifiche. Tratteremo invece tutte le altre Specifiche citate.

I WebServices, ad oggi, rimangono comunque una tecnologia nel complesso ancora molto immatura, poiché può parlarsi di standards, peraltro in continua evoluzione, solo ai livelli più bassi degli *stacks di protocolli*. Sarà il mercato a decidere quali Specifiche diventeranno realmente degli standards. L'argomento è infatti tuttora molto controverso, e i produttori organizzano meetings per ottenere feedback dagli utilizzatori, in base ai quali migliorare ed estendere i propri prodotti. Allo scopo i produttori forniscono librerie di "prova", insieme ad esempi già implementati, utilizzabili all'interno dei framework di sviluppo offerti per Web Services. Dunque, possiamo considerarci ancora in una fase di "beta testing".

Microsoft mette a disposizione degli utenti il pacchetto "Web Services Enhancements (WSE) 3.0", un add-on per "Microsoft Visual Studio® 2005 Beta 2" e "Microsoft .NET Framework 2.0 Beta 2", che supporta molte delle nuove Specifiche emergenti, compresa WS-Transactions. Tuttavia, essendo le Specifiche non ancora standardizzate, il supporto viene considerato una "pre-released" software: per il momento Microsoft

[49] Riferimenti[23]

propone tali tecnologie solo per avere feedback dagli utenti della community e poter migliorare il prodotto.

IBM si comporta praticamente allo stesso modo. Fornisce quelle che chiama “tecnology preview” con “Web Services ToolKit for dynamic e-business” e “Business Process Execution Language for Web Services Java Runtime”. Tuttavia, nel caso IBM, sono supportate attualmente solo WS-Coordination e WS-AT, ed inoltre, il supporto non è pieno (ad esempio il recovery non è supportato nel caso di caduta del server). Possono essere utilizzate in un programma J2EE tramite l’interfaccia Java Transaction API (JTA). Il motore sottostante, per garantire le proprietà ACID, è Java Transaction Service (JTS), con il limite che le suddette proprietà sono mantenute solo nelle interazioni con altre implementazioni conformi J2EE.

Per quanto riguarda WS-Transactions, e in generale per tutte le Specifiche sull’argomento, è possibile trovare in rete molte diatribe. Ad esempio, nello specifico di WS-Transactions, è possibile consultare il gruppo di discussione all’indirizzo:

http://groups.yahoo.com/group/WS-TX-Workshops/

Arjuna propone invece qualcosa di più “completo”. Il suo middleware “ArjunaTS Web Services Transaction Support” supporta pienamente WS-Transactions, utilizzando le API “Java API for XML Transactioning (JAX-TX)”. Purtroppo non supporta contemporaneamente le Specifiche sulla Sicurezza. Stranamente, non supporta WS-CAF, seppure il gruppo sia direttamente implicato nello sviluppo del prodotto.

Choreology propone “Cohesions 3.0”: supporta sia BTP che WS-Transactions, quest’ultimo nella versione 1.0. Come per Arjuna, non supporta contemporaneamente le Specifiche sulla Sicurezza.

BEA propone “Collaxa WSOS 2.0”, un add-on per “WebLogic Platform 7.0”: la versione di WS-Transactions è la 1.0, e valgono le solite limitazioni sulla Sicurezza.

IONA propone Artix, ovvero una libreria di API per C++ e Java, ma supporta solo WS-C e WS-AT, senza Sicurezza.

HP propone “HP-WST”, un pacchetto di librerie Java per BTP. Tuttavia l’HP è attualmente in una fase transitoria, e il supporto per alcuni suoi prodotti, compreso quello citato, è garantita solo per un periodo limitato di tempo. I clienti HP vengono indirizzati verso i prodotti BEA Systems, con cui l’HP ha stipulato un accordo.

Infine, in seguito ad una buona ricerca sul Web, non siamo riusciti a trovare tools di supporto per THP. Probabilmente esisteranno, ma è certo che, attualmente, non sono facilmente reperibili in rete.

Nel seguito di questa seconda sezione della testi descriveremo tutte le Specifiche citate (esclusa TIP). La descrizione sarà finalizzata al nostro obiettivo primario, ovvero quello di confrontare e valutare le varie soluzioni proposte.

WS-Transactions, come WS-CAF, fa parte di una più ampia architettura per WebServices costituita da un insieme di protocolli organizzati in uno *stack*. Per quanto riguarda WS-Transactions descriveremo marginalmente anche altre Specifiche ad essa strettamente connesse, ed appartenenti allo *stack* immaginato da Microsoft-IBM. Questo permetterà al lettore una maggiore comprensione delle varie problematiche che si presentano nell'utilizzo della tecnologia in questione.

Tutte le Specifiche che tratteremo fanno riferimento in qualche maniera alle tecnologie per Web Services, e così tutti i messaggi dei protocolli vengono definiti in "XML Schema", e vengono fornite le descrizioni WSDL di tutti i servizi. Tali schemi e descrizioni sono parte integrante delle Specifiche in questione, ma in questa tesi non verranno prese in considerazione, poiché, per il confronto, è necessario un livello di astrazione maggiore. Il lettore interessato può consultarle direttamente sui documenti originari elencati nella sezione "Riferimenti".

Le Specifiche, pur affrontando gli stessi problemi e spesso proponendo soluzioni analoghe, utilizzano differenti schemi e convenzioni descrittive, e mettono in risalto particolari aspetti piuttosto che altri. Per effettuare un'analisi migliore, cercheremo invece di uniformare la trattazione (secondo lo schema descritto nel seguito), cosicché sia possibile mettere in risalto le differenze vere, quando presenti.

Nella descrizione ci aiuteremo con dei modelli UML. Un modello UML sarà estrapolato da ogni Specifica trattata, e ne riassumerà i Concetti fondamentali, costituendone un punto di riferimento per la descrizione.

Si vuole ora sottolineare il valore dei suddetti modelli UML. Seppure si sia tentato di mantenerli il più possibile coerenti con quanto specificato nei documenti di partenza, essi vanno considerati in accordo con l'obiettivo per il quale sono stati ottenuti, e dunque non come una rappresentazione formale e completa delle Specifiche in

questione, bensì come una rappresentazione minimale, costruita sui concetti fondamentali e su quegli aspetti delle Specifiche di particolare interesse per un'analisi comparativa tra le varie soluzioni. In altri termini, non esiste una relazione biunivoca tra la soluzione di una Specifica e il modello con cui la rappresenteremo, poiché si collocano a diversi livelli di astrazione.

Sulla base di questa assunzione, è possibile che nei nostri modelli vengano utilizzati nomi[50] leggermente diversi da quelli originari, che compaiano delle nuove entità, quando la loro introduzione serva a rappresentare meglio dei concetti, ed è possibile che alcuni dettagli, non consoni al livello di astrazione adottato, siano completamente ignorati.

Inoltre, per non appesantire troppo la lettura, nella descrizione di quanto segue, come già fatto in precedenza, ci prenderemo qualche libertà di linguaggio, consapevoli che ciò potrà risultare in una minore precisione del testo.

Schema Descrittivo e Convenzioni

La trattazione delle diverse Specifiche è stata uniformata utilizzando lo "schema" seguente:

Introduzione: chi ha prodotto la Specifica, a chi si rivolge, quale problema si propone di risolvere, eventuali esempi del problema.

Modello Concettuale: modello delle sole entità, con relazioni, ruoli e attributi, ma senza metodi. La descrizione associata è composta da una prima parte riassuntiva e, quando necessario, da una seconda parte dettagliata in cui ogni entità viene specificata più approfonditamente.

Protocolli: descrive i vari protocolli e fornisce un "Modello delle Interfacce" che riassume tutti i messaggi previsti. I protocolli, nel senso di messaggi e sequenze di messaggi ammesse, vengono descritti insieme ad eventuali ottimizzazioni.

[50] Ovviamente, le stesse convenzioni sui nomi valgono anche per la descrizione associata a detti modelli.

Per meglio rappresentare le sequenze dei messaggi ammesse e il comportamento di alcune entità, sono forniti anche opportuni State Diagrams e/o Sequence Diagrams. Questi ultimi mostreranno le sequenze di messaggi, tra quelle ammesse, relative ai casi d'uso più semplici, mentre dagli State Diagrams sarà possibile capire le sequenze più complesse.

Failure Handling: messaggi di errore previsti.

Recovery Handling: come possono essere recuperate le relazioni in caso di *failures*.

Binding: per garantire interoperabilità, oltre alla semantica e alla sintassi, una Specifica dovrebbe definire come i messaggi siano rappresentati e codificati, e come possano essere trasmessi. Tale specificazione è detta "Carrier Protocol Binding" o più semplicemente Binding. Informazioni particolari riguardanti il protocollo del Trasporto, ad esempio relative alla gestione degli Indirizzi, sono trattate in questa sede.

Altre informazioni, non collocate precisamente, ma sparse nel suddetto "schema", sono:

Ruoli e responsabilità delle varie entità (servizi) presenti nel Modello Concettuale. Tali informazioni possono essere dedotte dal Modello Concettuale e dalla descrizione dei protocolli.

Relazioni con altre specifiche: quali altre Specifiche sono utilizzate. Come si colloca, eventualmente, nello Stack dei WebServices. Questo tipo di informazioni potrebbe essere trattato in un paragrafo a parte.

Affidabilità del Trasporto: come viene gestita la rispedizione dei messaggi.

Sicurezza: come viene gestito il problema della Sicurezza.

Responsabilità di Implementazione: chi deve implementare cosa.

CONVENZIONI

Nella descrizione delle Specifiche dovremo utilizzare spesso nomi propri per entità, ruoli, interfacce, messaggi, protocolli, modelli e così via. Per evitare un uso troppo frequente delle virgolette e per migliorare la leggibilità del testo, utilizzeremo sempre il *corsivo* per indicare i messaggi di protocollo, e spesso il "Maiuscolo" negli altri casi.

Inoltre, frequentemente sostituiremo i nomi inglesi originari con i corrispondenti nomi "italianizzati", utilizzati come sinonimi dei primi, per una maggiore scorrevolezza della lettura.

Infine, quando possibile, nei modelli UML distingueremo con colori diversi:

- Giallo: entità già definite in un altro modello ed "estese" dal modello corrente.
- Verde: nuove entità (non definite in altri modelli) che possono essere associate al ruolo di *esecutore*.
- Rosa: nuove entità (non definite in altri modelli) che possono essere associate al ruolo di *richiedente*.

2.1 Business Transaction Protocol (BTP)

La Specifica di BTP nasce dalla collaborazione di BEA Systems, Hewlett-Packard, Sybase, Choreology, IONA, Oracle e Sun Microsystems, per citarne solo alcuni, i quali, formando un comitato dedicato all'interno dell'OASIS, hanno formalmente avviato i lavori il 13 marzo 2001. La prima versione della Specifica è stata approvata, con voto unanime, il 16 maggio 2002. Tuttavia nella versione attuale[51], quella che tratteremo in questa tesi, vengono citati solo autori appartenenti ai gruppi di BEA Systems e Choreology, seppure tale versione sia praticamente identica alla prima.

Loro intento era quello di progettare un protocollo per la coordinazione di lavori applicativi tra più partecipanti la cui proprietà, e il cui controllo, appartengono ad organizzazioni autonome. In tale contesto, gli autori ammettono la forte esigenza di utilizzare modelli per la Sicurezza, ma delegano la questione ad altre Specifiche. Così, la versione di BTP qui presa in esame, presuppone un dominio di forte fiducia tra i partecipanti, e non aderisce esplicitamente a nessuno standard per la Sicurezza.

BTP definisce un particolare protocollo a due fasi per supportare transazioni di tipo Atom, nelle quali il risultato voluto dall'applicazione è noto a priori (tutti o nessuno), e transazioni di tipo Cohesion, nelle quali l'applicazione può intervenire per decidere selettivamente quali lavori debbano essere confermati.

Definisce il ruolo degli attori, i messaggi che tra questi possono essere scambiati, e gli "obblighi" che tali attori devono soddisfare, basandosi su un approccio minimalista e permissivo per il quale sono evitati vincoli implementativi. Quando possibile, cerca di evitare dipendenze da altri standard e così, per garantire la massima interoperabilità, non impone nessun protocollo per il Trasporto. Garantisce inoltre un'alta flessibilità per l'implementazione dei partecipanti, ad esempio nei riguardi della *modalità di inversione*

[51] La Specifica del protocollo BTP che si va a discutere è riportata in Riferimenti[3].

di una transazione atomica, che può eventualmente essere ottenuta anche mediante un'operazione di **compensazione**[52].

BTP definisce il concetto di **Relazione Business** come: "una qualsiasi informazione di stato condivisa e distribuita tra le parti, che sia soggetta a vincoli contrattuali accettati dalle parti stesse."

In tale definizione, il *contratto* ha un significato specifico per la particolare applicazione business. Ad esempio, per un'applicazione di compra-vendita on-line, la *relazione business* sarà costituita da tutte le informazioni contenute in un ordine di acquisto, mentre il contratto stabilirà il relativo significato, i termini di validità, la sequenza dei messaggi necessaria per scambiare dette informazioni, e tutto quanto sia necessario specificare per garantire la corretta esecuzione dell'acquisto.

BTP definisce pure il concetto di **Transazione Business**: "un cambiamento **consistente** nello stato di una *relazione business* tra due o più parti".

In questa, i cambiamenti di stato ammissibili, sono da intendersi come parte integrante del *contratto* stabilito tra le parti.

Le definizioni date non dicono volutamente nulla sul tipo di *business*, poiché BTP assume che questo possa essere di qualsiasi tipo. Ad esempio, il business può anche concernere un'attività *no-profit*, la concessione di un'autorizzazione, l'esecuzione di una verifica, una richiesta di pubblicazione, o qualsiasi altra generica interazione che possa essere automatizzata e che richieda un alto livello di confidenza tra le parti coinvolte.

Scopo di BTP è proprio quello di coordinare dette Transazioni Business, affinché i cambiamenti di stato, causati dallo scambio di messaggi applicativi, possano avvenire in maniera consistente, come da definizione.

A tal proposito, i servizi coinvolti nella Transazione Business devono rispettare delle regole ben precise, dettate dal protocollo BTP, affinché siano dei servizi *conformi BTP*.

Ad esempio, considerando lo scenario "1.3.4 Manifacturer-Supplier-Shipper", un'interazione tra il manifatturiere e il fornitore è rappresentata dall'invio di un ordinativo (un messaggio applicativo), a cui il fornitore può reagire controllando la

[52] Il concetto di *compensazione* è stato introdotto nello scenario "1.3.4 Manifacturer-Supplier-Shipper", e sarà chiarito in "3.1 Modelli tradizionali".

validità della richiesta, la disponibilità del materiale, ed in caso affermativo, riservando il materiale a nome del manifatturiere, e inviando un messaggio di risposta. Quando il manifatturiere accetta definitivamente l'acquisto, invia un messaggio BTP di conferma. In questo caso il fornitore sta offrendo un servizio conforme BTP.

In generale, per soddisfare il Contratto stabilito in una Relazione Business, un servizio conforme BTP deve supportare, in qualche maniera, richieste relative ai seguenti tre *Effect*:

- **Provisional Effect**: cambiamento provvisorio o tentativo di cambiamento dello stato della Relazione Business.
- **Final Effect**: conferma del tentativo di cambiamento.
- **Counter Effect**: annullamento del tentativo di cambiamento.

Il significato concreto di tali "effetti" dipende dalla particolare applicazione e dalla particolare Relazione Business. Nell'esempio, la riserva momentanea dei prodotti ordinati può corrispondere al Provisional Effect, mentre la registrazione definitiva dell'acquisto al Final Effect.

Ci sono diversi modi di implementare detti Effect, ma dal punto di vista di BTP essi sono completamente equivalenti. Così, non viene fornito nessuno strumento esplicito per renderli visibili al *richiedente* della transazione.

	Approccio conservativo	**Approccio compensativo (ottimistico)**	**Approccio tipico dei DBMS (intermedio)**
Provisional Effect	Memorizza il cambiamento voluto senza effettuarlo realmente	Effettua i cambiamenti rendendoli visibili, ma memorizza le informazioni per effettuare l'undò.	Memorizza lo stato originale, previene l'accesso esterno, ed effettua i cambiamenti
Final Effect	Effettua realmente i cambiamenti	Cancella le informazioni di undò	Riabilita l'accesso esterno
Counter Effect	Cancella i cambiamenti memorizzati	Effettua le azioni di undò	Ripristina lo stato originale e riabilita l'accesso esterno

Tabella 1 Alcune modalità implementative degli "Effect"

Alcuni approcci implementativi per gli Effect, suggeriti dalla Specifica, sono mostrati in Tabella 1. Le alternative mostrate non sono le sole, poichè BTP nulla impone in proposito: esse possono essere combinate e/o variate; inoltre, lo stato visibile prima della conferma o dell'annullamento, può essere differente sia dallo stato iniziale che da quello finale.

Il Couter Effect, soprattutto nell' *approccio compensativo*, può avere diversi significati. Può significare, ad esempio, cancellazione dei cambiamenti mediante una precisa *inversione*, oppure effettuazione di specifiche operazioni che in qualche modo compensino, rendano buoni, o allevino gli effetti di un'operazione già eseguita.

BTP si propone, dunque, di coordinare le transizioni tra gli stati applicativi corrispondenti a detti Effect, e per farlo, specifica un particolare protocollo a due fasi.

La logica di coordinamento, nella sostanza, è molto semplice, poiché corrisponde a quella di un protocollo *2PC generlalizzato*.

Nella prima fase, in seguito alla ricezione di un messaggio applicativo che specifica il cambiamento di stato voluto, ogni partecipante riceve un messaggio BTP di *prepare*, che richiede l'esecuzione del corrispondente Provisional Effect. Alla ricezione di tale messaggio il partecipante verifica che sia in grado di eseguire la successiva richiesta, sia essa di Final o di Counter Effect, e comunica tale capacità ad un'entità coordinatrice centrale.

Nella seconda fase, mediante un ulteriore messaggio BTP di *confirm* o di *cancel*, l'entità coordinatrice richiede rispettivamente il Final o il Counter Effect, determinando così la conferma o l'annullamento del precedente "tentativo di cambiamento".

2.1.1 Modello Concettuale

Nella figura seguente viene mostrato un possibile modello[53] concettuale, estrapolato dalla Specifica di BTP:

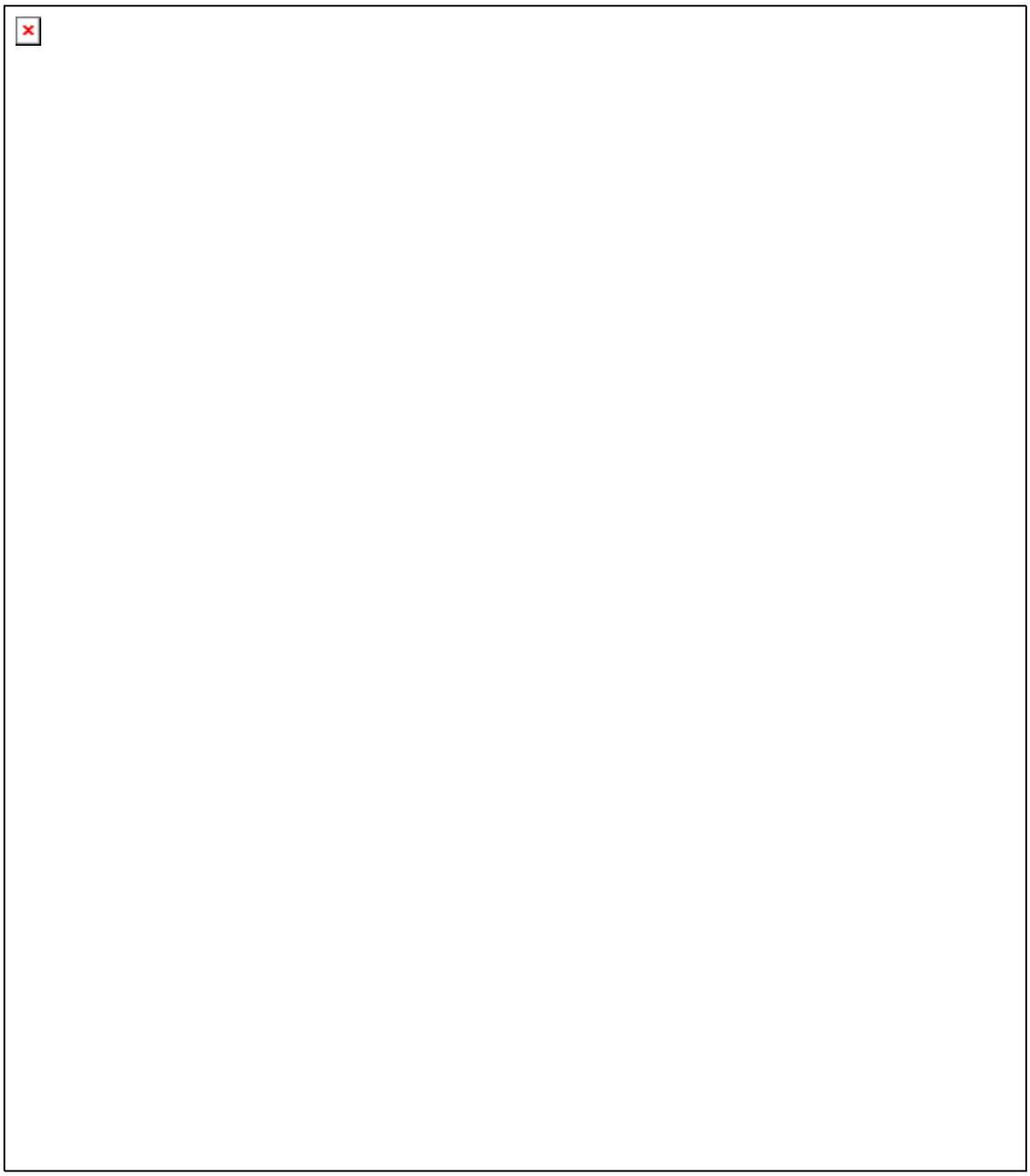

Figura 23 Modello Concettuale di BTP

[53] Rispetto alla nomenclatura utilizzata nella Specifica, nel modello e nella relativa descrizione qui presentati, è stato aggiunto il prefisso "BTP" al nome del Context. E' stata aggiunta, inoltre, l'entità BTPCoordinator , alla quale la Specifica non assegna un nome.

Ogni entità concettuale che partecipa in una transazione BTP è rappresenta da una classe. Le entità stereotipate con <<Agent>> possono essere considerate come degli *agenti software*. Ogni Agente permette ad un attore di assumere un determinato ruolo[54] nella Transazione Business, ed ogni attore, nel ruolo conferitogli da un particolare Agente, ha la caratteristica di essere singolarmente distinguibile, poiché indirizzabile separatamente.

Lo stesso attore può assumere più ruoli nella stessa Transazione Business, o lo stesso ruolo in più Transazioni Business, concorrentemente o sequenzialmente.

In Figura 23 sono evidenziate, con colori diversi, le entità **IniziatingApplicationElement** e **ServiceApplicationElement**, corrispondenti alle parti applicative degli attori rispettivamente nel ruolo[55] di *richiedente* ed *esecutore*. Tali parti svolgeranno le funzioni proprie della logica business, mentre gli Agenti definiti da BTP corrispondono al Middleware di supporto che permetterà la coordinazione all'interno della transazione.

L'entità IniziatingApplicationElement comunica con l'agente coordinatore, detto **BTPCoordinator**, per istruirlo circa la transazione che deve essere coordinata.

BTPCoordinator è legato con l'entità **Participant** dalla relazione Superior-Inferior, che oltre a rappresentare una relazione di controllo del tipo "padre-figlio", rappresenta il fulcro dell'intero modello BTP. Il Participant, che rappresenta l'agente BTP associato all'elemento applicativo dell'*esecutore*, assume il ruolo di Inferior, e come tale si sottopone alla decisione finale del coordinatore. Questo, nel ruolo di Superior, ha il compito di coordinare verso il completamento della transazione tutti gli Inferiors con cui è relazionato, in funzione delle direttive fornite da IniziatingApplicationElement. Come già osservato, ogni *esecutore* può essere associato a più Participants, e può, così, partecipare a più relazioni Superior-Inferior, nella stessa o in diverse transazioni business.

[54] Ogni Agente permette ad un *richiedente* o ad un *esecutore* di assumere tale ruolo all'interno di una transazione business, ma ogni agente, allo stesso tempo, può assumere di per sè svariati ruoli, così come definito da BTP.
[55] Con i ruoli di *richiedente* ed *esecutore* si fa riferimento a quanto definito nel capitolo introduttivo.

Un Agente che nella stessa transazione business assuma sia il ruolo di Inferior che quello di Superior è detto **Interposed** o **Intermediate**.

Con questo nuovo Agente, che esegue due ruoli contemporaneamente, BTP supporta la tecnica dell'Interposizione. L'albero dei partecipanti che così si determina, non è limitato da BTP, e può, in teoria, avere qualsiasi larghezza e profondità, a seconda dalle necessità applicative.

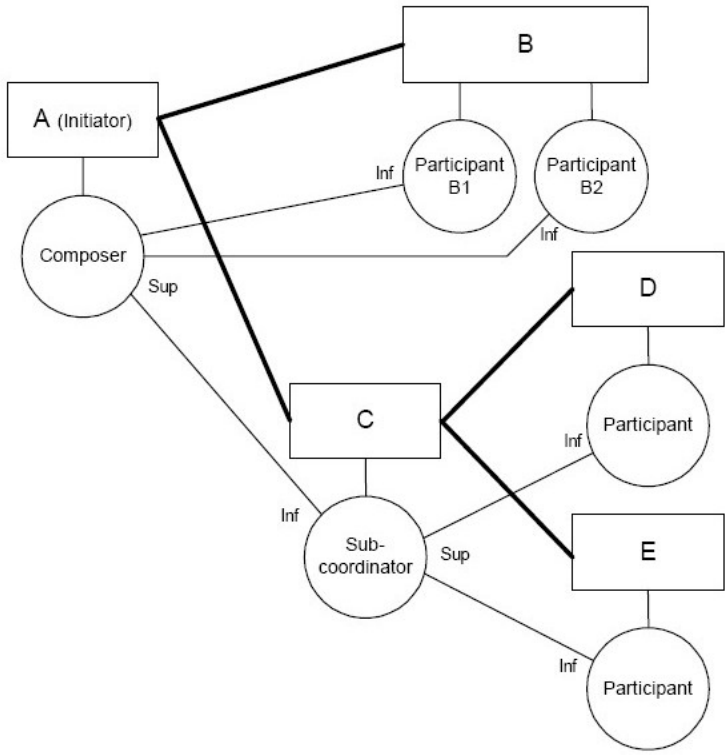

Figura 24 Esempio di Albero di Transazione
-tratta da Riferimenti[3 p34]-

Un Superior, oltre a poter essere un Interposed, viene specializzato a seconda che consenta di trattare i propri Inferior *atomicamente* o *coesivamente*.

Nel caso *atomico*, non consentirà, all'elemento applicativo associato, la selezione degli Inferiors che dovranno confermare e di quelli che dovranno annullare, mentre questo è reso possibile nel caso *coesivo*. Inoltre, in quest'ultimo caso, l'elemento applicativo associato al Superior può reagire dinamicamente ad un Inferior che comunichi di non essere Prepared, ad esempio *cancellandolo* e ingaggiando altri Inferiors, effettuando del lavoro applicativo, o più semplicemente richiedendo l'annullamento per tutti i

partecipanti già preparati nella transazione. In altri termini Cohesion supporta, a pieno titolo, il requisito della Composizione Dinamica.

Un Superior che tratti i suoi Inferiors atomicamente è detto **Coordinator**, altrimenti è detto **Composer**. Se poi tale Superior è pure un Interposed, si parla rispettivamente di **SubCoordinator** e **SubComposer**.

Una Transazione Business è detta **Atom,** se il Superior principale è un Coordinator, altrimenti, se è un Composer, è detta **Cohesion**.

In una Cohesion, l'insieme dei partecipanti *preparati* nella prima fase, e che dovrà confermare nella seconda, è detto **Confirm-set**. In una Atom, il Confirm-set è costituito da tutti gli Inferiors *arruolati*.

Nel caso del SubCoordinator, prima che questo possa divenire Prepared nei confronti del diretto superiore, lo dovranno essere tutti i suoi inferiori, mentre nel caso del SubComposer, sarà l'elemento applicativo associato a decidere, in funzione dei propri inferiori Prepared, se questi siano sufficienti a dichiarare che il SubComposer sia *pronto*.

L'Agente che opera da Superior principale nell'albero della transazione, ha un ruolo particolare nei confronti dell'elemento applicativo cui è associato. Infatti, quando sia istruito dall'applicazione a confermare la transazione, determina la decisione finale in funzione della *preparazione* degli inferiori nel Confirm-set. Pertanto, il suo ruolo è detto **Decider**. L'elemento applicativo che gli chiede di confermare è detto **Terminator**, poiché, in ogni caso, conduce verso la chiusura della transazione.

Nella nomenclatura BTP, per Partecipante si intende strettamente un Inferior che non abbia a sua volta inferiori. Diversamente, esso viene identificato più propriamente come SubComposer o SubCoordinator. Un Superior, in ogni caso, non è al corrente del ruolo specifico di un suo diretto inferiore: per esso è sempre e solo un partecipante.

Una Transazione Business è rappresentata da un **BTPContext**, che ne definisce i "connotati". Esso contiene i seguenti campi[56]:

- superiorAddress: l'indirizzo del Superiore al quale dovranno fare riferimento gli inferiori. Può essere un set di indirizzi alternativi[57].
- superiorID: identificatore, globalmente univoco, del Superior.
- superiorType: può indicare Cohesion o Atom.
- qualifiers: può contenere eventuali Qualificatori[58] opzionali. Contiene sempre il Qualificatore *transactionTimelimit*, che descriveremo in seguito.

L'esecuzione di una transazione, all'interno di un contesto ben definito, avviene proprio condividendo il BTPContext. Questo deve essere "contenuto" nei messaggi applicativi, per indicare al ricevente il contesto nel quale detti messaggi debbano essere interpretati. In ogni caso, tale relazione di contenimento non è definita da BTP, e così sono possibili più scelte implementative[59].

[56] Si noti, tra i campi contenuti nel BTPContext, l'assenza di un identificatore della transazione. La transazione si identificherà dunque con l'identificatore univoco del Superior. Lo svantaggio di questo approccio consiste nell'impossibilità di verificare facilmente l'esistenza di *deadlock* di tipo *permanente*.
[57] Vedi "Meccanismo di Redirection" in "2.1.3 Recovery and Failure Handling".
[58] Il concetto di Qualificatore sarà dato in seguito.
[59] Alcune di tali scelte saranno accennate nel paragrafo "2.1.5 Binding".

2.1.2 Protocollo

Nel modello concettuale mostrato nel precedente paragrafo compaiono svariate relazioni. Quelle in cui è stato definito e mostrato un ruolo, sono specificate da BTP sia nella sintassi che nella semantica. Per la relazione ServiceApplicationElement-Participant, invece, BTP specifica solo il significato.

La figura seguente mostra un modello di secondo livello in cui, ad ogni ruolo, è associata un'interfaccia atta a raggruppare tutti i messaggi che detto ruolo deve essere in grado di interpretare.

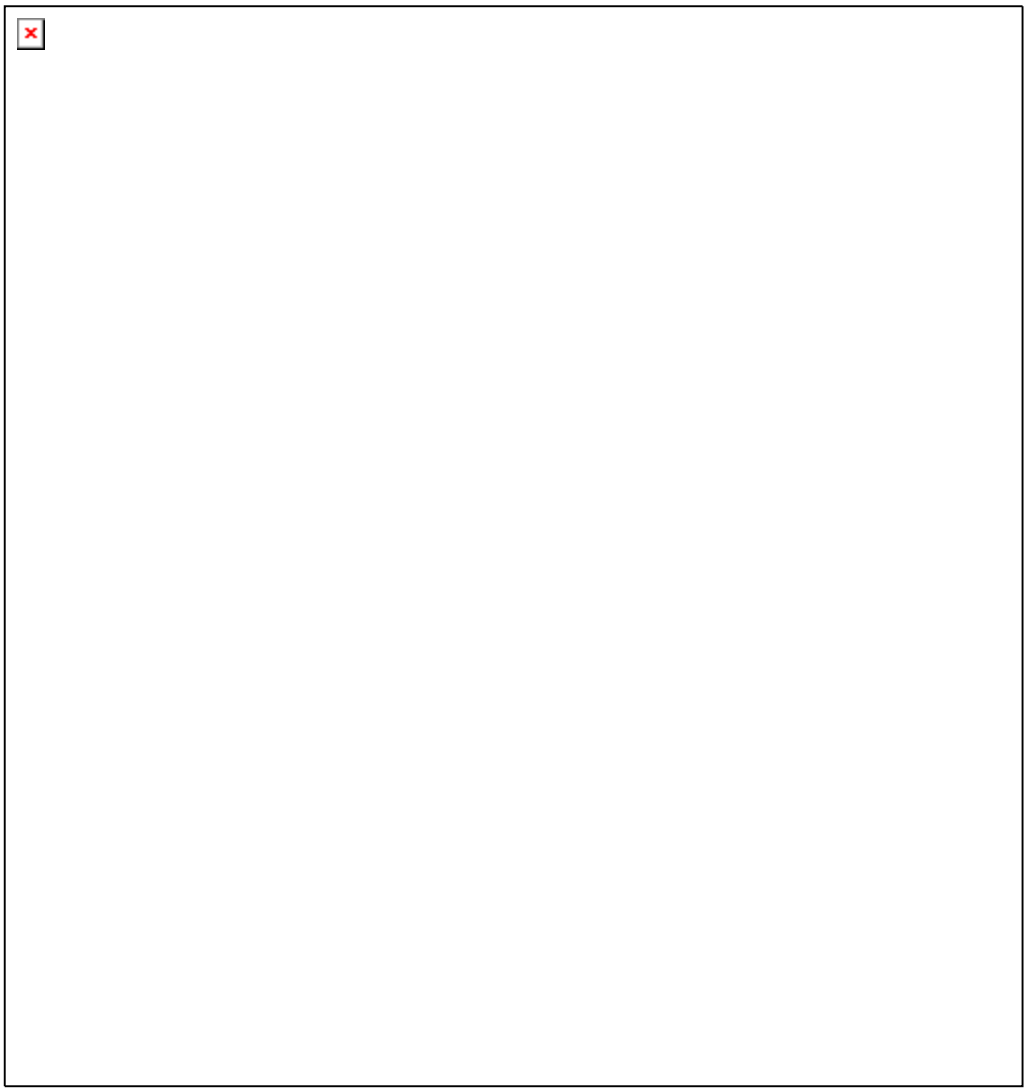

Figura 25 Modello delle Interfacce di BTP

Si noti che le interfacce corrispondenti a ruoli relazionati nel modello concettuale, continuano ad essere relazionate anche in questo modello. Tuttavia, in questo, sono presenti pure alcune novità. In particolare sono presenti due interfacce corrispondenti a ruoli BTP in precedenza volutamente trascurati:

- **Enroller**: rappresenta il ruolo di un'entità che richiede l'arruolamento di un Participant con un Superior. In Figura 25 tale ruolo è stato attribuito all'entità ApplicationElement, nel senso che potrebbe essere intrapreso da qualsiasi Agente software.
- **StatusRequestor**: rappresenta il ruolo di un'entità che richiede informazioni di stato ad un Participant o ad un Superior. Può essere intrapreso da qualsiasi Agente software.

E' presente anche una nuova entità:

- **Factory**: è stata introdotta in BTP semplicemente per poter discriminare le richieste di coordinamento effettuate da un Initiator, e poter conseguentemente avviare il giusto tipo di coordinatore, Composer o Coordinator, a seconda di quanto indicato nella richiesta dell'Initiator.

Le interfacce mostrate rappresentano, seppure prive di dettaglio, la sintassi del protocollo BTP. Nel seguito, per descriverne anche la semantica, percorreremo il **ciclo di vita di una Transazione Business**, cercando di seguire la stessa logica descrittiva che hanno utilizzato gli autori della Specifica in esame. Utilizzeremo alcuni termini che, pur avendo un significato intuitivo, è conveniente precisare in anticipo per consentire una migliore comprensione:

- **Identificatore**: identifica un particolare attore BTP in uno specifico ruolo; deve essere univoco in tutta la rete (va inteso dunque come URI).
- **Indirizzo**: rappresenta l'*endpoint* a cui risponde un Attore, indipendentemente dal ruolo assunto. Dipende dal Binding con il protocollo di Trasporto.
- **Qualificatori**: sono campi contenuti nei messaggi BTP con cui gli attori possono scambiarsi informazioni addizionali. Possono essere specificati da BTP,

diventando Qualificatori standard[60] (ad esempio i *timeout* di cui discuteremo in seguito), oppure da gruppi industriali e/o implementatori, per particolari propositi applicativi.

Per la comprensione di quanto segue, il lettore potrà aiutarsi con i Sequence Diagrams riportati in Figura 26 e Figura 27, che mostrano le interazioni principali rispettivamente nel caso di una transazione *atomica* e nel caso di una transazione *coesiva*, entrambe aventi, per semplicità, un solo *richiedente* ed un solo *esecutore*.

Figura 26 Interazioni principali in una Transazione Business di tipo Atom

[60] I Qualificatori standard definiti da BTP saranno descritti approfonditamente nel paragrafo "2.1.4 Qualificatori Standard".

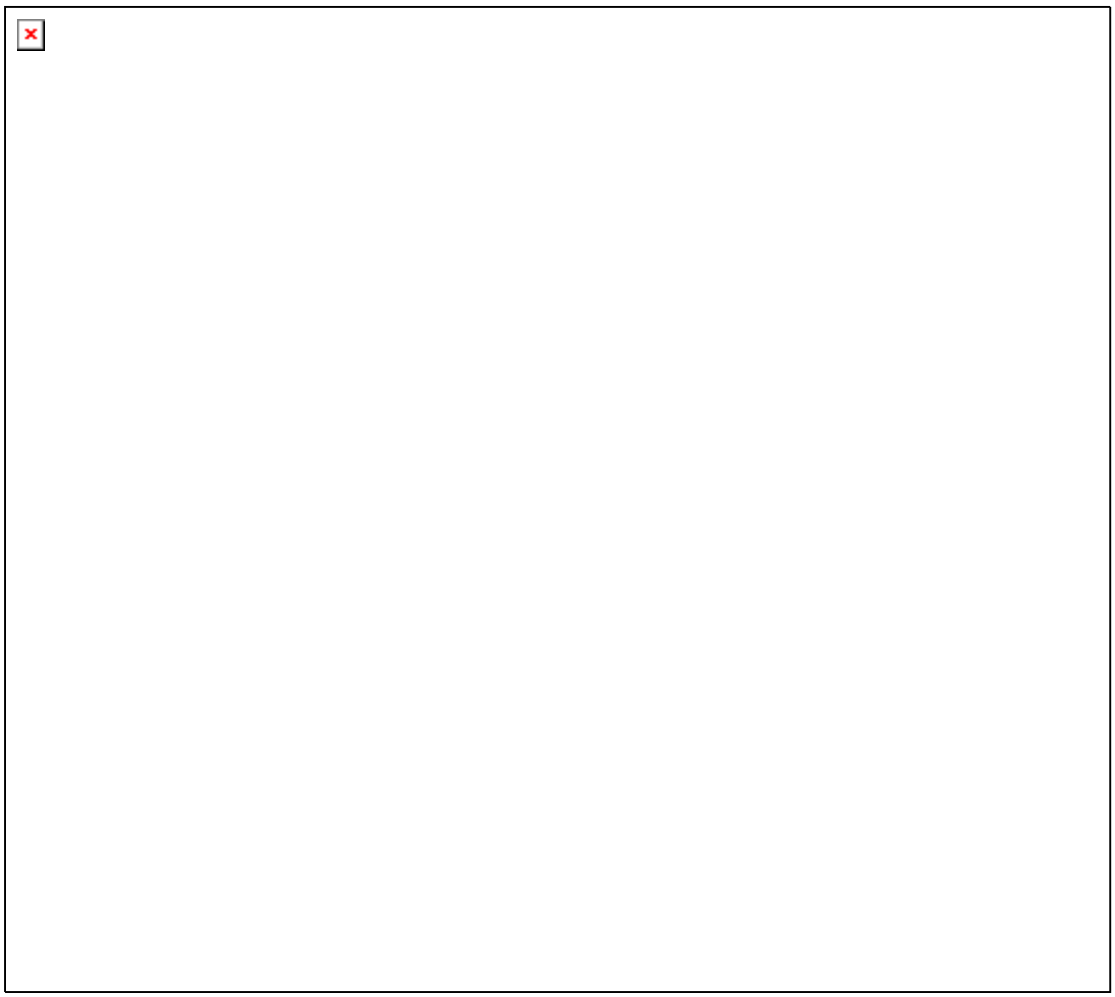

Figura 27 Interazioni principali in una Transazione Business di tipo Cohesion

Creazione

Al momento della "creazione" di una Transazione Business, l'elemento applicativo creante funge da Initiator. Esso comunica al Factory BTP un messaggio di *begin*, in cui specifica se richiede un Coordinator o un Composer. In risposta ottiene un *begun*, unito al BTPContext che rappresenta la nuova Transazione. A questo punto è stato creato anche il richiesto Agente software, Composer o Coordinator, a seconda di quanto richiesto. Questo potrà avere un identificatore distinto e risiedere su un attore già in uso, ad esempio sull'attore che attua da Initiator, o potrà avere un indirizzo del tutto indipendente e risiedere su un nuovo attore. In ogni caso, indirizzo e identificatore, quest'ultimo relativo al ruolo di Decider, saranno restituiti nel messaggio di *begun*, e potranno successivamente essere utilizzati dal Terminator. Il Terminator, normalmente,

è un ruolo attribuito allo stesso attore che funge da Initiator, ma scelte diverse sono altrettanto lecite.

Propagazione

Per stabilire relazioni di tipo Superior-Inferior all'interno di una stessa Transazione Business, insieme ai messaggi applicativi deve essere "propagato" il BTPContext, affinché l'elemento applicativo ricevente capisca che gli si richiede un lavoro, quello indicato nel messaggio applicativo associato, soggetto ad una decisione di conferma o di annullamento.

Il ricevente, ottenuto il BTPContext, lo passerà ad un nuovo Agente Participant il quale, utilizzando l'indirizzo del Superior in esso contenuto, potrà arruolarsi come Inferior nella Transazione, inviando un messaggio di *enrol*[61].

A questo punto il ricevente può inviare un messaggio applicativo di risposta all'elemento applicativo del richiedente, assieme ad un *BTPContext_Reply*.

Il *BTPContext_Reply* ha una funzione ben precisa. In generale, infatti, in seguito ad una richiesta applicativa, il servizio esecutore potrebbe fornire una risposta anche quando l'arruolamento degli inferiori da esso controllati non sia ancora stato concluso. In questa eventualità, se il richiedente effettuasse il Confirm, ricordando che l'arruolamento è di tipo "Pull" (avvenendo su iniziativa dell'arruolante), il lavoro applicativo che sarebbe dovuto essere parte di una transazione atomica risulterebbe parziale, violando così l'atomicità in una maniera trasparente all'applicazione stessa.

Per evitare questa eventualità, è stato introdotto il *BTPContext_Reply*, con cui l'Inferior può comunicare al suo diretto Superior che non effettuerà ulteriori "sotto-arruolamenti". In questo modo la risposta applicativa può essere fornita subito dopo la richiesta, senza dover attendere la conclusione degli arruolamenti, come dovrebbe avvenire, invece, in mancanza del messaggio in questione. Il suo uso, almeno per una Atom, è obbligatorio. Nel caso di una Cohesion, sebbene sia compito dell'applicazione che detiene il controllo degli inferiori stabilire quando e se sia il momento per poter richiedere il Confirm, il *BTPContext_Reply* può ancora essere utilizzato.

[61] Il modello per l'arruolamento è pertanto di tipo "Pull".

Con l'aggiunta di alcune informazioni, può essere utilizzato anche con un ulteriore significato: può indicare che l'Inferior mittente non è stato capace di arruolarsi con il Superior (pur essendo capace di rispondere al mittente del BTPContext).

Creazione di Intermediate

Affinché un Inferior possa essere anche un Superior, ovvero possa essere un SubComposer o un SubCoordinator, BTP richiede, insieme al messaggio di *begin* comunicato al Factory BTP, che sia inviato anche il BTPContext della "transazione padre", al quale il nuovo Contesto sarà relazionato[62].

Evidentemente, un elemento applicativo che voglia avere un maggiore controllo sui propri inferiori, può utilizzare delle strategie particolari. Ad esempio, può creare una Cohesion, e registrare se stesso sia come SubComposer che come SubCoordinator di questa. In tal modo potrà controllare direttamente alcuni lavori transazionali come atomici, e altri come coesivi.

Sequenza dei messaggi

La sequenza dei messaggi validi è scandita dagli stati in cui si trovano gli attori BTP nelle relazioni Superior-Inferior, come mostrato dagli State Diagrams in Figura 28 e Figura 29, ripresi dalla Specifica, per una singola relazione Superior-Inferior. Da uno stato BTP si può uscire mediante un comando applicativo, oppure per la ricezione di un messaggio da parte dell'interlocutore in detta relazione.

BTP consente che ogni particolare Implementazione possa decidere di aggregare alcuni di tali stati, e che possa scegliere di rispedire messaggi nel caso in cui sia ragionevole credere che non siano stati ricevuti (ad esempio, per via di una segnalazione di errore dal protocollo di Trasporto sottostante, per un timeout, o per il verificarsi di un evento di recovery).

Seppure non rappresentato negli State Diagrams, in qualsiasi stato un Inferior può inviare un messaggio di *hazard*. Questo rappresenta l'individuazione e la

[62] La relazione tra Contesto "figlio" e Contesto "padre" si traduce semplicemente nel fatto che il Factory avvierà un nuovo Superior che arruolerà automaticamente come inferiore presso il Superior indicato nel Contesto "padre" .

memorizzazione di un problema, e conseguentemente indica al Superior che l'Inferior mittente non può effettuare le operazioni di cui è responsabile in maniera consistente.

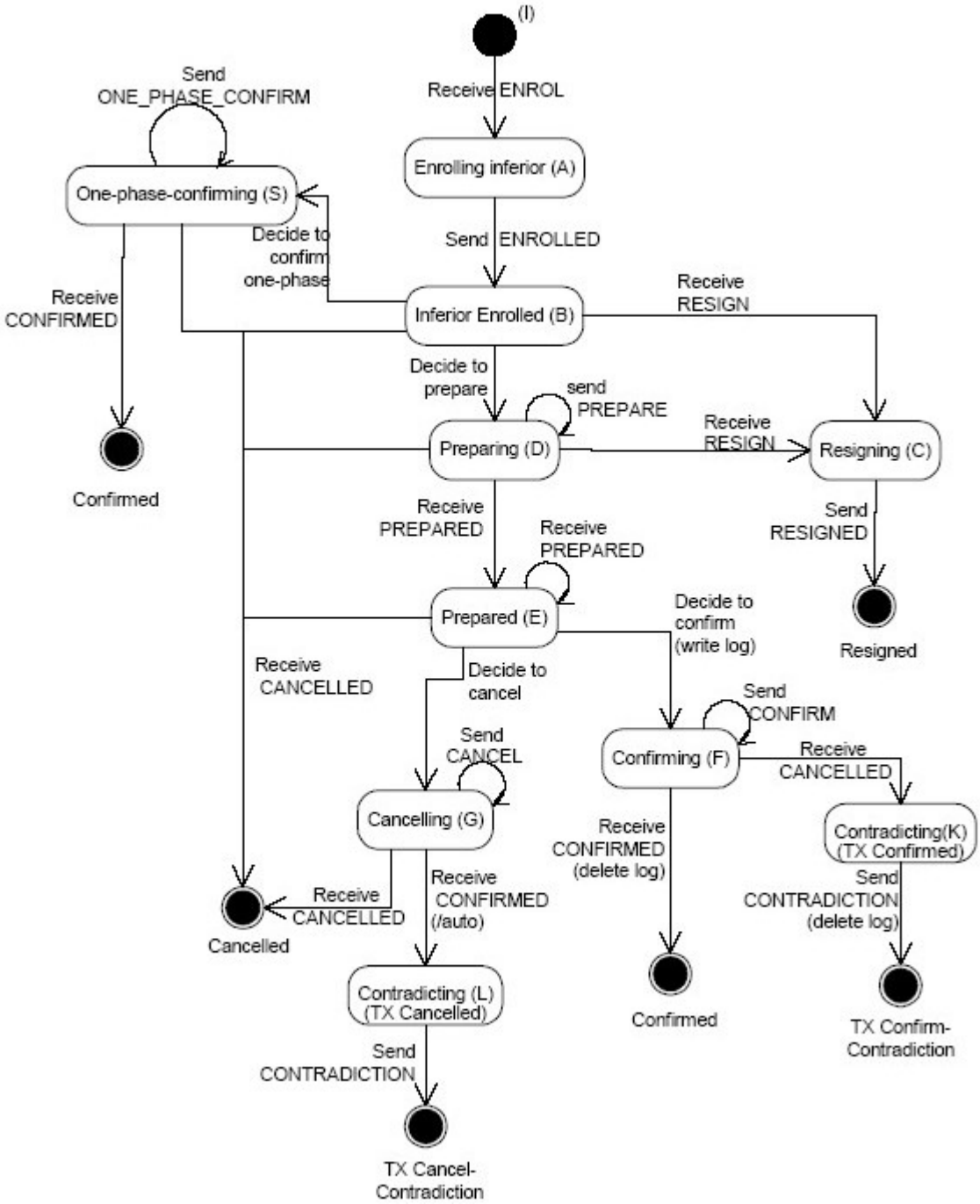

Figura 28 State Diagram di un Superior in una relazione Superior-Inferior
-tratta da Riferimenti[3 p32]-

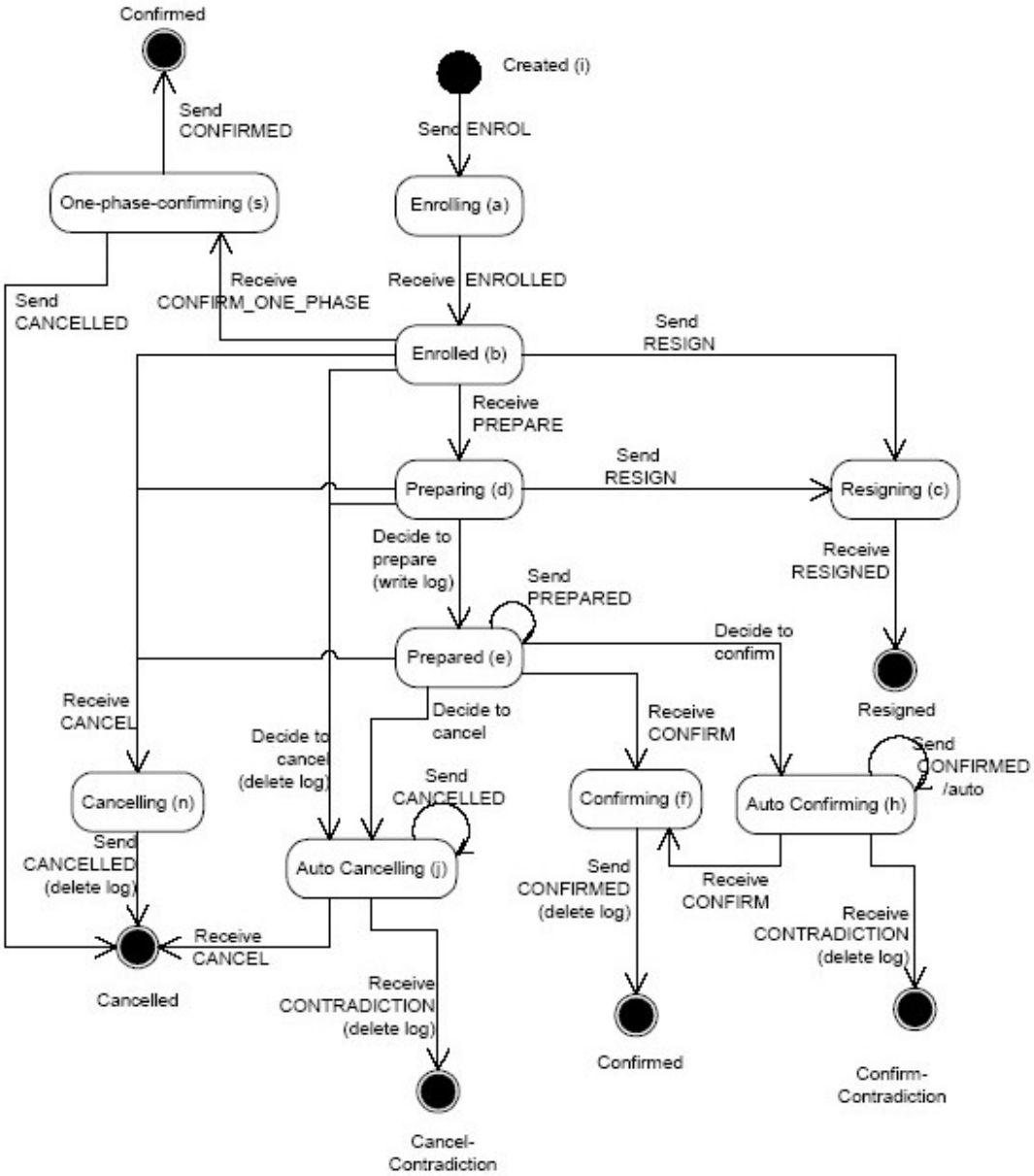

Figura 29 State Diagram di un Inferior in una relazione Superior-Inferior
-tratta da Riferimenti[3 p33]-

Controllo degli inferiori

Ogni Superior controlla, e ha visibilità, solo dei suoi diretti inferiori. Contrariamente ad una transazione classica, in cui il lavoro viene eseguito da un singolo *esecutore* (il DBMS), in una Transazione Business il lavoro viene eseguito da più agenti software. In una Cohesion è dunque necessario identificare ogni singolo Agente Inferior per poterlo confermare selettivamente. Tale identificativo è contenuto nel messaggio BTP di *enrol*,

e pertanto è noto al coordinatore (il Superior), ma deve essere conosciuto anche dall'elemento applicativo a questo associato, poiché è quest'ultimo ad effettuare la conferma selettiva.

La Specifica di BTP non definisce come l'elemento applicativo corrispondente al Superior possa venire a conoscenza dei singoli identificativi, ma suggerisce che una possibile scelta implementativa consiste nell'inserire l'identificativo nel messaggio applicativo di risposta inviato dal ServiceApplicationElement associato all'inferiore. Tali considerazioni sono valide anche per una Atom, seppure perdano di rilevanza, non essendoci possibilità di selezione.

Composer, Coordinator, ed evoluzione del Confirm-set

La Composizione Dinamica è supportata da BTP mediante le transazioni di tipo Cohesion. In queste è infatti possibile selezionare individualmente gli inferiori che debbano annullare e quelli che debbano confermare. In una transazione di tipo Atom, viceversa, non è possibile nessuna selezione. Per una transazione atomica, dunque, è lecito attendersi un comportamento del Coordinator molto simile a quello del classico Transaction Manager[63] descritto in relazione al protocollo 2PC.

In Figura 30 viene riportato lo State Diagram[64], ripreso dalla Specifica, per il comportamento di un BTPCoordinator, valido sia nel caso di Composer che di Coordinator.

[63] Vedi paragrafo "1.2.1 Two Phase Commit e transazioni distribuite".

[64] Il diagramma, a differenza di quello precedentemente riportato, che rappresentava solo il comportamento da Superior in una singola relazione Superior-Inferior, rappresenta il comportamento complessivo di un BTPCoordinator: considera tutti gli Inferiors, e la relazione con lo/il Initiator/Terminator. Tali diagrammi, modellando il comportamento dello stesso ruolo sono, ovviamente, tra loro relazionati.

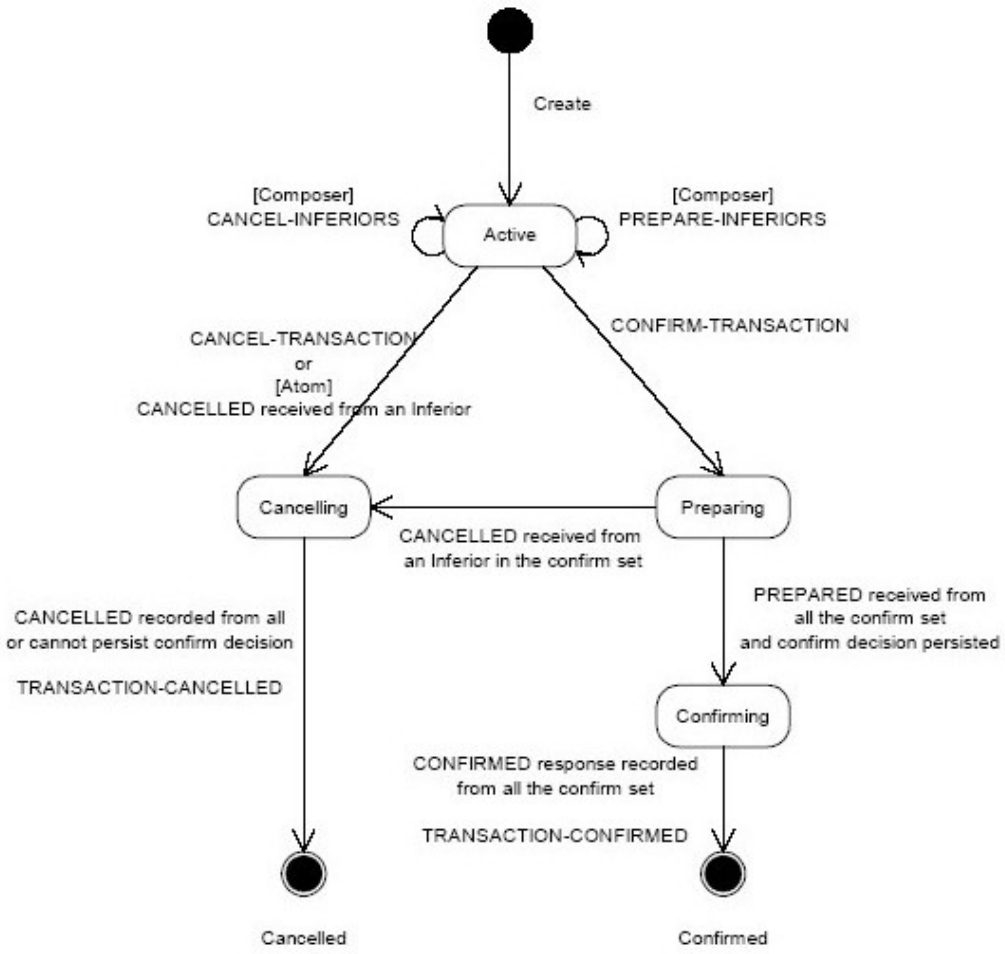

Figura 30 State Diagram di un BTPCoordinator (Composer o Coordinator)
-tratta da Riferimenti[3 p37]-

Come anticipato, per un Coordinator lo State Diagram mostra il comportamento atteso, e non necessita di particolari spiegazioni. Le uniche novità, peraltro non tutte mostrate nel diagramma, sono elencate di seguito:

- Un inferiore può inviare un *resign* al suo superiore, non nel senso che si ritira dalla Transazione, bensì nel senso che non è interessato alla decisione finale, poiché, ad esempio, i suoi dati vengono utilizzati in sola lettura, e dunque sono indipendenti da una eventuale annullamento o conferma. Il messaggio di *resign* può essere inviato in qualsiasi momento, purchè chi lo invii non abbia precedentemente inviato un messaggio di *prepared* o di *cancelled*.

- In ogni momento prima del *confirm*/*cancelTransaction*, il Terminator può richiedere lo stato corrente dei "preparandi", inviando una richiesta di *requestInferiorStatutes* al BTPCoordinator.
- La Transazione arriva al completamento con un messaggio di *transactionConfirmed* o *transactionCancelled*, dal Coordinator al Terminator, o con un messaggio di *inferiorStatutes*, in caso di problemi, e se il Terminator aveva precedentemente indicato di essere interessato ai dettagli del risultato.
- Un partecipante può inviare, in qualsiasi momento, un messaggio di *cancelled* per indicare che non può effettuare il lavoro richiesto. Dopo l'invio di tale messaggio, tuttavia, non potrà ritenersi completamente svincolato dalla Transazione, poiché dovrà attendere una risposta dal suo diretto Superior.

I quattro punti detti risultano validi anche per un Composer, ma, per comprendere il comportamento di quest'ultimo, sono necessarie alcune informazioni aggiuntive:

- Un messaggio di *prepareInferiors*, parametrizzato con gli identificatori degli inferiori da preparare, può essere inviato dal Terminator al Composer per richiedere la preparazione dei partecipanti (nel caso di transazione atomica, tale messaggio non è lecito). Quando il Composer avrà ricevuto tutti i messaggi di riscontro dall'insieme indicato, invierà un messaggio di *inferiorStatutes* al Terminator, con il quale quest'ultimo potrà verificare l'eventualità che alcuni inferiori abbiano inviato un *cancelled* piuttosto che un *prepared*.

  A questo punto il Terminator potrà inviare un messaggio di *confirmTransaction*, questa volta parametrizzando il Confirm-set (eventualmente in maniera differente dall'insieme definito in *prepareInferiors*). Alternativamente potrà richiedere un *cancelTransaction*, ovviamente non parametrizzato.
- Dopo l'invio di un *confirmTransaction*, con allegato il Confirm-set parametrizzato, se un inferiore di tale insieme abbia già inviato un *cancelled*, la transazione viene annullata dal Composer in maniera analoga a quanto farebbe un Coordinator.
- Il Composer, dopo la "creazione" della transazione, si pone nello stato di Active, nel quale può ricevere un numero qualsiasi di messaggi di *prepareInferiors* o di *cancelInferiors* dal Terminator. Da tale stato può uscire solo con una richiesta di *confirm*/*cancelTransaction*. Per ogni richiesta di *prepareInferiors*, il Composer

invierà il messaggio di *prepare* solo agli inferiori dai quali non ha ancora ricevuto un *resign*, un *cancelled* o un *prepared*. Pertanto, un servizio "cancellato" non può essere recuperato in seguito, e l'annullamento può essere considerato equivalente al "dearruolamento" dalla transazione. Analogamente, per ogni richiesta di *cancelInferiors*, il Composer invierà un messaggio di *cancel* a tutti gli inferiori dai quali non ha ancora ricevuto un *cancelled* o un *resign*.

Riassumendo, per una Cohesion, il concetto chiave è rappresentato dal fatto che il Confirm-set può variare all'interno degli inferiori "enrolled", tra quelli che non hanno ancora "cancellato" (e non sono stati ancora "cancellati"), e non hanno effettuato il *resign* (indicante che l'inferiore è invariante rispetto alla decisione globale). Nel caso in cui l'Initiator decida di inserire nel Confirm-set un inferiore che abbia già annullato, può farlo, ma, evidentemente, se in tal caso inviasse un *confirmTransaction*, l'esito della transazione sarebbe scontato.

Per una transazione atomica, invece, il Confirm-set è statico, ed è definito da tutti gli inferiori *enrolled*.

Intermediate ed evoluzione del Confirm-set

La logica è praticamente quella di un Coordinator o di un Composer, questa volta subordinato non solo rispetto all'elemento applicativo associato, ma anche rispetto al diretto Superior. Come mostrato dallo State Diagram in Figura 31, che rappresenta il comportamento di un Intermediate, l'unica differenza sostanziale risiede nella presenza di un nuovo stato, quello di Prepared, nel quale l'Intermediate transita per comunicare al suo Superior quando sia pronto per la seconda fase del protocollo.

Nel caso di una transazione di tipo Atom, affinché il SubCoordinator possa dichiararsi preparato, devono esserlo anche tutti i suoi diretti Inferiors.

Nel caso di una Cohesion, invece, l'elemento applicativo associato al SubComposer è chiamato a valutare il Confirm-set locale, ovvero l'insieme degli Inferiors che devono essere preparati per considerare preparato anche il SubComposer nei riguardi del suo diretto Superior. Il Confirm-set, dunque, evolve ancora come nel caso del Composer, ma, ora, al posto del Terminator, è l'elemento applicativo associato al SubComposer che, tenendo conto delle richieste del Superior, decide come "pilotare" i suoi Inferiors.

Per un SubComposer, in realtà, seppure non deducibile dallo State Diagram mostrato, l'elemento applicativo associato non deve limitarsi solo alla gestione dei propri Inferiors. Infatti, in funzione delle necessità applicative e delle decisioni del Superior, può effettuare qualsiasi lavoro, e/o intraprendere altre "azioni BTP". Ad esempio, anche quando abbia ricevuto un messaggio di *confirm* dal Superior, l'elemento applicativo dell'Intermediate può ancora arruolare inferiori, o può cambiare il Confirm-set. In altri termini, almeno per una Cohesion, BTP non pone alcuna limitazione all'implementazione della logica applicativa.

Nel caso di una Atom, invece, la logica applicativa associata al SubCoordinator è vincolata dalla proprietà di Atomicità della transazione. In tal caso, lo State Diagram di Figura 31 risulta completamente auto esplicativo.

Si ricordi che il *BTPContext_Reply* viene scambiato tra gli elementi applicativi per garantire al Terminator/Initiator che non ci saranno più ulteriori sotto-arruolamenti. Quanto detto vale ricorsivamente attraverso l'albero della transazione, e dunque vale anche nei confronti degli elementi applicativi associati ad un SubComposer o SubCoordinator.

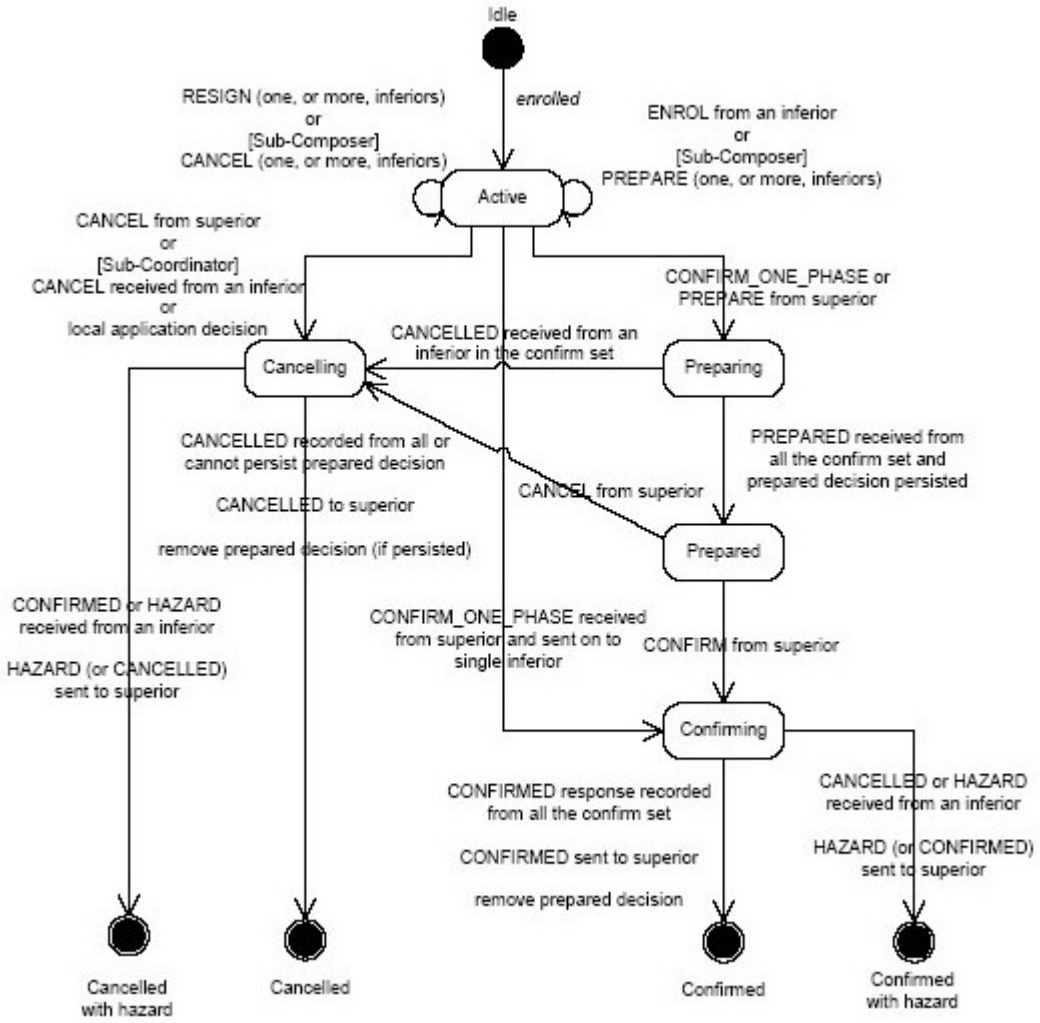

Figura 31 State Diagram di un Intermediate
-tratta da Riferimenti[3 p41]-

2.1.2.1 Ottimizzazioni e varianti

Nella precedente descrizione del protocollo BT, abbiamo già incontrato alcune varianti rispetto al 2PC classico. In questo paragrafo le descriveremo singolarmente, insieme ad altre varianti e ad alcune utili ottimizzazioni previste da BTP.

Preparazione spontanea

BTP permette che un inferiore possa risultare preparato, inviando il corrispondente messaggio al superiore, prima ancora che riceva un messaggio di *prepare* dal suo diretto Superior. Dal punto di vista del Superior, tuttavia, nulla deve cambiare rispetto alla situazione ordinaria.

One-shot

La sequenza ordinaria dei messaggi scambiati in una relazione Superior-Inferior si svolge con otto messaggi in quattro fasi:

1. Sequenza applicativa di request/response;
2. Sequenza ENROL/ENROLLED (in verso opposto, dall'Inferior al Superior);
3. Sequenza PREPARE/PREPARED;
4. Sequenza CONFIRM/CONFIRMED;

L'ottimizzazione in questione consiste nel riunire le prime tre fasi, ottenendone così, in totale, soltanto due. Questo è realizzabile grazie alla possibilità di "preparazione spontanea" prima descritta, associata ad un arruolamento di tipo "push". In pratica, in risposta al messaggio applicativo del *richiedente*, il partecipante comunica insieme al "response" anche i messaggi di *prepared* e di *enrol*. Sarà poi il *richiedente* ad arruolare effettivamente l'inferiore, associato al partecipante in questione, presso il coordinatore.

Si noti che, con la sequenza appena descritta, il partecipante non ottiene alcun riscontro sull'arruolamento, e questo potrebbe essere non desiderabile per il partecipante stesso. BTP risolve il problema prevedendo il Qualificatore "inferiorTimeout", con cui un inferiore può indicare per quanto tempo, al massimo,

cercherà di rimanere nello stato di Prepared. In questo modo l'Inferior non rischierà di rimanere preparato all'infinito, anche nell'eventualità in cui l'arruolamento non sia riuscito.

Analogamente, mediante il Qualificatore "TransactionTimelimit", BTP permette ad un superiore di specificare, nel *prepare*, per quanto tempo esso si aspetta che l'inferiore mantenga lo stato di "preparato". Trascorso tale tempo senza ricevere ulteriori comunicazioni, l'inferiore può supporre circa l'esistenza di un qualche problema, e comportarsi di conseguenza.

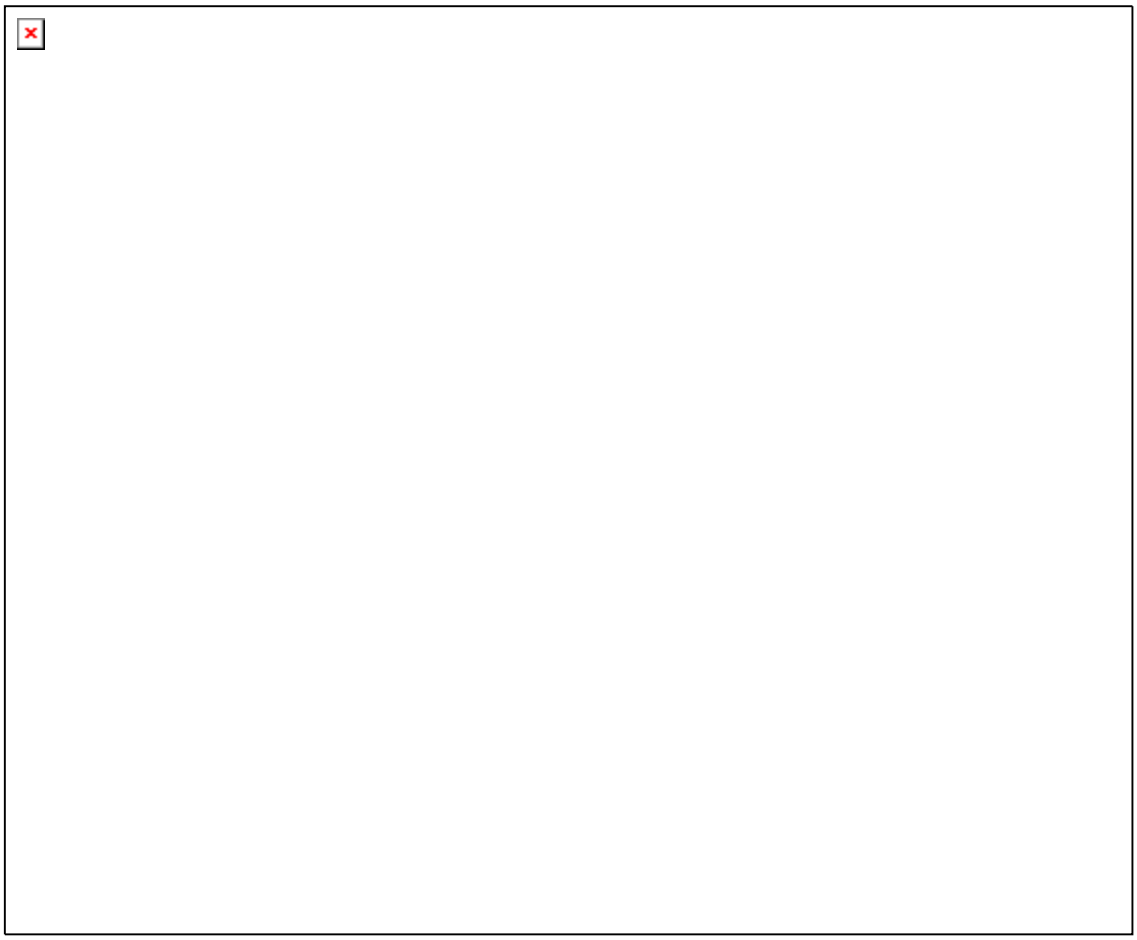

Figura 32 Ottimizzazione One-shot per una transazione di tipo Atom

Resignation

Abbiamo già osservato che esiste la possibilità, per un Inferior che sia *invariante* rispetto alla decisione finale, ovvero tale che i suoi dati siano utilizzati in sola lettura, di poter richiedere il *resign* al suo diretto Superior. In seguito alla richiesta, esso non

parteciperà alla seconda fase del protocollo di coordinamento, consentendo una minimizzazione dei messaggi scambiati.

Conferma in una fase

Nel caso in cui il Confirm-set sia composto da un solo Inferior, il Superior di questo, piuttosto che avviare il 2PC con un messaggio di *prepare*, può richiedere direttamente la conferma, con un messaggio di *confirmOnePhase*, delegando la "decisione finale" all'Inferior. In questo modo si evita l'inutile esecuzione della prima fase del protocollo.

Cancellazioni e conferme autonome, contraddizioni

Abbiamo più volte sottolineato come BTP non ponga vincoli per quanto riguarda l'implementazione degli Effects. Ogni applicazione può concretizzarli come meglio crede.

Abbiamo anche visto che sono permesse le *preparazioni spontanee*.

In aggiunta, per consentire una maggiore autonomia agli inferiori, BTP permette ad un Inferior di prendere autonomamente anche la propria decisione finale, prima ancora che gli pervenga una richiesta di *prepare*. Come per la preparazione anticipata, anche per una decisione finale anticipata BTP prevede che l'inferiore possa specificare per quanto tempo "promette" di rimanere nel corrispondente stato[65].

Una decisione autonoma di questo tipo può essere presa per ragioni impreviste, ad esempio in occorrenza di fallimenti, ma anche in maniera pianificata.

Dal punto di vista del Superior, non si pone alcun problema se la sua decisione finale è concorde con quella autonoma. In caso contrario, il Superior comunica all'inferiore un messaggio di *contradiction*, con cui lo informa della discrepanza. Ricevendo tale messaggio l'inferiore diviene conscio che il suo superiore è al corrente della situazione anomala. BTP richiede che un inferiore che prenda una decisione autonoma si assicuri sempre che il suo Superior ne sia a conoscenza, aspettando un messaggio di *contradiction* o un messaggio di *confirm*/*cancel*.

[65] Ciò può avvenire mediante il Qualificatore InferiorTimeout, discusso in "2.1.4 Qualificatori Standard".

Come vedremo, in caso di contraddizioni, BTP mette a disposizione degli appropriati messaggi affinché esse possano essere riportate “indietro” fino alla radice dell’albero della transazione, coincidente con il Terminator.

La Specifica tiene a sottolineare che, a differenza degli altri standards, in cui, solitamente, le decisioni autonome sono contemplate come rari casi eccezionali derivanti da euristiche (ovvero non pianificati), in BTP si assume che le decisioni autonome possano essere prese anche in maniera pianificata.

In effetti, la concessione di decisioni autonome può comportare, in presenza di *failures*, una serie di problemi aggiuntivi a quelli già noti del classico 2PC, e di una certa complessità.

Ad esempio, cosa accadrebbe se un Inferior, a seguito di una decisione autonoma di Confirm, non potesse attendere il riscontro del Superior? La contraddizione non sarebbe individuata se il messaggio contenente la decisione autonoma si perdesse.

Al contrario, se la decisione autonoma dovesse essere di Cancel, il problema non si presenterebbe. Infatti, se il corrispondente messaggio dovesse perdersi, il coordinatore avvierebbe il protocollo a due fasi, e non ottenendo il messaggio di *prepared*, deciderebbe per un abort globale (almeno nel caso di una Atom).

Nel prossimo paragrafo vedremo i meccanismi di recovery definiti da BTP, e vedremo come questi siano in grado di assicurare, anche in caso di *failures*, l’individuazione di risultati inconsistenti derivanti da decisioni autonome.

2.1.3 Recovery and Failure Handling

BTP classifica i “fallimenti” in due gruppi:

- **Fallimenti di comunicazione**: i messaggi tra gli attori BTP vengono persi e non sono consegnati. In proposito BTP assume che il protocollo del Trasporto sottostante li consegni correttamente o non li consegni affatto. Tuttavia non richiede che questi segnali eventuali perdite, né richiede che i messaggi siano consegnati nell’ordine di invio.
- **Fallimenti di nodo (di sistema o di sito)**: una macchina che ospita uno o più attori BTP si ferma, e tutte le informazioni di stato non persistenti vengono perse. In proposito BTP assume che qualsiasi attore operi correttamente o non operi affatto, ovvero si fermi, e che non operi mai in maniera scorretta.

Un *fallimento di comunicazione* può essere riconosciuto perché segnalato, in una particolare Implementazione, dal sottostante protocollo del Trasporto, o più in generale grazie allo scadere di *timeout* opportunamente settati. Per recuperare questi fallimenti, BTP richiede semplicemente che i due attori possano continuare a comunicare.

Un *fallimento di un nodo* si riconosce invece dal fatto che viene perso lo *stato volatile*. Per garantire la consistenza in presenza di questi fallimenti, BTP richiede che alcune informazioni di stato siano rese persistenti. Dette informazioni riguardano lo stato di avanzamento del protocollo BTP, ma anche lo stato applicativo associato. Pertanto, BTP sottolinea che le reali possibilità di recovery dipendono strettamente dalle capacità proprie dei vari servizi applicativi coinvolti.

In ogni caso, BTP richiede che esista un’entità, eventualmente creata all’occorrenza, che abbia accesso ai dati persistenti dell’entità caduta, e che possa ristabilire un collegamento con l’ex interlocutore di questa. Tale entità potrà avere lo stesso indirizzo di quella caduta, o un nuovo indirizzo, ma, dopo il recupero, dovrà avere un comportamento equivalente a quello che si avrebbe in caso di fallimento di comunicazione.

Nel seguito del paragrafo approfondiremo i meccanismi messi a disposizione da BTP, e cosa sia richiesto ai partecipanti di una transazione, affinché possa essere raggiunto un risultato consistente anche in presenza di *fallimenti di comunicazione* e/o *di nodo*.

Quanto si va a discutere, tuttavia, non deve essere inteso come un vincolo, bensì come una possibilità. BTP, infatti, riconosce l'autonomia delle applicazioni anche nella scelta del livello di persistenza: ad esempio, nella "parte normativa" della Specifica, assume che un'applicazione con soli dati volatili non ha necessità di rendere permanente alcuna informazione, tanto meno le informazioni di stato riguardanti il protocollo BTP.

Informazioni persistenti

BTP adotta il noto modello[66] dell' "Abort Presunto", e pertanto richiede che siano rese persistenti solo le seguenti informazioni:

1. la decisione di un inferiore di essere Prepared;
2. la decisione finale di Confirm di un superiore;
3. la decisione finale autonoma di un inferiore;

In caso di *fallimenti di nodo*, le prime due informazioni assicurano che sia possibile raggiungere una decisione consistente, mentre la terza, in presenza di contraddizioni, assicura che queste possano essere comunicate al Superior.

Rispetto al modello dell' "Abort Presunto" descritto in "1.2.1 Two Phase Commit e transazioni distribuite", BTP non richiede ad un inferiore la memorizzazione del record finale di Commit. Ciò è possibile poiché al suo posto richiede la cancellazione delle informazioni persistenti possedute dall'inferiore. Analogamente, il record di Complete (non richiesto dal modello dell' "Abort Presunto") viene sostituito dalla cancellazione delle informazioni mantenute dal Superior, cosicchè, anche in caso di caduta di questo, non sarà necessario ripetere la seconda fase del protocollo.

Ovviamente, per poter ristabilire la connessione, un Inferior dovrà memorizzare anche l'indirizzo e l'identificatore del superiore (sono entrambi contenuti nel BTPContext). Per il Superior, altre informazioni da mantenere permanentemente, oltre a quelle che dipendono dalla particolare applicazione, sono gli indirizzi e gli identificatori almeno

[66] Tale modello è stato descritto in "1.2.1 Two Phase Commit e transazioni distribuite".

degli inferiori ai quali si richiederà il *prepare*. Questi potranno essere utilizzati, dopo un'eventuale "caduta", per ripristinare le connessioni e richiedere lo stato corrente di ogni inferiore.

Potrebbe essere necessario rendere persistenti pure eventuali Qualificatori utilizzati nei messaggi di *prepared* o di *confirm*.

Infine, per recuperare le proprie funzionalità, un superiore o un inferiore dovranno memorizzare anche il proprio identificatore, ed eventualmente, in dipendenza dell'Implementazione, il proprio indirizzo del Trasporto.

Per quanto riguarda le informazioni da memorizzare relativamente all'applicazione, esse non sono definibili a priori. Tuttavia BTP assume che un inferiore che decida di essere *preparato* mantenga sufficienti informazioni per poter eseguire, in seguito ad un ripristino, sia il Provisional che il Counter Effect. BTP inoltre permette, ma non lo richiede, il recovery nello stato di Active: la scelta è lasciata alle particolari Implementazioni, e BTP non specifica nulla in proposito.

Le informazioni persistenti tenute da un Superior circa un inferiore, devono essere mantenute fino alla ricezione di un *confirmed*, *cancelled* o *hazard* dall'inferiore.

Un inferiore deve invece mantenere le proprie informazioni persistenti fino alla ricezione della decisione finale da parte del Superior (messaggio di *cancel* o *confirm*) o di un messaggio di *contradiction*.

Si noti che i meccanismi esposti consentono di recuperare lo stato di una Relazione Business tra Superior ed Inferiors, ma non anche lo stato delle altre relazioni. Per queste BTP non definisce nessun meccanismo esplicito.

Messaggi per il recovery

Anche per una relazione Superior-Inferior che sia già uscita dallo stato di Active, BTP non definisce messaggi speciali per recuperarla, ma permette semplicemente la rispedizione (ad esempio, per i messaggi di *confirm*, *cancel*, *prepare*, *prepared* e *confirmed*).

Non impone nemmeno quali tecniche debbano essere utilizzate per individuare un *fallimento di comunicazione*, ma suggerisce, ad esempio, la tecnica dei *timeout*.

BTP richiede semplicemente che alla ricezione di un messaggio duplicato venga rispedita la risposta, poiché questa potrebbe essere stata persa in precedenza. Il fatto che BTP non specifichi una sequenza di messaggi speciali per il recovery risiede nella volontà di lasciare massima libertà all'Implementazione, che può così avvantaggiarsi di eventuali meccanismi messi a disposizione dal particolare protocollo del Trasporto utilizzato.

Tuttavia, per il recupero della fase Active, seppure opzionale, non essendoci "ultimi" messaggi da rispedire, è necessario qualche meccanismo che consenta ad un attore di verificare l'esistenza del proprio interlocutore, e se questi sia ancora consapevole di essere parte di una relazione Superior-Inferior. A tal proposito BTP definisce dei nuovi messaggi, con cui richiedere lo stato e comunicare il proprio: *inferiorState* e *superiorState*.

Tali messaggi possono essere utilizzati pure come risposta nel caso in cui il ricevente di una richiesta non sia più attivo (ad esempio perché ha completato) o non esista (indicando lo stato di "unknown"). Inoltre, possono essere utilizzati anche quando, durante il recovery, la particolare Implementazione non riesca a determinare temporaneamente lo stato corrente o la disponibilità delle informazioni persistenti. In tal caso, nel messaggio, dovrà essere indicato lo stato di "inaccessible", che il ricevente potrà interpretare normalmente come un "ritenta più tardi". I valori di stato per *inferiorState* sono: Active, Inaccessible, Unknown. Per un messaggio di *superiorState*, ai precedenti va aggiunto il valore di PreparedReceived.

Infine, un qualsiasi servizio, e in qualsiasi momento, nel ruolo di StatusRequestor può interrogare un Superior o un Inferior con un messaggio di *requestStatus*, la cui risposta potrà essere un messaggio di *fault*, indicante che la richiesta non è stata accettata, o un messaggio di *status*, indicante lo stato corrente con uno dei seguenti valori[67]: Created, Enrolling, Active, Resigning, Resigned, Preparing, Prepared, Confirming, Confirmed, Cancelling, Cancelled, CancelContradiction, ConfirmContradiction, Hazard, Contradicted, Unknown, Inaccessible.

[67] I valori di stato elencati non sono tutti applicabili ad entrambi i ruoli di Inferior e Superior. Inoltre possono assumere un significato diverso in funzione del ruolo, secondo quanto specificato a pag 74 della Specifica.

Meccanismo di Redirection

Come precedentemente osservato, BTP utilizza il modello di "Presume-Abort" per il recovery, adottando il quale, in una relazione Superior-Inferior, gli interlocutori non memorizzeranno "sincronamente" le stesse informazioni durante il ciclo di vita della transazione. Questo accade, per esempio, se l'inferiore ha deciso di essere preparato, ma il superiore non ha ancora deciso di confermare; oppure se il superiore ha già confermato e l'inferiore ha conseguentemente rimosso le informazioni persistenti, ma il messaggio di *confirmed* non ha ancora raggiunto il superiore. In altri termini, in un dato istante, è sempre possibile che solo uno dei due interlocutori possieda informazioni persistenti riguardo la transazione. A seguito di un ripristino di sistema, potrebbe dunque accadere che il peer senza informazioni persistenti non sia più consapevole di essere ancora partecipe in una relazione Superior-Inferior.

Pertanto, nel caso in cui il peer che possiede informazioni persistenti cerchi di ristabilire, senza successo, una connessione precedentemente fallita, è importante che riesca a distinguere quando il problema sia dovuto all'inattività dell'interlocutore, piuttosto che ad un fallimento di comunicazione. Infatti, mentre nel primo caso il peer può dedurre il giusto stato di completamento della transazione[68], nel secondo deve attendere che la connessione possa essere ripristinata.

Per agevolare il compito, BTP mette a disposizione due particolari meccanismi:

- Al posto di un singolo indirizzo, un attore può avere un **set di indirizzi**. Il peer che cerca di ristabilire la connessione può utilizzare un indirizzo alternativo se il primo non dovesse rispondere, e tentare di capire se il problema risieda nell'interlocutore o nella rete.
- Un attore può utilizzare un messaggio di ***redirect***, con il quale può comunicare, al suo interlocutore, che il set di indirizzi precedentemente utilizzati per contattarlo non è più valido. Nello stesso messaggio può essere comunicato un nuovo set di indirizzi validi, e può essere invalidata anche solo una parte del set precedente.

[68] In dipendenza delle proprie informazioni persistenti e del fatto che l'interlocutore ne è privo, il peer che tenta di ristabilire la connessione può dedurre che l'interlocutore non è mai stato preparato, e non ha mai confermato, oppure che ha già completato la transazione ed ha rimosso in seguito le informazioni persistenti.

Quest'ultimo meccanismo si presta ad una moltitudine di necessità pratiche. Consente, ad esempio, che ad un indirizzo non più valido risponda un semplice Redirector, ovvero un agente con l'unica funzione di indirizzare al corretto indirizzo.

Le Implementazioni più semplici, inoltre, potrebbero utilizzare una singola entità, con indirizzo costante, che tratti con tutte le transazioni distinguendole dall'identificatore. In tal caso il messaggio di *redirect* potrebbe essere utilizzato per comunicare una "migrazione" permanente di tale entità.

Recovery della relazione Terminator-Decider

Per quanto riguarda la relazione Terminator-Decider, BTP non definisce alcun tipo di messaggio che ne possa agevolare il recovery. Affida alle Implementazioni eventuali scelte di questo tipo.

Tuttavia, poiché nella relazione Terminator-Decider quest'ultimo non ha modo di conoscere se il Terminator sia ancora attivo, BTP definisce un Qualificatore standard, denominato "transactionTimelimit", con cui il Terminator può indicare al Decider per quanto tempo esso rimarrà, con buona probabilità, attivo. Trascorso tale tempo senza aver ottenuto un messaggio di conferma o di annullamento, il Decider potrà valutare se avviare autonomamente l'annullamento della transazione.

Contradictions e Hazard

Come precedentemente detto, essendo possibili decisioni autonome, pianificate o meno, da parte degli inferiori, sono possibili anche contraddizioni. Una contraddizione può presentarsi tra la decisione del superiore e quella anticipata dell'inferiore (per una coesione, la decisione del superiore si intende relativa ad ogni singolo inferiore).

Per avere certezza che una contraddizione sia riconosciuta anche in caso di fallimenti, BTP richiede che la decisione autonoma sia memorizzata in maniera persistente fino a quando non arrivi un messaggio dal Superior che indichi assenza di contraddizione (messaggio di *cancel*/*confirm*), o presenza di contraddizione (messaggio di *contradiction*). In ogni caso l'inferiore diventerà cosciente che la condizione anomala è nota anche al superiore.

Le azioni del Superior in caso di contraddizione non sono definite dalla Specifica. In particolare, anche per un Intermediate, BTP non richiede che questi avvisi del fatto il suo diretto superiore, in quanto quest'ultimo potrebbe appartenere ad un'organizzazione differente. Tuttavia, vengono messi a disposizione dei meccanismi per comunicare il problema sia verso un Superior, che verso un Terminator. Nel primo caso la contraddizione può essere comunicata con un messaggio di *hazard*, nel secondo con un messaggio di *inferiorStatutes*.

Un messaggio di *hazard* può essere utilizzato anche in altre due circostanze. Quando un Intermediate abbia inviato la decisione finale ai propri inferiori, ma non abbia ancora ottenuto un riscontro da questi entro un tempo limite, può decidere di attendere oltre, o può decidere di informare il diretto Superior della situazione. In questo secondo caso può inviare al Superior un messaggio di *hazard*, indicando che la contraddizione non è certa, ma possibile. Tale condizione di incertezza deve essere anch'essa memorizzata permanentemente dall'Intermediate, fino a quando non arrivi un messaggio di *contradiction* dal Superior, o un messaggio di riscontro dall'inferiore che aveva determinato la particolare condizione.

La seconda circostanza in cui un messaggio di *hazard* può essere utilizzato, è quando un attore sia impossibilitato a raggiungere uno stato consistente di Cancel o di Confirm. Tuttavia, come sottolineato dalla Specifica, in tal caso potrebbe non essere possibile una segnalazione affidabile del problema, poiché la causa di questo potrebbe risiedere proprio nell'incapacità di memorizzare informazioni in maniera persistente.

2.1.4 Qualificatori Standard

Oltre ai Qualificatori "transactionTimelimit" e "inferiorTimeout" precedentemente accennati, BTP definisce altri due Qualificatori standard. Il loro significato preciso, compresi i primi due, è dato di seguito:

TransactionTimelimit

Permette ad un Superior (o Initiator) di indicare per quanto tempo esso si aspetta che durerà la fase di Active, dando così ad un Inferior la possibilità di avviare la

"cancellazione" nel caso in cui tale fase dovesse prolungarsi per "troppo" tempo. Il tempo limite indicato nel Qualificatore in questione perde di validità quando l'inferiore decida di essere preparato, ed invii un messaggio di *prepared* al Superior. Ad ogni modo, lo scadere del Timelimit è solo un'indicazione, poiché l'Inferior, per motivi interni, può avviare l'esecuzione della "cancellazione" in qualsiasi momento.

InferiorTimeout[69]

Permette ad un Inferior di limitare la durata per il quale "promette" di mantenere l'abilità a confermare o annullare, in seguito all'invio di un messaggio di *prepared*. Se tale qualificatore non è presente, l'Inferior dovrebbe mantenere la sua promessa indefinitivamente. Nel caso in cui il Timeout scada, l'inferiore dovrebbe prendere la decisione che aveva indicato in questo stesso Qualificatore (in un campo dell'elemento XML con cui il Qualificatore si identifica).

Se l'inferiore dovesse essere costretto ad assumere un comportamento differente da quello appena esposto, l'evento deve essere trattato come una decisione euristica.

MinimumInferiorTimeout

Permette ad un Superior di limitare il valore dell' "inferiorTimeout". Se il superiore conosce già il tempo minimo che impiegherà prima di prendere una decisione finale, può inviare tale Qualificatore nel messaggio di *prepare*. L'inferiore che non voglia attendere per il tempo indicato, potrà ritirarsi dalla transazione con un messaggio di *cancelled*.

InferiorName

Permette all'arruolante di specificare un nome simbolico (stringa) per l'inferiore che va ad arruolare. A differenza di un Identificatore, tale nome non deve avere nessun requisito di univocità. Può essere utilizzato per essere facilmente comprensibile da un

[69] In Riferimenti[28] gli autori del documento, fonte Microsoft-IBM, criticano il fatto che il qualificatore InferiorTimeout viene comunicato solo al coordinatore, e non anche al richiedente, che avrebbe una maggiore utilità nel conoscere la durata della "promessa di *pronto*" del partecipante. La critica è tuttavia ingiustificata, poichè nulla, in BTP, impedisce di inserire tale qualificatore anche nel messaggio di *risposta applicativa*.

operatore umano, e dunque per aiutare nel tracing, nel debugging, o nella selezione dei partecipanti in una Cohesion.

2.1.5 Binding

Le caratteristiche di BTP, seppure valide in un ambito più generale, lo rendono un protocollo particolarmente indicato per l'ambiente dei Web Services. Per questo motivo, oltre alla definizione astratta dei messaggi in XML, la Specifica fornisce anche gli "XML Schema" e il Binding per SOAP. Tuttavia, non impone un particolare protocollo di Trasporto, e né impone un particolare formato sulla modalità di trasmissione dei messaggi applicativi o BTP. Definisce i messaggi in XML; come questi siano scambiati, ad esempio tramite http o SMTP, è un problema applicativo. Richiede semplicemente che i messaggi BTP possano essere inviati ad un attore BTP, il cui indirizzo del Trasporto sia noto al mittente in qualche maniera, e che un messaggio arrivi correttamente o non arrivi del tutto.

Suo obiettivo è quello di fornire la massima flessibilità, permettendo alle Implementazioni la definizione del Binding sul protocollo di Trasporto voluto.

Come vedremo tra breve, gli stessi campi "indirizzo" previsti in ogni messaggio del protocollo, nel caso in cui siano implicitamente presenti nei sottostanti meccanismi di Trasporto, possono essere omessi nei messaggi BTP.

I messaggi BTP vengono scambiati in tre circostanze distinte:

a) tra elementi applicativi ed Agenti BTP;
b) tra Agenti BTP;
c) associati a messaggi applicativi, e tra elementi applicativi (come accade per il BTPContext).

Abbiamo già osservato che BTP non definisce i messaggi scambiati nel modo a). Per quelli scambiati nel modo b), la Specifica di BTP ne definisce la semantica completa, e la sintassi, in forma di messaggi astratti rappresentati in XML.

Per i messaggi comunicati nel modo c), invece, la Specifica si limita a definirne la sintassi, sempre in XML, mentre assume che il significato dipenda dalla particolare applicazione. Ad esempio, un messaggio di *enrol* inviato all'interno di un messaggio applicativo può significare (come da contratto applicativo e non BTP), che tale applicazione è pronta per fornire un qualche lavoro. Oppure, dopo la ricezione di un BTPContext, il contratto applicativo potrebbe prevedere che il ricevente arruoli degli inferiori per effettuare un certo lavoro.

Inoltre la Specifica non definisce come possa essere ottenuta l'associazione tra messaggi BTP e messaggi applicativi, ma fornisce alcuni suggerimenti pratici:

- I messaggi BTP possono essere contenuti nei messaggi applicativi, o entrambi in un costrutto superiore.
- Il messaggio applicativo può contenere un campo che indica lo stesso identificativo (del Superior o dell'Inferior) presente anche nel messaggio BTP associato.
- Il messaggio BTP può contenere un campo che fa riferimento ad un messaggio applicativo in qualche maniera.

In ogni caso, qualsiasi sia la modalità scelta, BTP richiede che le parti concordino[70] sulle modalità di trasmissione e sul significato, affinché possano capirsi vicendevolmente.

Da quanto detto precedentemente circa l'arbitrarietà di "Trasporto", segue anche che messaggi applicativi e messaggi BTP possono eventualmente utilizzare un protocollo di Trasporto differente. Nella pratica, tuttavia, è lecito supporre che sia utilizzato generalmente lo stesso protocollo. Questo è il caso della "Specifica di binding" per Web Services fornita dall'edizione di BTP presa in esame, che definisce il binding per "SOAP 1.1 over HTTP". Questa stabilisce le regole per trasportare i messaggi e per relazionare un messaggio BTP ad un messaggio applicativo: ad esempio, stabilisce che i messaggi di *BTPContext* e di *enrol* siano trasportati nell'*header* SOAP, mentre i messaggi applicativi siano contenuti nel *body* SOAP.

[70] Potrebbe essere utilizzato, ad esempio, WS-Policy (Riferimenti[19])

Ogni messaggio BTP, oltre a "contenere" i campi propri del messaggio, contiene uno o più campi indirizzo. Un **indirizzo BTP** è formato da tre elementi[71]: un identificatore per il protocollo sottostante (il protocollo di Trasporto); un "binding address", nel formato richiesto dal particolare Binding; e un campo opzionale, che può essere utilizzato, ad esempio, per inserire la versione del protocollo del Trasporto.

Gli indirizzi BTP che devono essere "contenuti" in un messaggio BTP sono:

- targetAddress: è contenuto in tutti i messaggi di protocollo, e indica a chi è rivolto il messaggio.
- replyAddress: è contenuto in tutti i messaggi che richiedono una risposta, e che non contengono un senderAddress.
- senderAddress: è contenuto solo in alcuni messaggi scambiati nella relazione Superior-Inferior e nel messaggio di *enrolled*.

Il "replyAddress" e il "senderAddress", praticamente, possono svolgere la stessa funzione: indicare dove inviare una risposta. Tuttavia il "senderAddress" ha un significato diverso: viene utilizzato nei messaggi scambiati nella relazione Superior-Inferior affinché il ricevente, nel caso in cui abbia dimenticato le informazioni circa la propria partecipazione in detta relazione, possa inviare al mittente "sconosciuto" un appropriato messaggio di errore. Di conseguenza il "senderAddress" è contenuto anche nei messaggi di "reply", non solo in quelli di "request", come accade per il "replyAddress".

Come precedentemente anticipato, i campi indirizzo potrebbero essere già presenti nel sottostante protocollo del Trasporto, ed in tal caso possono essere omessi dai messaggi BTP.

[71] Si noti che un indirizzo BTP va considerato, a tutti gli effetti, come un'astrazione di un indirizzo del Trasporto. Esso è pertanto associato ad un particolare attore. L'Identificatore precedentemente definito a pag.110 è associato, invece, ad un determinato ruolo, intrapreso da un Attore con un certo indirizzo.

2.2 Web Services Transactions

L'alternativa proposta da Microsoft, IBM e BEA Systems, in supporto alle Transazioni Distribuite nell'ambito dei servizi Web, era stata inizialmente stilata in un documento di specifica unico: WS-Transactions.

Questo, successivamente, è stato suddiviso in tre sotto-Specifiche distinte, denominate rispettivamente WS-Coordination, WS-AtomicTransaction e WS-BusinessActivity.

WS-Coordination definisce un modello per la condivisione di un Contesto generico tra servizi Web. Per Contesto si intende semplicemente un'entità atta a contenere un'insieme di informazioni che rappresentano, in qualche maniera, un'attività distribuita.

Le altre due Specifiche, a partire dal modello definito nella prima, definiscono invece particolari protocolli per la coordinazione transazionale.

In particolare **WS-AtomicTransaction** fornisce un servizio di coordinamento per transazioni di tipo tradizionale, mentre **WS-BusinessActivity** si rivolge a servizi che richiedono un coordinamento di tipo "rilassato".

A differenza di BTP le tre Specifiche in questione nascono direttamente per la tecnologia dei Web Services: vengono forniti gli XML Schema e la descrizione WSDL di tutti i servizi definiti e, basandosi le Specifiche su WS-Addressing, viene fornito implicitamente il binding per SOAP. Oltre WS-Addressing già menzionata, molti altri Standards/Specifiche, nati per detta tecnologia, vengono spesso richiamati e utilizzati secondo convenienza.

Nel seguito, quando sarà necessario fare riferimento alle tre Specifiche nel loro insieme, utilizzeremo brevemente il termine "WS-Transactions" o "WS-Tx", e faremo riferimento ad esso come ad una Specifica unica.

2.2.1 WS-Coordination

La composizione dei servizi Web, per ottenere applicazioni che forniscano maggiore flessibilità all'utente, è, oggi, una pratica in continua evoluzione, e sempre più diffusa. La "collaborazione" che nasce in seguito a detta composizione, porta alla formazione di grandi unità computazionali, distribuite tra vari partecipanti. Queste sono denominate brevemente Activities in WS-Coordination, e nel seguito del presente testo le chiameremo anche Attività.

Essendo i lavori applicativi, costituenti un'Attività, distribuiti su macchine (nodi) distinte, serve un qualche meccanismo di associazione tra ogni lavoro, e l'Attività stessa di cui è parte.

Inoltre, affinché un servizio Web possa richiedere ad un altro servizio l'esecuzione di un lavoro come parte di una certa Attività, il meccanismo deve consentire anche l'identificazione dell'Attività in questione, e la condivisione del corrispondente identificativo. Infine, spesso è necessario che l'esecuzione dei singoli lavori sia coordinata in qualche maniera. Ad esempio, per il raggiungimento di un risultato consistente, così come si richiede alle Attività di tipo transazionale, è sempre necessaria una qualche forma di coordinazione.

WS-Coordination definisce un framework per gestire la composizione di applicazioni distribuite, estensibile rispetto ai protocolli di coordinamento. Tale framework, per la "propagazione" di un'Attività ad altri servizi, si basa sul meccanismo della condivisione del Contesto (analogo a BTP), di cui la Specifica ne definisce la struttura e i requisiti generali.

Inoltre, nell'intento degli autori, il supporto per il coordinamento è rivolto potenzialmente verso qualsiasi tipo di Attività, non necessariamente di tipo transazionale. [72]

[72] Probabilmente questo è il motivo per cui la Specifica originaria di WS-Transactions è stata suddivisa.

2.2.1.1 Modello Concettuale

Il meccanismo definito in WS-Coordination, come già osservato, si basa sulla condivisione del *Contesto*. Ad ogni Attività è associato un CoordinationContext, che la rappresenta formalmente mantenendone i "connotati", e ogni lavoro può essere richiesto come parte di un'Attività inviando il CoordinationContext insieme alla richiesta applicativa. Il CoordinationContext è gestito dal CoordinatorService, il quale fornisce tre servizi concettualmente distinti:

- **ActivationService**: un servizio di attivazione; consente alle applicazioni di creare un nuovo Contesto, e dunque una nuova Attività.
- **RegistrationService**: un servizio di registrazione; consente ad un servizio di essere registrato come partecipante in un'Attività, e per un particolare protocollo di coordinamento.
- **ProtocolService**: un servizio che attua da coordinatore, utilizzando per ogni partecipante i protocolli specificati all'atto della registrazione.

WS-Coordination non definisce nessun protocollo di coordinamento specifico, ma considera il ProtocolService come un servizio "plug-in", da inserire nel CoordinatorService a seconda delle necessità.

Inoltre, ogni ProtocolService può supportare un certo numero di protocolli diversi, e ogni Implementazione del CoordinatorService può scegliere quali ProtocolServices supportare.

Il **CoordinationContext** contiene i seguenti campi:

- Identifier: identificatore univoco dell'Attività.
- CoordinationType: il tipo di coordinatore (ProtocolService) utilizzato per la particolare Attività.
- RegistrationService: informazioni relative al RegistrationService che potrà essere utilizzato per gli arruolamenti dei partecipanti.
- Expires: la durata dell'Attività (opzionale).
- Extentions: eventuali estensioni.

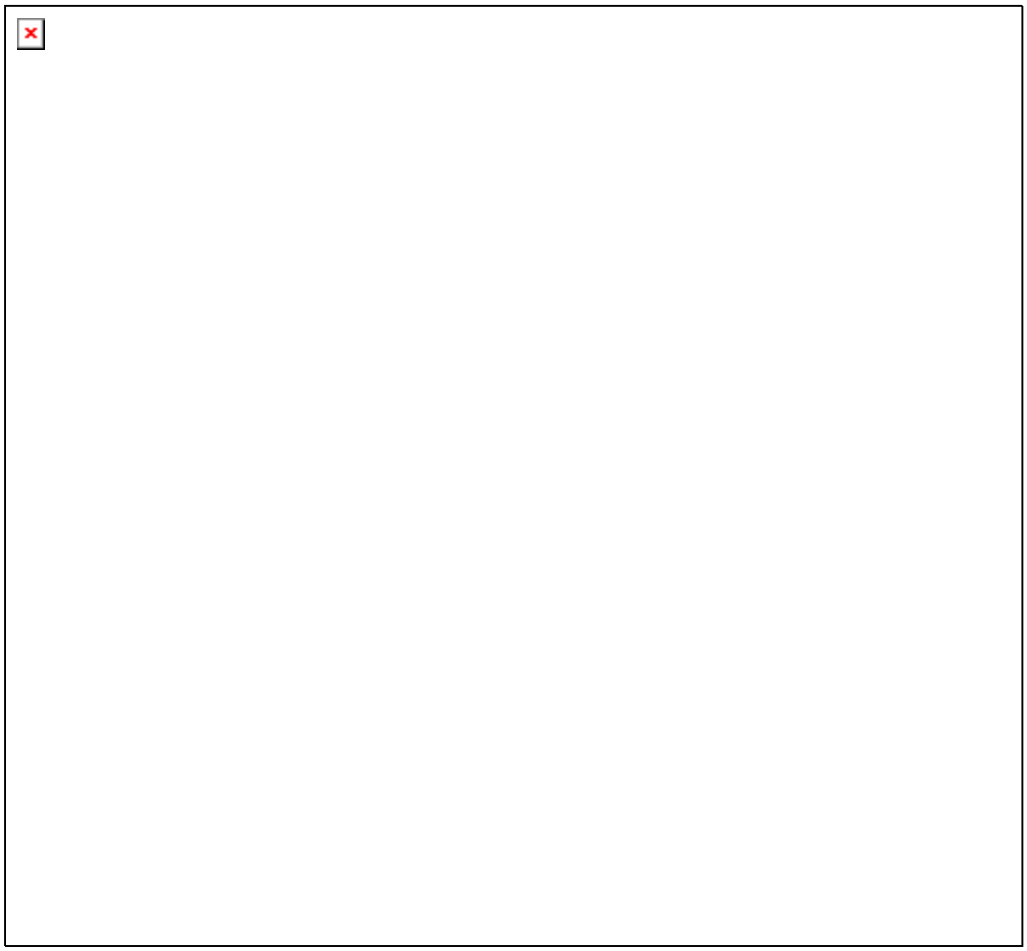

Figura 33 Modello Concettuale di WS-Coordination

In Figura 33 viene mostrato un possibile modello concettuale di WS-Coordination. In essa l'entità ApplicationService rappresenta il generico servizio Web dell'utente.

Seppure, come osservato, la Specifica non sia rivolta esclusivamente ad Attività transazionali, è ancora possibile parlare di *richiedente* ed *esecutore*, con significato analogo. Nel modello in Figura 33, l'entità che attua nel ruolo di *richiedente* non è tuttavia distinta da quella che attua nel ruolo di *esecutore*: entrambe sono rappresentate dall'ApplicationService, che svolge il ruolo di *esecutore* (Participant in figura), nel momento in cui esegue qualche lavoro all'interno di un'Attività. Il ruolo di Participant è invece sempre intrapreso dall'entità Interposed che, insieme a tale ruolo, assume anche quello di servizio coordinatore. In proposito, WS-Coordination vincola un coordinatore

Interposed ad essere dello stesso[73] tipo del coordinatore "padre". Si noti che, a differenza di BTP, il modello in Figura 33 non mostra l'entità Interposed come una generalizzazione dell'ApplicationService. Infatti, un Interposed, pur partecipando alle stesse relazioni dell'ApplicationService, non contiene logica applicativa, ma costituisce solo un elemento di intermediazione.

2.2.1.2 Protocolli

Un servizio che voglia creare un'Attività, può richiederne l'attivazione all'ActivationService mediante un messaggio di *createCoordinationContext*. La richiesta deve specificare:

- il tipo di coordinatore richiesto per l'Attività;
- il tempo massimo che si attenderà per la risposta sull'esito dell'attivazione (opzionale);
- un CoordinationContext, quando il coordinatore debba fungere da intermediario (opzionale); Tale coordinatore potrà avere un proprio servizio di attivazione, e un proprio servizio di registrazione ma, come detto in precedenza, dovrà sempre essere dello stesso tipo del coordinatore "padre".

In risposta viene restituito il nuovo CoordinationContext appena creato. Questo riporta lo stesso identificatore di Attività presente nel Contesto "padre", contiene informazioni per accedere al servizio di registrazione, indica il particolare tipo di ProtocolService, e contiene campi di estensione (ad esempio, con essi si potrebbe indicare il livello di isolamento richiesto dall'applicazione).

Il CoordinationContext dovrà essere propagato, insieme alle richieste applicative, verso i servizi Web cui si richiede un qualche lavoro nell'ambito dell'Attività. Un servizio che sia disponibile alla collaborazione, una volta ottenuto il CoordinationContext, può utilizzare le informazioni in esso contenute per richiedere la registrazione presso il

[73] La Specifica non è del tutto chiara, ma nell'unico riferimento alla questione, ovvero al punto 3 di pag. 8, riporta un esempio dal quale si deduce il vincolo sull'eguaglianza tra i tipi dei coordinatori "padre" e "figlio". Quanto detto è avvalorato da un ulteriore documento ancora di fonte Microsoft-IBM, riportato in Riferimenti[28], dove, al quarto capoverso, viene esplicitamente criticata la possibilità di Interposizione mista offerta da BTP.
Una curiosità: la critica richiamata, a nostro parere, non è giustificata. Nella nota 127, a p301, spiegheremo il motivo.

RegistrationService, mediante un messaggio di *register*, in cui deve specificare il comportamento di coordinazione che vorrà seguire, e l'indirizzo del servizio "partecipante" che dovrà essere coordinato. Per la registrazione può essere utilizzato il RegistrationService indicato nel CoordinationContext ricevuto, oppure il RegistrationService di un CoordinatorService di propria fiducia, che, in tal caso, fungerà da *intermediario*. In risposta, dal RegistrationService, viene comunicato un messaggio di *registrationResponse*, contenente l'indirizzo del ProtocolService a cui il partecipante registrato dovrà fare riferimento durante l'esecuzione del protocollo di coordinamento.

Tutti i messaggi suddetti, possono inoltre essere estesi mediante informazioni dipendenti dalla particolare applicazione. Ad esempio, per Attività transazionali potrebbero essere aggiunte informazioni relative al livello di isolamento supportato/richiesto, e più in generale, informazioni relative alle modalità di autenticazione supportate, al protocollo di coordinamento utilizzato, etc.

In seguito alla registrazione di tutti i partecipanti, la composizione dell'Attività sarà terminata: ogni attore coinvolto avrà un riferimento agli indirizzi con cui dovrà comunicare[74], e l'Attività potrà essere avviata e coordinata secondo i protocolli di coordinamento scelti.

[74] WS-Coordination descrive questa situazione, in maniera equivalente, dicendo che < tutte le connessioni logiche> sono state stabilite. Sottolineiamo la terminologia adottata dalla Specifica poiché si parlerà ancora di <connessioni>, in WS-AT, ma in un contesto poco chiaro.

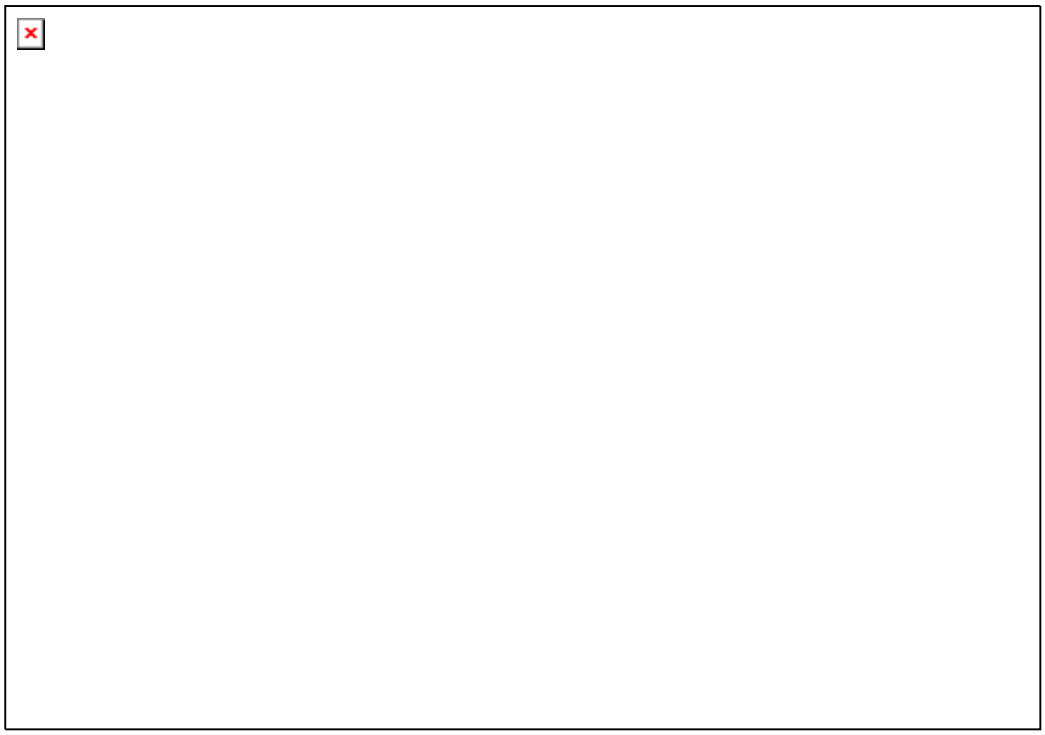

Figura 34 Modello delle Interfacce di WS-Coordination

In Figura 34 è riportato un modello di secondo livello, nel quale vengono mostrate le interfacce dei servizi. In questo sono state evidenziate, con colori diversi, rispettivamente rosa e verde, l'interfaccia che deve implementare un ApplicationService per svolgere il ruolo di *richiedente*, e quella che deve implementare per svolgere il ruolo di *esecutore*.

Sulla base del modello appena discusso, il Sequence Diagram nella figura seguente mostra le interazioni principali nel semplice caso di un'Attività con un solo partecipante, quando questo decida di utilizzare lo stesso CoordinatorService indicato nel Contesto ottenuto dal *richiedente*.

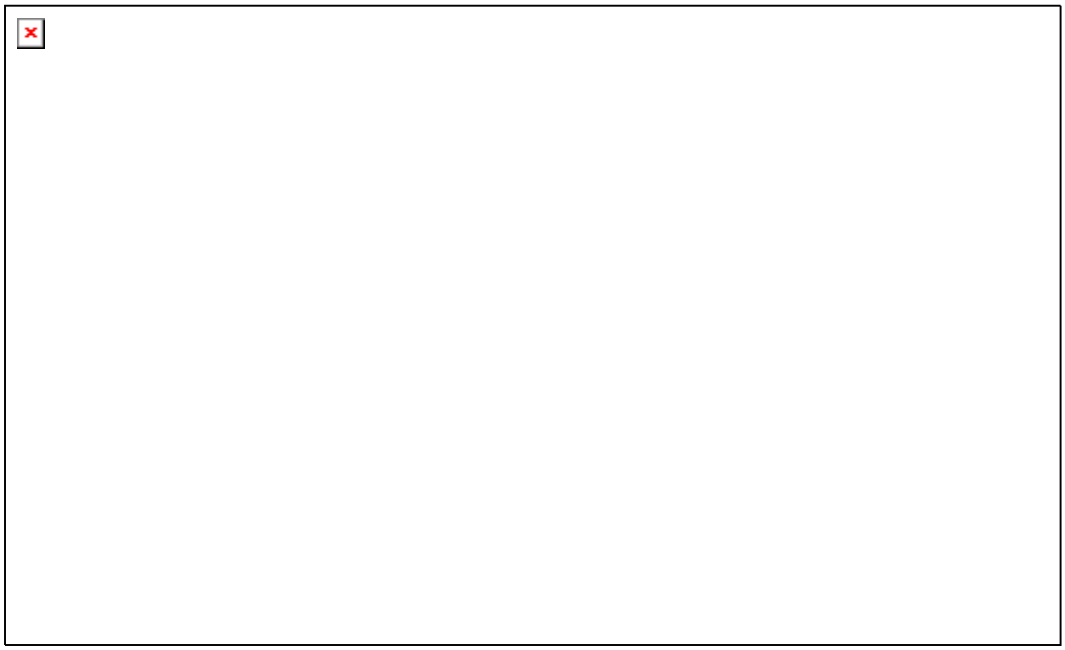

Figura 35 Interazioni base in WS-Coordination

In tale diagramma, con “request” e “response” sono state indicate rispettivamente la richiesta e la risposta applicativa.

Si noti che il tipo di arruolamento mostrato segue il modello “pull”. Per un arruolamento di tipo “push”, seppure possibile, sarebbero necessarie delle interazioni aggiuntive, a cura dell’Implementazione, affinché il *richiedente* possa comunicare almeno l’indirizzo del ProtocolService, contenuto nel messaggio di *registrationResponse*, al servizio *esecutore* arruolato.

2.2.1.3 Failure Handling

WS-Coordination definisce una serie di messaggi[75] con cui gli attori possono comunicarsi vicendevolmente particolari condizioni di errore:

- *invalidState*: comunicato all'interlocutore quando il partecipante o il coordinatore entra in uno stato non valido.
- *invalidProtocol*: comunicato all'interlocutore quando il partecipante o il coordinatore riceve un messaggio non valido per un dato protocollo.
- *invalidParameters*: comunicato all'interlocutore quando il partecipante o il coordinatore riceve un messaggio valido, ma con parametri non validi.
- *noActivity*: comunicato dal coordinatore verso un partecipante che non mostri attività da "troppo tempo" [76].
- *contextRefused*: inviato ad un coordinatore per indicare che il Contesto non può essere accettato dal partecipante che lo ha ricevuto.
- *alreadyRegistered*: inviato dal coordinatore ad un partecipante che tenti di registrarsi più volte per la stessa attività e per lo stesso protocollo.

La Specifica assume che tutti i messaggi, sia quelli di errore descritti in questo paragrafo, sia quelli ordinari, possano essere rispediti quando il mittente lo ritenga necessario. Le condizioni per le quali un messaggio debba essere rispedito, pertanto, non sono specificate da WS-Coordinator, ma, in WS-ReliableMessaging, sono precisate le modalità con cui tale rispedizione debba avvenire, mentre nelle Specifiche di Supporto al modello[77] adottato per la Sicurezza, sono precisate le tecniche da adottare per evitare "attacchi alla disponibilità".

[75] Per semplicità, i messaggi di errore non vengono mostrati in Figura 34.
[76] Nella Specifica non viene precisato il significato di "troppo tempo".
[77] Il modello per la Sicurezza sarà descritto nel paragrafo "2.2.5 Modello per la Sicurezza".

2.2.2 WS-AtomicTransaction (WS-AT)

Estende WS-Coordination basandosi sul modello classico del 2PC, per permettere la coordinazione di Attività transazionali che richiedano la proprietà del "tutti o nessuno".

A tal proposito definisce un particolare tipo di ProtocolService, denominato "Atomic Transaction", che nel seguito del capitolo chiameremo brevemente ATCoordinator o più semplicemente coordinatore.

Come gli autori della Specifica precisano, questo tipo di coordinazione si rivolge ad Attività di corta durata, nelle quali esista un alto livello di fiducia[78] tra le parti coinvolte.

2.2.2.1 Modello Concettuale

WS-AtomicTransaction definisce due specifici protocolli per transazioni atomiche: il protocollo Completition, che realizza la relazione tra l'ApplicationService nel ruolo di Initiator, ovvero di *richiedente*, e l'ATCoordinator; e il protocollo 2PC, che realizza la relazione tra l'ATCoordinator e i partecipanti di tipo ATParticipant. Quest'ultimo rappresenta semplicemente la specializzazione di un Participant che supporta il protocollo 2PC in questione. Definisce inoltre due tipi di ATParticipant, assieme alle corrispondenti varianti del protocollo 2PC:

- **VolatileParticipant**: utilizza risorse "volatili", ad esempio una cache.
- **DurableParticipant**: utilizza risorse "permanenti", ad esempio un database.

Lo stesso ApplicationService può essere registrato per più di un protocollo, inviando più richieste al servizio di registrazione[79]: ad esempio può essere registrato contemporaneamente sia come partecipante Volatile, che come partecipante Durable.

[78] Il "livello di fiducia" richiesto va interpretato in relazione al livello di Isolamento "imposto" dal particolare modello di coordinazione definito da WS-AT.

[79] Potendo l'ApplicationService essere registrato per più di un protocollo contemporaneamente, l'ATPartecipant dovrebbe essere considerato più propriamente come un'entità, con un proprio identificativo, al fine di poter essere indirizzato singolarmente. Tuttavia, in accordo alla Specifica

In Figura 36 è riportato un possibile modello concettuale concorde con la descrizione appena esposta. Le entità definite in WS-Coordination, ed estese/specializzate da WS-AtomicTransaction, sono evidenziate in giallo. Il modello, per semplicità, non mostra la specializzazione della relazione "Prt. Coordinamento", definita in WS-Coordination, anche per un Interposed, portando, questa, alle stesse relazioni mostrate per l'ApplicationService.

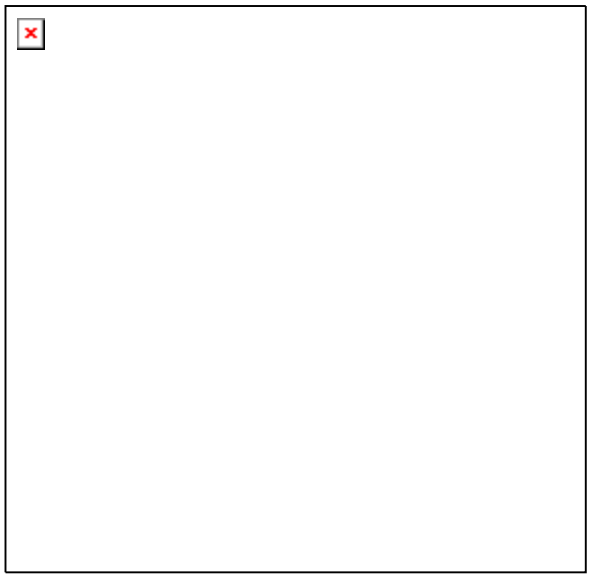

Figura 36 Modello Concettuale di WS-AtomicTransaction

WS-AtomicTransaction precisa la semantica del messaggio, definito in WS-Coordination, con cui è possibile richiedere la creazione di un nuovo Contesto:

- se la richiesta include l'elemento CurrentContext, il coordinatore sarà un Interposed, ovvero un subordinato[80];
- se la richiesta non include tale elemento, il coordinatore creerà una nuova Transazione e attuerà come "root".

Precisa inoltre che l'attributo di Expired, contenuto nel CoordinationContext, indica ora l'intervallo di tempo entro il quale la transazione dovrebbe terminare. Se il coordinatore

originaria, la questione è interna all'ApplicationService, e dunque il concetto di Participant è stato rappresentato come un ruolo.

[80] Come specificato in WS-Coordination, il coordinatore Interposed potrà essere solo di tipo AT.

non dovesse ricevere una decisione entro tale intervallo, può avviare il rollback della transazione.

2.2.2.2 Protocolli

La Figura 37 riassume i messaggi coinvolti nei protocolli che si andranno a descrivere. Come per il Modello Concettuale da cui deriva, anche in questo modello di secondo livello l'entità Interposed può essere sostituita all'ApplicationService.

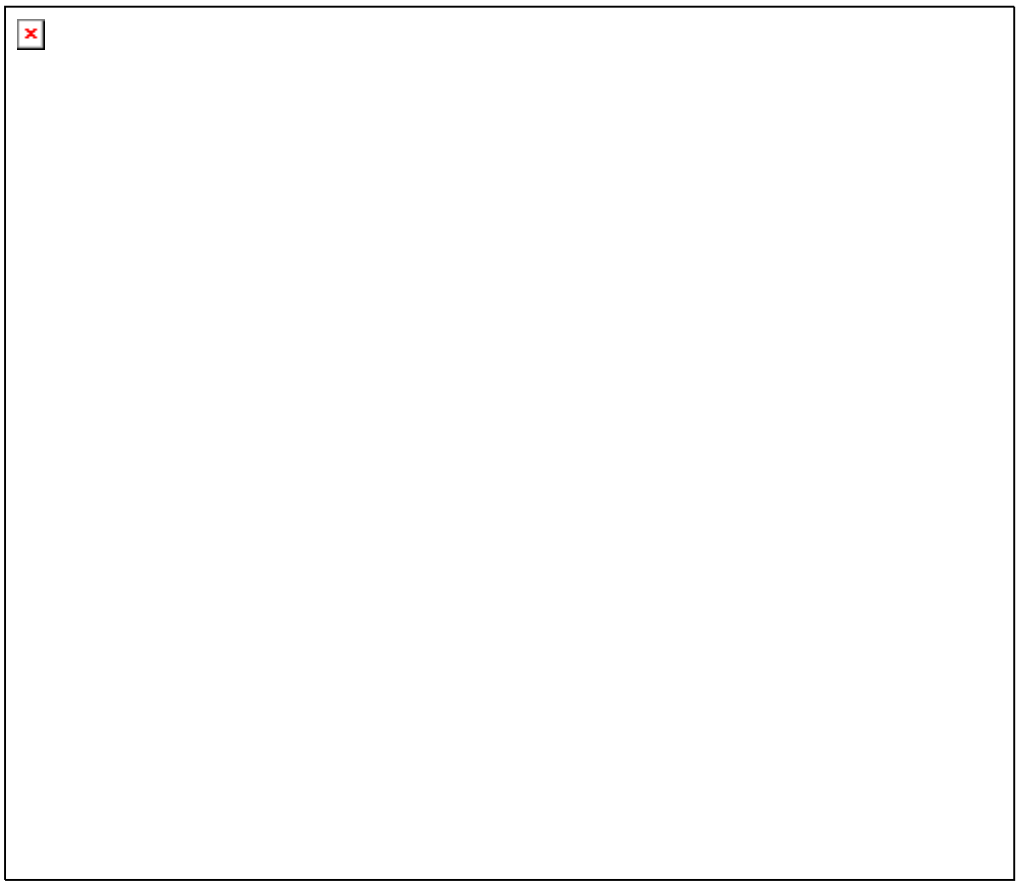

Figura 37 Modello delle Interfacce di WS-AtomcTransaction

2.2.2.2.1 Protocollo Completition

Un servizio, nel ruolo di Initiator, può utilizzare questo protocollo per richiedere l'avvio della coordinazione di un'Attività transazionale, precedentemente creata secondo WS-Coordination, e per ricevere informazioni circa l'esito della coordinazione richiesta.

Uno schema astratto del semplice protocollo è presentato in figura:

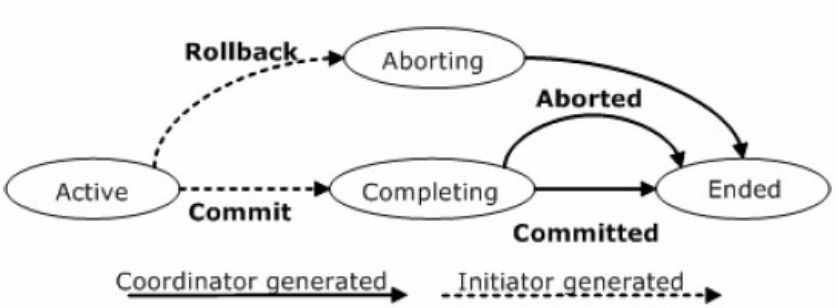

Figura 38 Protocollo Completition
-tratto da Riferimenti[6 p6]-

In essa viene mostrato lo State Diagram della transazione in funzione dei messaggi scambiati tra l'Initiator e l'ATCoordinator. WS-AT specifica il significato dei messaggi rappresentati come di seguito esposto.

Dopo che il coordinatore ha ricevuto:

- *commit*: sa che i partecipanti hanno completato il loro lavoro applicativo, e che dovrebbe tentare il Commit della transazione;
- *rollback*: sa che i partecipanti hanno completato il loro lavoro applicativo, e che dovrebbe abortire la transazione.

Dopo che l'Iniziator ha ricevuto:

- *committed*: sa che il coordinatore ha raggiunto la decisione di Commit;
- *aborted*: sa che il coordinatore ha raggiunto la decisione di Abort.

Si noti, in quanto detto, che gli ultimi due messaggi fanno riferimento esplicitamente alla decisione finale del coordinatore, piuttosto che allo stato finale della transazione. In altri termini, WS-AT assume che le due cose debbano necessariamente coincidere, proprio come avviene nei DBMS transazionali.

2.2.2.2.2 Protocollo 2PC

Corrisponde in pratica al protocollo 2PC tradizionale, con alcune semplici variazioni nel comportamento del coordinatore in funzione del tipo di partecipante che, come precedentemente detto, può essere *durable* o *volatile*.

La differenza di coordinazione nei confronti dei due tipi, consiste nel fatto che i partecipanti registrati come VolatileParticipant vengono avviati per primi alla fase di Prepare: tutti i partecipanti di questo tipo devono rispondere prima che un messaggio di *prepare* sia inviato ai partecipanti *durable*. Inoltre, altri VolatilePaticipants possono essere arruolati per la stessa transazione fino a quando il coordinatore non abbia inviato il primo *prepare* ad un partecipante *durable*. Tuttavia, per un partecipante *volatile*, non vi è alcuna garanzia che sia messo al corrente della decisione finale.

Quando la fase di Prepare per i VolatileParticipants viene conclusa, il coordinatore avvia dunque la stessa fase per i DurableParticipants. Solo quando tutti i partecipanti abbiano risposto con un messaggio di *prepared* o *readOnly*, il coordinatore potrà avviare la seconda fase richiedendo il Commit.

In proposito all'utilità della distinzione fatta tra i due tipi di ATParticipant, WS-AT non è di chiarimento. Tuttavia è possibile trovare delle spiegazioni in un altro documento[81], sempre di origine Microsoft. Tale documento riporta un esempio[82] concreto in cui

[81] Riferimenti[32 cap. 4.3]

[82] L'esempio riguarda quelle applicazioni Web, oggi molto comuni, che seguono un'architettura a strati. In particolare, per architetture "three tiers", molto spesso esiste uno strato "front-end" di presentazione, uno strato intermedio che implementa la logica applicativa, e uno strato di "back-end" che gestisce i dati persistenti dell'applicazione appoggiandosi su una base di dati. Lo strato intermedio, per questioni di efficienza, svolge comunemente anche la funzione di "cache" nei confronti della base-dati. Se la base-dati sul server di "back-end" venisse utilizzata da un servizio Web, nel ruolo di partecipante "durable" in una "Atomic Transaction", servirebbe qualche meccanismo per forzare l'aggiornamento dalla "cache", affinché i dati utilizzati dal partecipante non siano "obsoleti". In tal caso, un altro partecipante, registrato come VolatileParticipant nella stessa transazione, può richiedere all'applicazione residente sul server intermedio il "flushing" della cache, e WS-AT assicura che tale partecipante sia contattato per primo nell'esecuzione del protocollo 2PC.
La soluzione proposta presenta tuttavia un problema. Infatti il "flushing" della cache dovrebbe avvenire prima dell'esecuzione del protocollo 2PC, ovvero quando il servizio Web che vi accede riceve la richiesta applicativa, e non durante il protocollo di coordinamento, quando i dati "obsoleti" sono stati ormai già utilizzati. Inoltre, non si capiscono le ragioni perché il servizio partecipante che accede alla base-dati non possa contattare esso stesso il server della "cache" per richiederne il "flushing", e soprattutto quando sia realmente opportuno farlo.

dovrebbe essere utile il ruolo di VolatileParticipant. Purtroppo, analizzando l'esempio, è facile rendersi conto che la soluzione in esso proposta presenta alcuni problemi, e che tali problemi non sarebbero presenti con tecniche risolutive più semplici, per le quali non c'è alcun bisogno del ruolo in questione. Probabilmente gli autori della Specifica avevano in mente qualcos'altro, ma non ci è dato di conoscerlo.

Ritornando alla descrizione del protocollo, in Figura 39 viene mostrato lo State Diagram della transazione, in funzione dei messaggi scambiati tra l'ATCoordinator e il singolo ATParticipant.

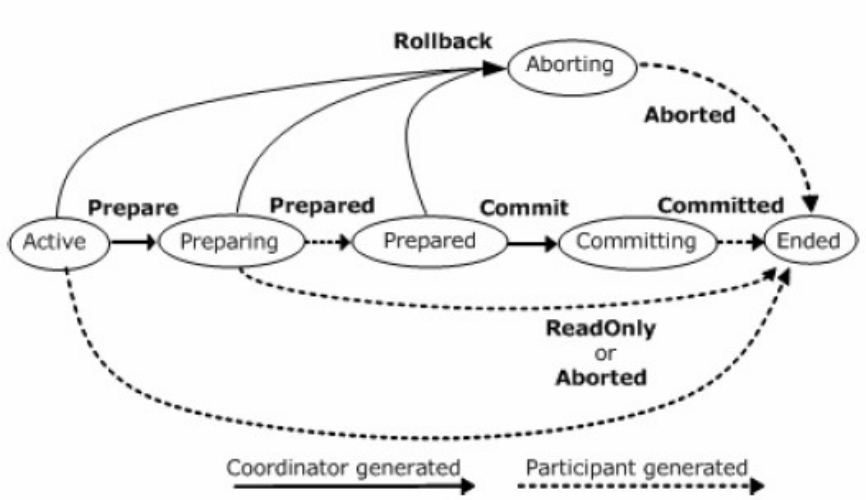

Figura 39 Protocollo 2PC
- tratto da Riferimenti[6 p7]-

Il significato dei messaggi rappresentati, è specificato da WS-AT come di seguito esposto.

Dopo che un partecipante ha ricevuto:

- *prepare*: sa che entra nella prima fase del protocollo e che dovrà votare per il risultato della transazione; se il partecipante non conosce la transazione, deve votare per l'Abort. Se il partecipante ha già votato, deve rispondere con lo stesso voto;

Per di più, se la consistenza tra la "cache" e la base-dati fosse assicurata da opportuni Gestori Software, come quasi sempre accade, il partecipante potrebbe risiedere nello strato intermedio, e accedere direttamente alla "cache".

- *rollback*: sa che deve annullare e "dimenticare" la transazione; il coordinatore può inviare questo messaggio in entrambe le fasi del protocollo[83], e una volta inviatolo, può "dimenticare" la transazione.
- *commit*: sa che deve effettuare il Commit della transazione; questo messaggio può essere ricevuto solo dopo che il partecipante ha votato per il Commit; se il partecipante non conosce la transazione, deve rispondere con un *committed*.

Dopo che il coordinatore ha ricevuto:

- *prepared*: sa che il partecipante ha votato per il Commit della transazione;
- *readOnly*: sa che il partecipante ha votato per il Commit della transazione, e che la ha "dimenticata", non essendo interessato al risultato finale;
- *aborted*: sa che il partecipante ha "abortito" e "dimenticato" la transazione;
- *committed*: sa che il partecipante ha effettuato il Commit della transazione; il partecipante può essere "dimenticato";
- *replay*: può assumere che il partecipante abbia effettuato il recovery dopo un fallimento, e dovrà inviargli l'ultimo messaggio del protocollo.

[83] Si fa riferimento alle fasi di Prepare e di Commit.

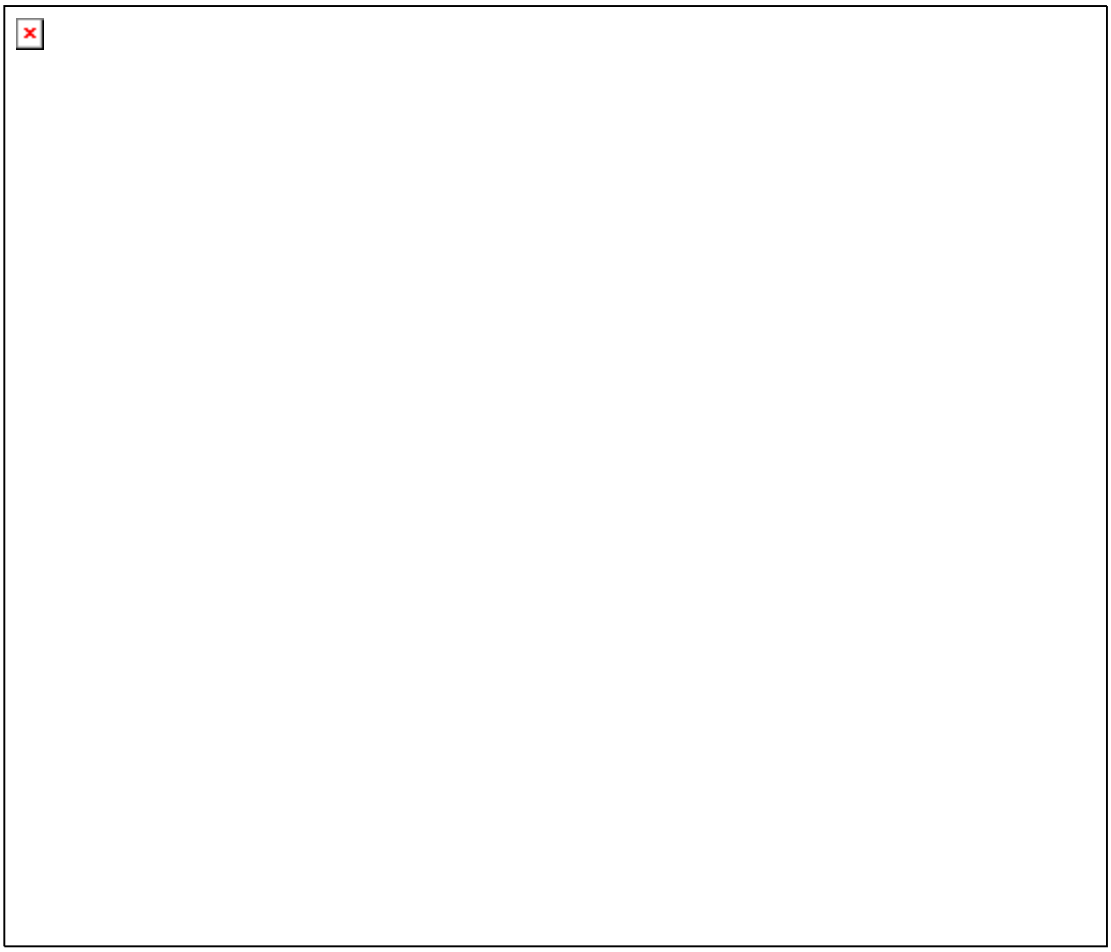

Figura 40 Semplici interazioni conformi ai protocolli Completition e 2PC

La Specifica, pur non definendo il comportamento interno dei partecipanti di tipo AT, definisce implicitamente il livello di Isolamento che essi devono supportare e garantire. Sottolinea infatti che le azioni intraprese prima del Commit devono essere solo dei tentativi (non visibili alle altre Attività), e solo in caso di Commit, possono diventare visibili alle altre transazioni; in caso di Abort, tali tentativi devono apparire come se non fossero mai stati eseguiti.

2.2.2.3 Failure Handling

Ai messaggi di errore definiti in WS-Coordination, WS-AT aggiunge un ulteriore messaggio:

- *inconsistentInternalState*: viene inviato al coordinatore da un partecipante che non sia in grado di mantenere le sue "obbligazioni". Indica un errore globale che compromette la Consistenza[84], e rappresenta una condizione non recuperabile.

Inoltre, le considerazioni fatte in WS-Coordination, in proposito alla rispedizione dei messaggi, continuano a valere per tutti i messaggi definiti in WS-AtomicTransaction.

2.2.2.4 Recovery Handling

Similmente a quanto abbiamo già visto per BTP, WS-AT non definisce particolari messaggi per il recovery (a parte il messaggio di *replay*), ma ammette la rispedizione dei messaggi. A differenza di BTP, invece, non specifica chiaramente quali siano le informazioni che debbano essere mantenute persistentemente dal coordinatore e dai partecipanti. Tali informazioni possono tuttavia essere estrapolate dalle tavole di stato riportate alla fine della Specifica, come di seguito esposto.

Il coordinatore dovrà ricordare:

- informazioni che identificano ogni partecipante arruolato, fino a quando questi non invii un messaggio di *readOnly*, di *committed* o di *aborted*.
- la decisione di ogni partecipante ad essere "prepared", fino a quando questi non invii un messaggio di *committed* o di *aborted*.
- la decisione finale di Commit, prima di avviare la seconda fase del 2PC, e fino a conclusione della stessa.

[84] Nel senso di Consistenza Globale, o meglio di Atomicità.

Un partecipante dovrà ricordare:

- La propria decisione ad essere "prepared", e informazioni che identificano il coordinatore, fino alla comunicazione della decisione globale.

WS-AT precisa inoltre che, affinché il coordinatore possa prendere una decisione finale di Commit, questa deve prima essere memorizzata, altrimenti dovrà decidere per l'Abort. Analogo discorso vale per il partecipante, che prima di inviare un *prepared*, deve memorizzare localmente la condizione di "pronto", altrimenti dovrà inviare un *aborted*.

2.2.2.5 Binding

Le tre Specifiche componenti la famiglia di WS-Transactions si basano su WS-Addressing[85], pertanto il "binding" è implicitamente fornito per SOAP. Relativamente all'uso di WS-Addressing, WS-AT classifica i messaggi che definisce, e quelli definiti in WS-Coordination, come segue:

- messaggi di Request: *createCoordinationContext* e *register*;
- messaggi di Reply: *createCoordinationContextResponse* e *registerResponse*;
- messaggi di Notification: questi sono suddivisi ulteriormente in:
    - Terminali: *committed*, *aborted* e *readOnly*;
    - Non terminali: *commit*, *rollback*, *prepare*, *prepared* e *replay*.

I messaggi di "richiesta" e di "risposta" devono essere trattati secondo il modello "Request-Reply", mentre i messaggi di "notifica" secondo il modello "one-way", essendo entrambi i modelli definiti in WS-Addressing.

I messaggi di "richiesta" devono contenere:

- MessageID header[86]: per correlare la successiva risposta al messaggio di "richiesta";

[85] Riferimenti[18].
[86] Per "header", si intende un particolare elemento strutturato in XML, definito da WS-Addressing, e che dovrà essere contenuto nell'Header dei messaggi SOAP.

- ReplyTo header: specifica l'indirizzo a cui dovrà essere restituita una risposta.

I messaggi di "risposta" devono contenere:

- RelatesTo header: specificando il MessageID del corrispondente messaggio di richiesta.

I messaggi di "notifica" non terminale devono contenere:

- ReplyTo header: specifica l'indirizzo a cui dovrà essere restituito il corrispondente messaggio di riscontro.

Quest'ultimo header non dovrebbe essere contenuto, invece, nei messaggi di "notifica" terminali.

WS-AT precisa che i messaggi di "notifica" possono essere inviati, dal coordinatore e dai partecipanti, utilizzando gli EndpointReferences[87] ottenuti durante lo scambio nella fase iniziale di registrazione. Se un messaggio di "notifica" contiene un header "ReplyTo", questo può essere utilizzato dal ricevente, ad esempio, nel caso in cui abbia dimenticato una transazione completata e debba rispondere ad un messaggio di protocollo rispedito.

Precisa inoltre che, un'eventuale perdita **permanente** della connettività tra partecipante e coordinatore in uno stato dubbioso, può determinare la corruzione dei dati.

Infine, precisa che ogni messaggio deve essere consegnato utilizzando la connessione attivata dal mittente del messaggio stesso[88], e che ogni EndpointReference deve contenere un indirizzo fisico, e mai un "anonymous endpoint" (definito in WS-Addressing).

[87] Un EndpointReference è un elemento strutturato XML definito da WS-Addressing. Esso è composto da diversi altri campi, ma ai fini del nostro discorso può essere interpretato come un semplice indirizzo definito al "livello dei WebServices".

[88] La Specifica non risulta chiara in proposito. Assumendo che per "connessione" intenda lo scambio degli indirizzi nella fase di registrazione (come può dedursi dalla nota 74 a pag 139), probabilmente vuole precisare semplicemente che i messaggi debbano essere inviati agli indirizzi scambiati esplicitamente in fase di registrazione. In altri termini, si richiede che il ricevente verifichi che l'indirizzo del mittente sia quello comunicato all'atto dell'inizializzazione della "connessione".

2.2.3 WS-BusinessActivity (WS-BA)

Estende il modello definito da WS-Coordination con il modello "Business Activity" (BA), definendo il corrispondente tipo di coordinatore (che chiameremo BACoordinator), e i relativi protocolli. Gli autori della Specifica considerano il modello BA particolarmente indicato per quelle Attività che gestiscono le eccezioni[89] mediante logica business. Tali eccezioni vengono dette dunque "business exceptions", e le attività "Business Activities". La Specifica caratterizza un'Attività Business assegnandole le seguenti proprietà:

- Può consumare molte risorse e per un lungo periodo di tempo.
- Può coinvolgere un numero significativo di transazioni atomiche.
- Il risultato dei singoli tasks costituenti un'Attività Business può essere "visto" prima del completamento dell'Attività stessa, e spesso può avere delle conseguenze al di fuori del sistema informatico.
- Le elaborazioni possono essere molto lunghe, e un servizio partecipante all'Attività, prima di poter rispondere ad una richiesta, potrebbe impiegare molto tempo. Ad esempio, un'approvazione umana, un assemblaggio, una consegna, potrebbero dover essere intrapresi prima dell'invio di una risposta.
- Nel caso di *business exceptions* per le quali un'Attività debba essere logicamente sottoposta ad *undò*, è necessario spesso applicare un approccio compensativo, per esempio, nella forma di un task di compensazione, che inverta gli effetti di un business task già completato.

Inoltre assume che i partecipanti in un'Attività Business possano risiedere in domini di fiducia indipendenti, e che, in tal caso, tutte le relazioni di fiducia interdominio siano stabilite esplicitamente.

[89] Per "eccezioni" si deve intendere "situazioni eccezionali".

Sulla base delle caratteristiche dette, gli autori hanno realizzato il modello di coordinazione per Attività Business oggetto del presente capitolo, cui attribuiscono le seguenti proprietà:

1. Supporta "Attività innestate", in un'organizzazione ad albero di tipo "padre-figli" con un numero di livelli di innesto arbitrario. Ogni livello ha il proprio Scope[90], e permette di selezionare quali figli debbano partecipare al risultato comune.
2. Si basa sul concetto di Compensazione, e consente ai partecipanti di effettuare operazioni di "tentativo" come ordinaria amministrazione. Il risultato di tali "tentativi" può essere visibile prima che l'Attività sia completata, e può richiedere compensazioni in caso di errori (*business exceptions*).
3. A differenza delle transazioni atomiche, consente che la lista dei partecipanti possa variare dinamicamente, e permette ad un partecipante di lasciare un'Attività Business in qualsiasi momento.[91]
4. Permette ad un partecipante di anticipare le proprie decisioni senza aspettare sollecitazioni.[92]

WS-AT precisa inoltre che, a differenza di una Atomic Transaction, le azioni richieste in una Business Activity sono applicate immediatamente, e sono subito permanenti, mentre un'azione compensatrice può essere invocata in caso di *business exceptions*.

[90] Per "Scope" si intende il contesto di esecuzione, ovvero il contesto di un'Attività Business ottenuto dalla condivisione del CoordinationContext.
[91] Dagli State Diagrams che saranno riportati in Figura 43 e in Figura 44, il lettore potrà notare che, in realtà, il messaggio di *exit* permette ad un partecipante di ritirarsi da un'Attività Business solo quando sia ancora in corso la fase di Active (e non "in qualsiasi momento")
[92] Per "decisioni anticipante" possono intendersi quella di Exit, e quella di Completed, quest'ultima solo nella variante BAwPC.

2.2.3.1 Modello Concettuale

Un possibile modello concettuale, estrapolato da WS-BusinessActivity, è rappresentato[93] nella figura seguente:

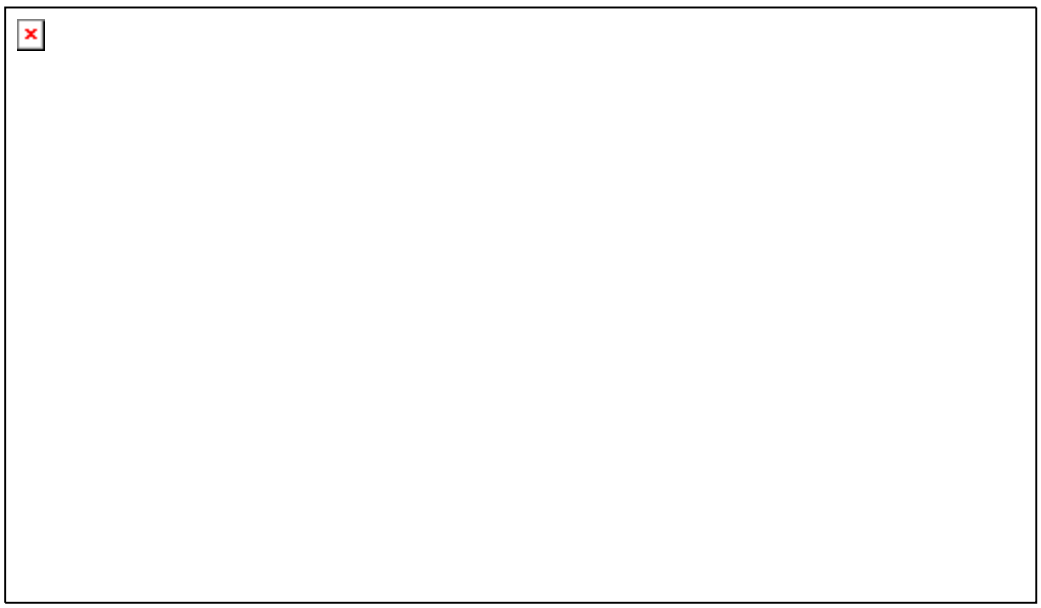

Figura 41 Modello Concettuale di WS-BusinessActivity

Come mostrato, il modello BA definisce due tipi di coordinatore:

- **AtomicOutcomeCoordinator**: dirige tutti i partecipanti a chiudere o compensare;
- **MixedOutcomeCoordinator**: può dirigere indipendentemente ogni partecipante a chiudere o compensare;

e due protocolli, entrambi utilizzabili da ciascun tipo di coordinatore:

- BusinessAgreementWithParticipantCompletition (BAwPC);
- BusinessAgreementWithCoordinatorCompletition (BAwCC).

I due protocolli, in realtà, possono più propriamente essere considerati come un unico protocollo, "BusinessAgreement", a cui può essere applicata una variante. L'unica

[93] Anche in tal caso, per semplicità, non sono mostrate le relazioni per un Interposed, essendo le stesse dell'ApplicationService.

differenza consiste infatti nell'attore che avvia l'esecuzione: nel protocollo BAwPC, il partecipante sa quando avviare l'esecuzione del lavoro di cui è responsabile, e si limita semplicemente a comunicarne il completamento al coordinatore; nel protocollo BAwCC, è invece il coordinatore ad indicare al partecipante quando avviare l'esecuzione. Il lettore troverà più chiare le affermazioni fatte in seguito alla comprensione dei due protocolli in questione.

Come WS-AT, anche WS-BA precisa che l'attributo Expired del CoordinationContext, se utilizzato, indica il tempo entro il quale l'Attività Business dovrebbe terminare.

Precisa inoltre che, grazie alla possibilità di estensione di WS-Coordination, è possibile, oltre che definire nuovi protocolli di coordinazione, definire anche la logica del comportamento di coordinazione. Ad esempio è possibile definire un tipo di coordinatore in cui la decisione venga presa a maggioranza.

In WS-BA non vengono aggiunte ulteriori informazioni relative ad un coordinatore Interposed e pertanto, è lecito supporre che valga, senza modifiche, quanto specificato in WS-Coordination[94].

Si noti infine che per una Business Activity non è stato definito nessun "Protocollo di Completamento".

[94] In maniera analoga a quanto osservato per WS-AT, un coordinatore subordinato di un coordinatore di tipo BA, può essere solo un coordinatore di tipo BA. Pertanto, un eventuale partecipante che funga anche da coordinatore per una Transazione Atomica, va inteso come un semplice partecipante, e tale transazione dovrà essere considerata come un'Attività autonoma, e non "innestata" (si noti che tra le caratteristiche assegnate dalla Specifica ad un'Attività Business, e riportate a pag 158, si fa esplicitamente riferimento, al secondo punto, alla possibilità che una Business Activity coinvolga molteplici Transazioni Atomiche).

2.2.3.2 Protocolli

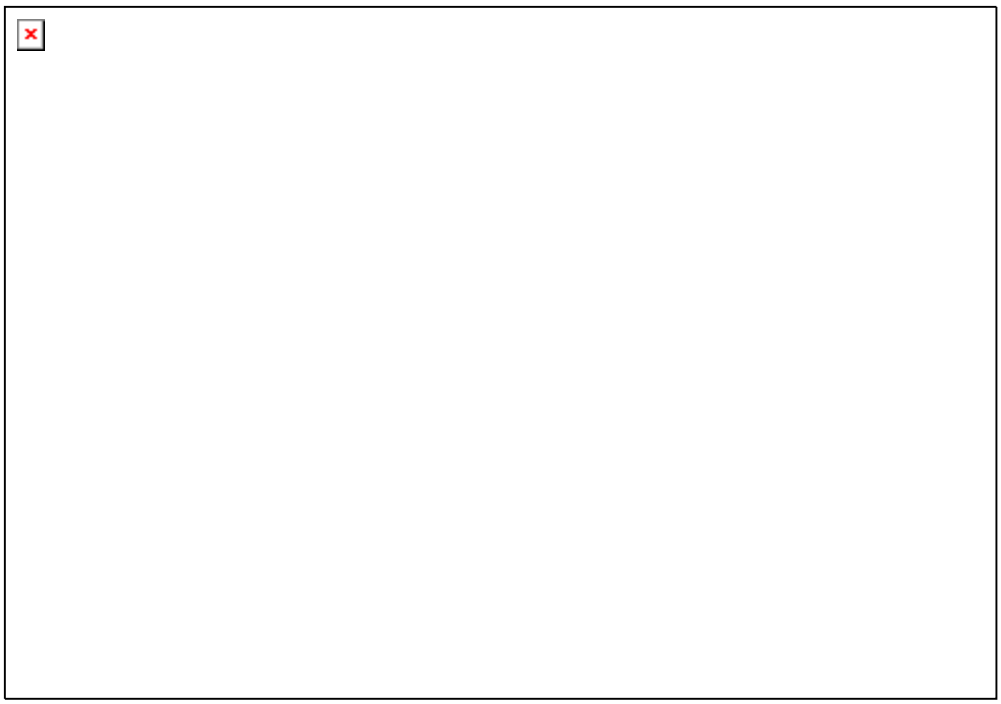

Figura 42 Modello delle Interfacce di WS-BusinessActivity

2.2.3.2.1 BusinessAgreementWithParticipantCompletition

Questo protocollo di coordinazione prevede che il partecipante comunichi al coordinatore quando abbia terminato il lavoro di cui è responsabile per l'Attività Business.

Nella figura seguente viene mostrato il comportamento astratto del protocollo. In essa, gli stati rappresentano la conoscenza che il coordinatore e un partecipante hanno della loro relazione. Tale conoscenza, precisa WS-BA, può differire solo per tempi transitori.[95]

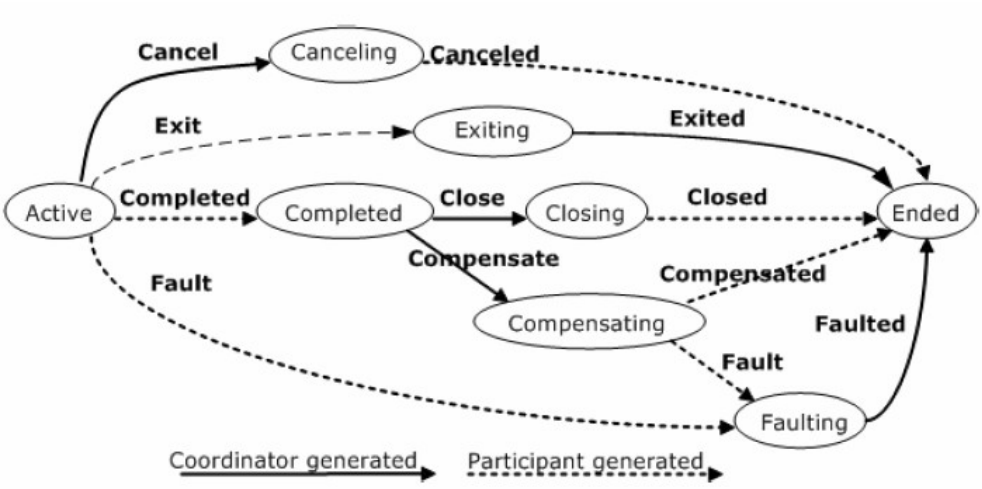

Figura 43 Protocollo BusinessAgreement con "completamento" del partecipante
- tratto da Riferimenti[7 p8] -

Il significato dei messaggi rappresentati è specificato da WS-BA come di seguito esposto.

Dopo che il coordinatore ha ricevuto:

- *completed*: sa che il partecipante ha completato il lavoro di cui è responsabile, e che aspetta una decisione finale (Close o Compensate);
- *fault*: sa che il partecipante ha fallito nell'eseguire il suo lavoro o nella compensazione, e che aspetta un messaggio di *faulted*; quest'ultimo gli assicurerà che il coordinatore è consapevole della situazione. Il messaggio di *fault* può indicare anche la causa del fallimento;

[95] Si veda "2.2.3.4 Recovery Handling".

- *compensated*: sa che il partecipante ha ricevuto e salvato la richiesta di compensazione;
- *closed*: sa che il partecipante ha concluso con successo;
- *canceled*: sa che il partecipante ha concluso con successo la richiesta di cancellazione;
- *exit*: sa che il partecipante non vuole più partecipare all'Attività Business, e che aspetta una notifica di *exited*;

Dopo che il partecipante ha ricevuto:

- *close*: sa che la transazione è stata completata con successo, e invierà una notifica di *closed* al coordinatore;
- *cancel*: sa che il lavoro eventualmente già fatto deve essere disfatto, e invierà una notifica di *canceled* al coordinatore;
- *compensate*: sa che il lavoro fatto deve essere compensato, e invierà una notifica di *compensated* al coordinatore;
- *faulted*: sa che il coordinatore è consapevole della situazione anomala precedentemente comunicatagli. Nessuna ulteriore azione è richiesta;
- *exited*: sa che il coordinatore ha ricevuto il messaggio di *exit* precedentemente inviatogli.

Sia il coordinatore che il partecipante accettano:

- *getStatus*: richiede lo stato corrente del ricevente. In risposta il ricevente invia un messaggio di *status*;
- *status*: indica lo stato corrente del mittente.

2.2.3.2.2 BusinessAgreementWithCoordinatorCompletition

A differenza del protocollo precedente, ora il partecipante attende che il coordinatore gli indichi di avviare il lavoro per l'Attività Business. Questa è l'unica differenza, e si traduce in un solo messaggio aggiuntivo:

- *complete*: dopo aver ricevuto questa notifica, il partecipante sa che non riceverà ulteriori richieste di lavoro da effettuare all'interno della stessa Attività Business. Potrà pertanto avviare e completare il lavoro applicativo, e trasmettere la notifica di *completed* al coordinatore.

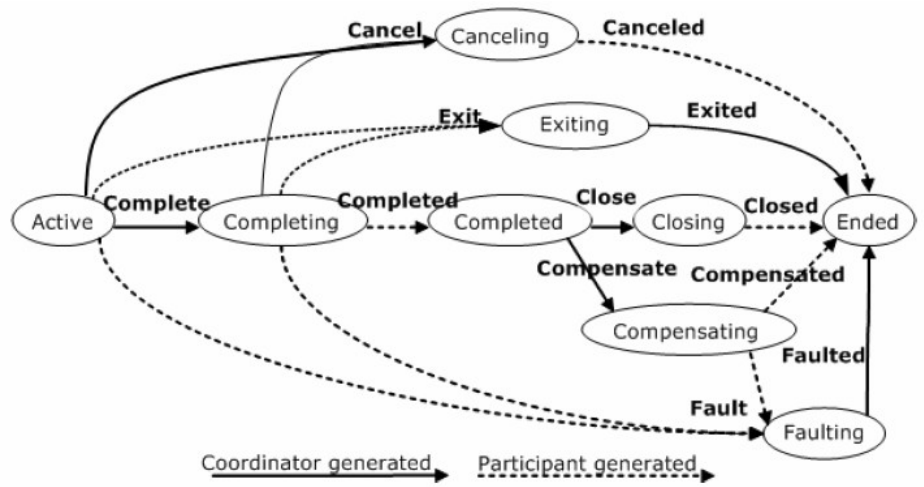

Figura 44 Protocollo BusinessAgreement con "completamento" del coordinatore - tratto da Riferimenti[7 p9] -

Per quanto riguarda questo specifico protocollo, risulta maggiormente evidente la mancanza di un protocollo di completamento: ad esempio, come fa il coordinatore a sapere quando inviare il messaggio di *complete* a tutti i partecipanti BAwCC-Participant?

In Figura 45 viene riportato un semplice Sequence Diagram che mostra delle possibili interazioni tra un coordinatore di tipo MixedOutcameCoordinator e due partecipanti, rispettivamente di tipo BAwCC-Participant e BAwPC-Participant. Ivi si è assunto che il *richiedente* coincida con il coordinatore.

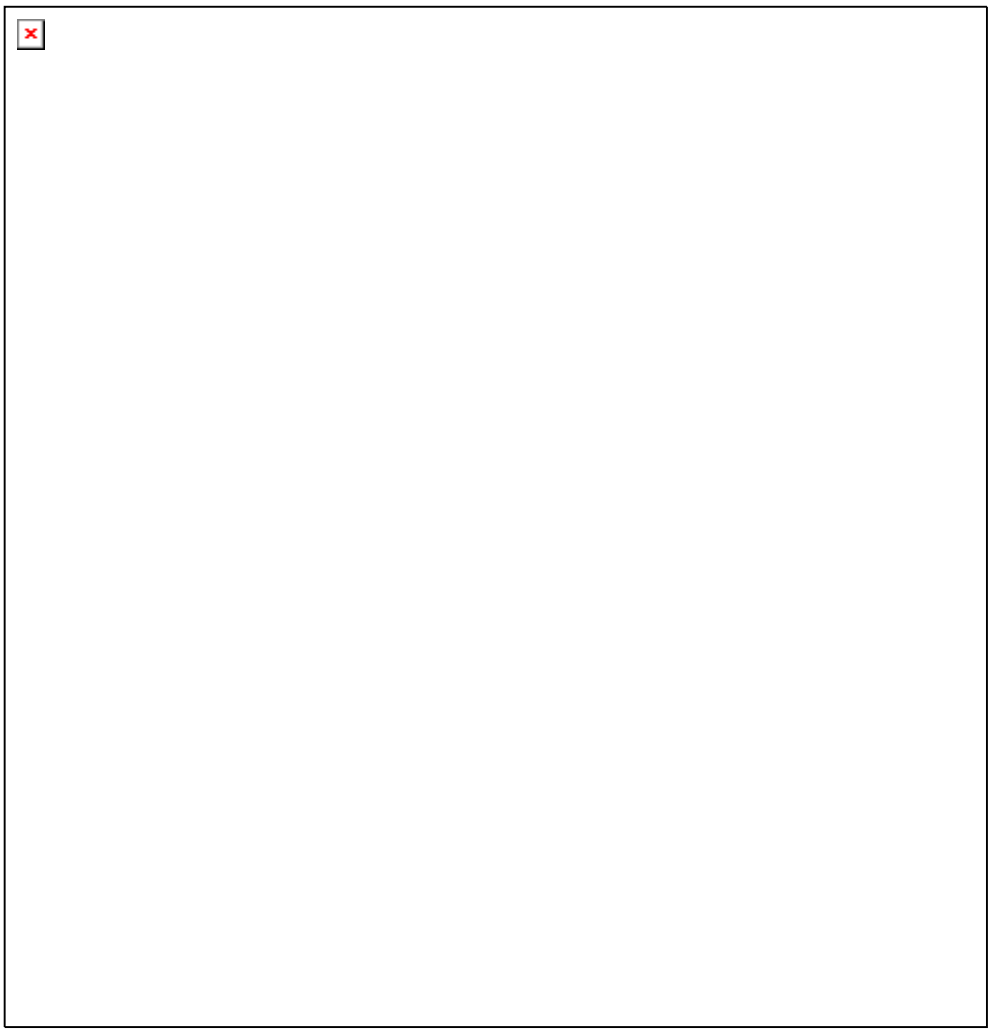

Figura 45 Semplici interazioni per il protocollo “Business Agreement”

2.2.3.3 Failure Handling

WS-BusinessActivity non aggiunge ulteriori messaggi di errore a quelli già definiti in WS-Coordination, e continuano a valere le stesse considerazioni già fatte sulla rispedizione dei messaggi.

WS-BA precisa, nel caso in cui il coordinatore riceva un messaggio da un partecipante che si riferisca ad uno stato precedente, e non allo stato corrente, che è suo compito tornare indietro a detto stato, per guidare il partecipante allo stato attuale.

2.2.3.4 Recovery Handling

WS-BA non prevede particolari messaggi per il recovery, ma assume che i messaggi siano rispediti quando necessario, e richiede che tutte le transizioni tra stati siano precedute da una memorizzazione permanente dello stato originale, incluso lo stato applicativo, e dei "metadata" utilizzati per la coordinazione.

Sottolinea che nei protocolli precedentemente descritti tutti i messaggi sono seguiti da un messaggio di riscontro, per assicurare che sia il partecipante che il coordinatore abbiano una "vista comune" del progresso raggiunto nell'esecuzione.

2.2.3.5 Binding

Come per le altre due Specifiche della famiglia, anche WS-BA fornisce gli XML Schema e la descrizione WSDL dei servizi.

Stranamente, quanto specificato in WS-AT circa l'uso di WS-Addressing, non viene ripreso e specializzato per i messaggi definiti da WS-BA. Ciò nonostante, viene precisato che ogni messaggio deve essere considerato singolarmente, poiché, per transazioni di lunga durata, i meccanismi di richiesta/risposta del protocollo di Trasporto, e l'utilizzo di "timeout" per le rispedizioni, non sono generalmente sufficienti per raggiungere un risultato consistente. Pertanto, può dedursi che siano ammessi solo i modelli "one-way" e "request/reply" asincrono, ma non il modello "request/reply" sincrono.

2.2.4 Specifiche di Supporto

Nella descrizione di WS-Transactions abbiamo già incontrato WS-Addressing e WS-ReliableMessaging, e nel paragrafo 2.2.5, descrivendo il modello per la Sicurezza definito da WS-Tx, incontreremo molte altre Specifiche. Esse, qui indicate genericamente come Specifiche di Supporto, costituiscono i "mattoni" di una più ampia architettura per WebServices: quella immaginata da Microsoft e IBM.

In questo paragrafo ne daremo una breve panoramica, non tanto perché mancano soluzioni alternative e/o complementari[96], bensì perchè risulta una delle architetture più complete nell'ambito in questione. Potremo così capire tutte le varie problematiche che si presentano nella pratica utilizzando tale tecnologia.

Vedremo come queste Specifiche propongano soluzioni generali a problemi particolari, molti dei quali coincidono proprio con i requisiti che abbiamo ricavato per le Transazioni Business nel capitolo "1.4 Analisi e specifica dei Requisiti". Tali problemi potrebbero essere risolti con meccanismi proprietari, magari anche più semplici ed efficienti di quelli proposti dalle Specifiche, ma ne sarebbe penalizzata la stessa caratteristica di interoperabilità che è fondamenta della tecnologia dei Web Services.

Ad esempio, definire un modello per la Sicurezza in ques'ambito, non significa semplicemente definire un particolare modello per realizzare un dominio sicuro, bensì significa garantire contemporaneamente l'interoperabilità interdominio, che detto in altri termini, significa lasciare ai singoli servizi la possibilità di scelta su quale modello per la Sicurezza adottare, in funzione delle particolari relazioni business di volta in volta stabilite. Sappiamo infatti che i diversi modelli per la Sicurezza oggi disponibili hanno delle caratteristiche peculiari, che li rendono più o meno idonei a seconda delle circostanze. In particolare, ogni modello ha delle caratteristiche proprie di efficienza e di sicurezza, caratteristiche, queste, molto spesso in competizione.

Le Specifiche di Supporto definiscono dei modelli capaci di assicurare massima libertà di scelta, e dunque massima interoperabilità, proprio come si richiede in un ambiente interdominio quale è quello dei servizi Web. Esse nascono per detta

[96] Vedi architettura secondo Sun-Oracle nel cap2.3.

tecnologia, e come tali si basano esplicitamente sugli standard da questa adottati: SOAP e XML. Pertanto, molto spesso, richiamano a loro volta altre Specifiche definite nello stesso ambito.

Nel seguito, data la vastità degli argomenti da esse trattati, le descriveremo brevemente limitandoci a fornire solo qualche concetto basilare, col semplice intento di rendere meglio comprensibile come siano integrabili nel framework definito da WS-Transactions.

2.2.4.1 Web Services secondo Microsoft-IBM

Come ogni architettura di comunicazione, anche la "visione" Microsoft-IBM può essere riassunta in uno stack di livelli:

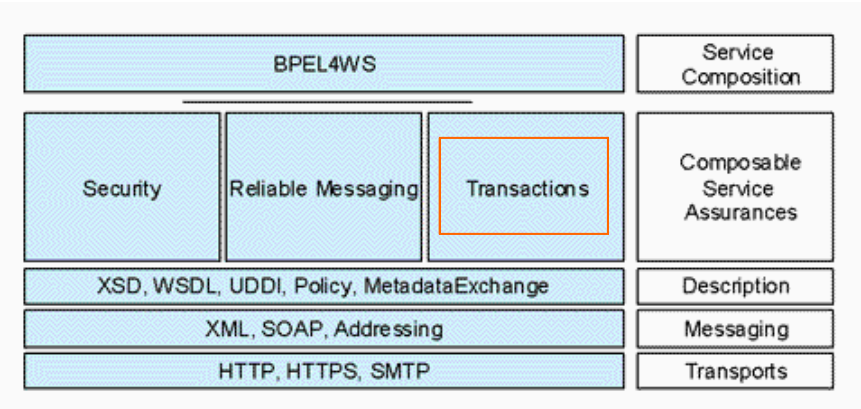

Figura 46 Architettura Microsoft-IBM per Web Services
- tratta da Riferimenti[33] -

Il livello più basso, quello del Trasporto, definisce i protocolli di base su cui tutti gli altri si fonderanno. In particolare troviamo il noto protocollo **HTTP**, assieme al meno noto **HTTPS**, il quale altro non è che un estensione del primo con supporto per la Sicurezza, e il protocollo **SMTP** comunemente utilizzato per scambiare messaggi di posta elettronica.

Il livello successivo è quello dei Messaggi. Ivi troviamo gli standard XML, SOAP e Addressing. **XML** è ormai famoso quanto HTTP, e non necessita di particolari

spiegazioni. L'unica cosa che qui vale la pena sottolineare, è la flessibilità offerta da tale linguaggio: definire uno stack di protocolli basati su XML non impone una rigida struttura precostruita, come avverrebbe con un approccio tradizionale. Nell'approccio tradizionale, infatti, molto spesso la semantica dei messaggi è posizionale, ovvero il ricevente interpreta i dati contenuti in un messaggio, per un dato protocollo, in funzione della particolare posizione che in esso occupano. In XML, grazie alla possibilità di strutturazione mediante "marcatori" (noti anche come "tag"), invece, ogni elemento di un messaggio è automaticamente definito dal marcatore associato, qualunque sia la posizione che in esso occupa (tranne se diversamente specificato). Pertanto, è sempre possibile estendere un protocollo già definito per conferirgli nuove funzionalità, semplicemente definendo nuovi "marcatori". Questa grande flessibilità si traduce non solo nella possibilità di estensione detta, ma anche in quella di inserire più elementi, appartenenti a protocolli diversi, in uno stesso messaggio. Il tutto può avvenire in qualsiasi momento, e senza penalizzare la compatibilità con i servizi che già utilizzano i messaggi "non estesi", permettendo così un forte disaccoppiamento tra livelli adiacenti, e un'ottima convivenza di protocolli all'interno dello stesso livello. In una sola parola, XML permette la Scalabilità dei protocolli. Tutte le Specifiche che accenneremo nel seguito, basandosi su XML, godono di questa importante proprietà.

SOAP definisce quello che in gergo viene chiamato l' "inviluppo" dei messaggi XML. In pratica, definisce come un generico messaggio XML possa essere scambiato tra servizi Web, specificando la struttura cui tale messaggio deve sottostare.

Addressing definisce un meccanismo generale per gestire gli indirizzi, consentendo così il disaccoppiamento tra gli indirizzi definiti al livello del Trasporto e quelli definiti al "livello dei servizi Web". In particolare, inserendo gli indirizzi direttamente nei messaggi scambiati, permette di indirizzare istanze diverse di uno stesso servizio, e permette di specificare se un'eventuale risposta debba essere inviata ad un indirizzo differente da quello del mittente. Supporta tre modelli per lo scambio dei messaggi: il modello "one-way", per il quale un messaggio viene inviato senza richiedere una risposta; il modello "request/reply" sincrono, in cui il richiedente si blocca per attendere la risposta; e il modello "request/reply" asincrono, in cui il richiedente attende, ma continuando nella sua esecuzione.

Sopra il livello dei messaggi, troviamo il livello della Descrizione. Questo è fondamentale ai fini dell'interoperabilità. In esso troviamo infatti lo standard **WSDL**, con cui è possibile descrivere l'interfaccia di un servizio Web, ovvero le modalità con cui è possibile utilizzare il servizio. Strettamente associato al WSDL troviamo lo standard XSD (**XML Schema**), che permette di definire i tipi di dati che possono essere scambiati all'interno dei messaggi con detta interfaccia. WSDL e XSD, insieme, definiscono dunque i meccanismi con i quali è possibile accedere ad una certa funzionalità, ed interpretare il risultato restituito. Non definiscono tuttavia come un servizio possa venire a conoscenza dell'interfaccia di un altro servizio. Evidentemente questo può avvenire con sistemi di scambio ordinari quali siti web, giornali, mezzi pubblicitari, etc. oppure può avvenire consultando un "registro **UDDI**".

Quest'ultimo è un particolare servizio Web, con un'interfaccia nota, che opera da "servizio informativo": in qualche maniera è analogo al servizio cartaceo fornito dall'elenco telefonico. UDDI, oltre a definire l'interfaccia del servizio per richiedere e pubblicare informazioni, definisce come interpretare le informazioni stesse. Un "registro UDDI" viene utilizzato ordinariamente in due modi: fornendo l'indirizzo del servizio Web di interesse (corrisponde al nome dell'utente, nell'analogia con il servizio telefonico), per ottenerne la descrizione WSDL; fornendo un'interfaccia nota, per ottenere gli indirizzi dei servizi che la supportano. Se è conosciuto l'indirizzo, è possibile utilizzare alternativamente i protocolli specificati in **MetadataExchange,** per ottenere le stesse informazioni direttamente dal servizio Web di interesse. Entrambe le tecniche, noto l'indirizzo, consentono dunque di ottenere l'interfaccia di accesso ad un servizio Web qualsiasi. UDDI permette anche il viceversa.

Per garantire una maggiore capacità di interoperabilità, sarebbe tuttavia necessario qualche meccanismo aggiuntivo che consenta di selezionare "dinamicamente" i servizi Web, non in base ad un indirizzo o ad un'interfaccia nota a priori, bensì in base al tipo di servizio stesso che essi offrono. Sarebbe necessario, cioè, un meccanismo più simile a quello delle "Pagine gialle" piuttosto che a quello dell'elenco telefonico. Quest'ulteriore meccanismo è definito dalla famiglia di Specifiche **Policy**.

Essa comprende tre Specifiche distinte:

- **WS-Policy**: definisce la sintassi per specificare la "politica" di un servizio, in termini di capacità fornite e richieste all'utente.

- **WS-PolicyAttachment**: definisce come pubblicizzare la policy propria di un servizio Web all'interno della sua descrizione WSDL.
- **WS-PolicyAssertions**: definisce una serie di "policy assertions" standard con cui etichettare i servizi secondo WS-Policy. In pratica una sorta di tassonomia predefinita.

Si noti che, in precedenza, abbiamo parlato di "selezione dinamica" ponendo l'avverbio "dinamicamente" tra virgolette. Infatti, se è vero che è possibile selezionare un servizio Web che possieda determinate caratteristiche in base alla sua "policy", e dunque è possibile conoscere l'interfaccia con cui accedervi, è anche vero che non è stato definito un meccanismo (e forse mai lo sarà) per venire a conoscenza del significato dei dati scambiati mediante tale interfaccia. Dunque il concetto di dinamismo, da un punto di vista di programmazione, solo in alcuni casi può essere inteso come la possibilità di selezione a "run-time". Ad esempio, ciò è possibile quando il significato dei messaggi non debba essere interpretato dall'elaboratore, ma da un utente umano; oppure quando il significato dei dati scambiati sia noto a priori, ad esempio perché rispetta uno specifico standard, come accade nel caso di diversi modelli per la Sicurezza.

Il livello superiore nello stack Microsoft-IBM corrisponde al livello che "garantisce la componibilità dei servizi". In esso troviamo le famiglia di Specifiche Security, Reliable Messaging, e Transactions.

La prima, **Security**, comprende tutta una serie di Specifiche per garantire la Sicurezza. In particolare ad essa appartengono: WS-Security, WS-Trust, WS-SecureConversation e WS-Federation.

WS-Security nasce per garantire l'integrità e la confidenzialità di un messaggio anche solo rispetto ad alcuni campi, piuttosto che rispetto all'intero messaggio. Questo non è possibile a livello di HTTPS, poiché tale livello non conosce nulla riguardo XML, SOAP e messaggi strutturati. Conseguentemente WS-Security permette che la confidenzialità e l'integrità siano mantenute facilmente anche se il messaggio debba essere processato, solo in parte, da un intermediario.

WS-Trust definisce invece un meccanismo per garantire l'autenticità di un messaggio. Tale meccanismo si basa sulla presenza di un servizio di *fiducia* che funge da garante per tutte le parti coinvolte nella comunicazione: il concetto è analogo a quello con cui

un ente statale, quale è l'ufficio anagrafico di un Comune, "garantisce" rilasciando carte d'identità.

WS-SecureConversation estende WS-Security per permettere confidenzialità e integrità dei messaggi in maniera efficiente, anche nel caso di comunicazioni di lunga durata.

Infine, **WS-Federation** concede a più servizi Web, federati in un unico gruppo, la possibilità di collaborare efficientemente per le operazioni richieste da un utente che sia registrato e autenticato presso uno solo di loro. In pratica, l'utente registrato con uno di questi servizi, può utilizzare le stesse informazioni di login anche per accedere agli altri servizi, oppure, in caso di problemi di privacy, può utilizzare uno pseudonimo fornito dal servizio che ha effettuato la registrazione. Quest'ultimo, dunque, fungerà da garante nei confronti degli altri servizi. Anche quando sia utilizzato lo pseudonimo, gli altri servizi della federazione potranno convenientemente personalizzare[97] l'esecuzione delle funzionalità offerte in base alle preferenze dell'utente in questione, pur senza violare i diritti di privacy.

La seconda famiglia, **Reliable Messaging**, definisce una serie di meccanismi per rendere affidabile un mezzo di trasporto inaffidabile, quale è, ad esempio, HTTP, HTTPS o SMTP. Tali meccanismi assicurano che i messaggi siano consegnati a destinazione, che siano interpretati nel giusto ordine, e che le rispedizioni siano riconosciute come tali, il tutto, anche in caso di caduta di connessione, congestioni, etc.

La terza famiglia, Transactions, corrisponde alle tre Specifiche che in questo stesso testo abbiamo indicato complessivamente come **WS-Transactions**. Come abbiamo visto, esse vogliono fornire supporto per Transazioni Distribuite, e dunque per una composizione affidabile dei servizi Web.

L'ultimo livello dello stack di Figura 46 può essere considerato al pari del livello Applicativo nel modello ISO/OSI. In questo livello vengono cioè definiti i protocolli che soddisfano particolari necessità applicative.

[97] La personalizzazione dinamica di un servizio, in funzione delle preferenze di ogni singolo utente, è oggi una pratica molto diffusa.

Nei seguenti sottoparagrafi approfondiremo leggermente quelle, tra le precedenti Specifiche menzionate, che giocano un ruolo particolare all'interno del modello per la Sicurezza definito da WS-Transactions.

2.2.4.2 WS-Security

Un *token* è definito come un insieme di *claims*, e un *claim* corrisponde ad una dichiarazione fatta da un'entità, ad esempio, riguardo la propria identità, il gruppo di appartenenza, privilegi e capacità posseduti, etc. Quando tali dichiarazioni siano anche "verificabili", si parla più propriamente di Credenziali o di *security token*. Le Credenziali sono comunemente utilizzate nella vita quotidiana in diverse forme. Esempi ne sono carte d'identità, passaporti, patenti, tessere di adesione, carte di credito, etc. Servono per dimostrare l'identità di chi le possiede, ma più spesso per dimostrare che si posseggono determinate capacità e privilegi. Ad esempio, una tessera può dimostrare che il possessore appartiene all'ordine dei medici italiani e che, di conseguenza, può accedere al parcheggio dell'ospedale appositamente riservato per essi.

WS-Security definisce una serie di meccanismi estensibili per:

- inviare un *security token* come parte di un messaggio;
- garantire l'integrità di un messaggio;
- garantire la confidenzialità di un messaggio;

vincolati ai seguenti requisiti:

a) supportare molteplici formati di *token*;
b) supportare molteplici tipologie di domini di fiducia;
c) supportare molteplici formati di *signature*;
d) supportare molteplici tecnologie di criptaggio;

In pratica definisce la sintassi affinché un messaggio SOAP possa contenere le informazioni che individuano un particolare modello per la Sicurezza, e dunque affinchè il ricevente del messaggio possa interpretarlo correttamente. Tale sintassi è stata definita generalizzando le informazioni relative ad autenticazione, signature, e

criptaggio, necessarie per supportare modelli[98] per la Sicurezza molto diffusi quali Kerberos e X.509 per l'autenticazione, "XML Encryption" e "XML Signature" rispettivamente per criptare e firmare messaggi XML, "XML Canonicalization" per preparare i messaggi XML ad essere criptati e firmati.

2.2.4.3 WS-Trust

Estende la specifica di WS-Security affinché un servizio Web possa richiedere e fornire *token* sicuri, e possa stabilire relazioni di fiducia.

Ad esempio, un servizio, affinchè possa essere utilizzato, richiede che l'utente abbia dei particolari requisiti. Questi sono pubblicati mediante *claims* all'interno della propria *policy*, secondo le convenzioni definite in WS-Policy. L'utente in possesso dei requisiti, può dichiarare di possederli mediante uno specifico *token*, che può ottenere (secondo WS-Trust) da un'autorità esterna (un altro servizio Web, denominato "Security Token Service" in Figura 47), di fiducia per entrambe le parti.

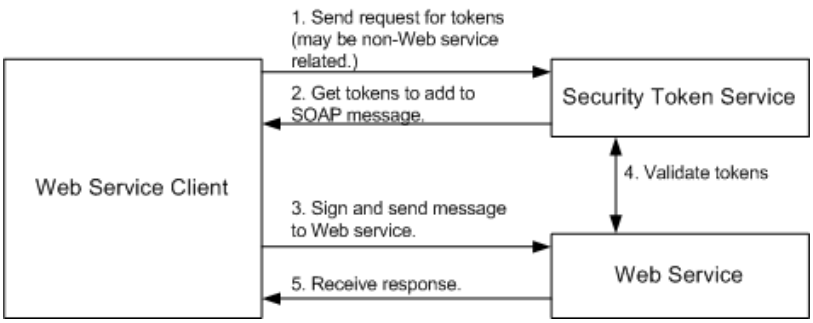

Figura 47 Flusso dei messaggi tipico per realizzare un dominio di fiducia
- tratta da Riferimenti[34] -

Tale autorità, in qualche maniera, fungerà da garante per l'utente, e permetterà che il servizio sia utilizzato correttamente. Ottenuto il *token*, l'utente può inserirlo nei messaggi (secondo WS-Security) indirizzati verso il servizio Web da utilizzare, il quale

[98] Per approfondimenti su detti modelli si possono consultare i documenti indicati nel capitolo "16 References" della Specifica in questione[4]

potrà verificare la validità della richiesta contattando a sua volta il servizio "garante" (secondo WS-Trust).

Adottando dunque i modelli definiti in WS-Security, WS-Trust e WS-Policy, è possibile realizzare scenari del tipo descritto, in cui le parti possono comunicare all'interno di "domini di fiducia" creati a tempo di esecuzione.

Si noti che il servizio "Security Token Service" può essere parte del servizio Web provider, e può essere un servizio Kerberos, X.509, o può essere un più semplice servizio di validazione che utilizza nome utente e password, e tecniche di autenticazione proprie. Quest'ultima soluzione è ancora oggi la soluzione più diffusa sul Web.

2.2.4.4 WS-Policy

Definisce un modello generale e la sintassi per descrivere la "politica" di un servizio Web, attraverso un insieme di costrutti base che possono essere estesi da altre Specifiche per Web Services. Per "politica" si intendono i requisiti e le capacità del generico servizio.

Più in dettaglio, WS-Policy definisce una *policy* come un insieme di *policy alternatives*, e ognuna di queste come una collezione di *policy assertions*. Una *policy assertion* costituisce la rappresentazione di un requisito, una capacità individuale, o un proprietà comportamentale.

Ad esempio, un servizio può indicare nella propria *policy* due alternative per l'autenticazione (ognuna con un proprio protocollo), e ogni alternativa sarà descritta da asserzioni che riguardano il particolare protocollo (tali asserzioni sono definite nella Specifica del protocollo stesso, o in una "Specifica di *policy assertion*" ad esso associata). Le *policy* possono essere di tipo standard (autenticazione, selezione del protocollo di trasporto, etc), oppure non ancora standardizzate (privacy policy, caratteristiche di QoS, etc).

Ad esempio, WS-AT assume che i partecipanti e il coordinatore descrivano e pubblicizzino le proprie capacità e i propri requisiti concordemente alla famiglia di

specifiche Policy, comprendente WS-Policy, WS-PolicyAttachment e WS-PolicyAssertions.

L'asserzione che dovrà essere utilizzata per indicare che un certo servizio Web supporta o utilizza il tipo di coordinatore ATCoordinator in una particolare versione, è l'asserzione "<wsp:SpecVersion…/>", definita in WS-PolicyAssertions:

```
<wsp:SpecVersion
wsp:URI="http://schemas.xmlsoap.org/ws/2004/10/wsat"
wsp:Usage="wsp:Required"/>
```

Per indicare i singoli protocolli supportati si dovranno utilizzare, invece, le seguenti asserzioni:

```
<wsat:Completion ... />
<wsat:DurableTwoPhaseCommit ... />
<wsat:VolatileTwoPhaseCommit ... />
```

Queste ultime sono definite in WS-AT, ma secondo le convenzioni specificate in WS-PolicyAssertions.

I meccanismi definiti in WS-Policy consentono, dunque, la selezione di un servizio in possesso di particolari caratteristiche, tra i molteplici servizi dello stesso tipo disponibili sul Web, incrementando così le potenzialità di interoperabilità offerte dall'ambiente dei Web Services. Tuttavia WS-Policy non specifica come la *policy* di un servizio possa essere messa a conoscenza degli altri servizi, e non specifica come possano essere assegnate ad un particolare servizio web, rimandando il compito ad altre Specifiche. WS-PolicyAttachment definisce meccanismi in proposito.

2.2.4.5 WS-SecureConversation (WS-SecConv)

Estende WS-Security e WS-Trust per consentire una comunicazione sicura ed efficiente, anche quando questa sia composta da una lunga sequenza di singoli messaggi. In particolare definisce un meccanismo per condividere un **Contesto** in maniera sicura, e derivare da questo delle *chiavi di sessione*[99].

Il Contesto Sicuro è definito come un nuovo tipo di *token* (secondo WS-Security), che può essere scambiato utilizzando WS-Trust.

[99] Si ricorda che le *chiavi di sessione* solitamente sono delle chiavi simmetriche (stessa chiave per criptate e decriptare), e che queste sfruttano algoritmi molto più veloci di quelli adottati dalle tecniche a chiavi asimmetriche (pubblica e privata).

2.2.5 Modello per la Sicurezza

Per quanto riguarda la Sicurezza, gli autori di WS-Transactions si sono posti i seguenti obiettivi:

a. Assicurare che solo i partecipanti autorizzati possano creare Contesti di coordinazione e possano registrarsi per un'Attività.
b. Assicurare che solo i Contesti di coordinazione legittimi siano utilizzati per la registrazione.
c. Rendere possibile l'utilizzo delle esistenti infrastrutture per la Sicurezza.
d. Permettere un'autorizzazione basata su un' "identità federativa".

Il lettore noti la particolare importanza dell'obiettivo "c", il cui raggiungimento permette di scegliere i protocolli per la Sicurezza in funzione delle specifiche necessità. Ad esempio, alcune interazioni possono avvenire all'interno dello stesso dominio business, mentre altre possono essere interdominio; nelle prime sarebbe desiderabile utilizzare modelli "meno sicuri", se questo andasse a beneficio di una maggiore efficienza.

Gli obiettivi detti potrebbero essere raggiunti con molteplici tecniche, ma la Specifica raccomanda che siano rispettati i modelli e i meccanismi definiti nelle Specifiche di Supporto come di seguito esposto.

La confidenzialità e l'integrità dei messaggi possono essere ottenute mediante i meccanismi di WS-Security. Alla richiesta di creazione di una nuova Attività, il richiedente deve avere i giusti requisiti così come richiesto dalla policy del coordinatore (WS-Policy). A tal proposito il richiedente, dopo essere stato autenticato (WS-Trust), fornisce un security token con i requisiti necessari (WS-Trust). Il coordinatore, verificate le credenziali e le capacità del richiedente, fornisce un "security context token" (il nuovo contesto) contenente una chiave di sessione (WS-SecConv), che potrà essere utilizzata dai futuri partecipanti all'Attività per dimostrare che hanno l'autorizzazione per registrarsi.

Molto importante è non utilizzare la chiave per la registrazione con il coordinatore anche per assicurare il canale di comunicazione dei messaggi. La confidenzialità deve

essere realizzata con un chiave differente, e tale chiave deve essere cambiata frequentemente per evitare attacchi. Una nuova chiave non deve mai essere ottenuta dalla precedente, ma può essere ottenuta, ad esempio:

- derivandola dalla chiave di sessione e da un timestamp contenuto in ogni messaggio, mediante un'opportuno algoritmo di derivazione;
- utilizzando una sequenza di derivazione;
- chiudendo e ristabilendo un contesto sicuro;
- scambiando nuove chiavi tra le parti.

In ogni caso, per evitare attacchi alla sicurezza, il cambio delle chiavi deve ancora avvenire con i meccanismi specificati in WS-Trust e WS-SecConv.

La Specifica fornisce infine una lista di attacchi comuni, assieme alle tecniche per mitigarli o evitarli del tutto:

- Message alteration: può essere evitata firmando i messaggi (WS-Security)
- Messagge disclosure: la confidenzialità può essere ottenuta criptando i dati "sensibili" (WS- Security)
- Key integrity: può essere ottenuta utilizzando il migliore algoritmo supportato dal servizio interlocutore, come indicato nella corrispondente "policy" (WS-Policy).
- Authentication: ottenibile con i meccanismi definiti in WS- Security e WS-Trust
- Accountability: la "non rifiutabilità" può essere assicurata, in alcuni ambienti, mediante algoritmi a chiavi simmetriche, in altri, mediante più robusti algoritmi a chiave pubblica e privata.
- Availability: molti attacchi alla disponibilità sono dovuti alla possibilità di reply dei messaggi, discussa al punto seguente. Per tutti gli altri attacchi alla disponibilità, a livello di rete, la Specifica non fornisce protezione alcuna. Tuttavia consiglia che l'elaborazione sia minima prima dell'autenticazione.
- Reply: per distinguere messaggi replicati ed evitare attacchi, i messaggi devono contenere timestamp come indicato in WS-Security. Tuttavia possono essere utilizzate altre tecniche come quella della "sequenzializzazione".

2.3 WS-CAF

La più recente tra le tre Specifiche sinora trattate, Web Services Composite Application Framework, in breve WS-CAF, definisce il framework per la composizione e la coordinazione dei servizi Web proposto da Sun Microsystems, Oracle Corporation, Arjuna Technologies, Fujitsu e IONA Technologies.

Come WS-Transactions, è composta da tre sotto-Specifiche distinte, ma la logica di suddivisione è diversa:

- WS-Context (WS-CTX): definisce un meccanismo standard per relazionare più servizi e permettere la condivisione di informazioni, quando detti servizi collaborino in un'attività distribuita attraverso molteplici interazioni.
- WS-Coordination Framework (WS-CF): definisce un generico servizio di coordinamento, utilizzando quanto già definito in WS-CTX.
- WS-Transaction Management (WS-TXM): definisce tre specifici protocolli di coordinamento per supportare Transazioni Distribuite, "realizzando" il generico servizio di coordinamento definito da WS-CF.

Le tre Specifiche, nell'intento degli autori, possono essere utilizzate singolarmente, a seconda delle necessità applicative. Esse si rivolgono alla tecnologia dei Web Services, per la quale vengono forniti gli XML Schema e la descrizione WSDL di tutti i servizi definiti.

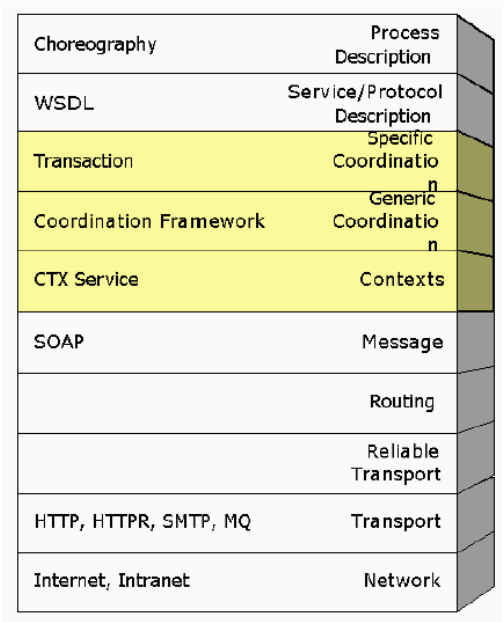

Figura 48 Architettura Web Services secondo Sun-Oracle
- tratta da Riferimenti[8 p12] -

La Figura 48 mostra lo stack architetturale per Web Services dal punto di vista Sun-Oracle, e dove siano collocate le Specifiche oggetto del presente capitolo. Come può notarsi, esistono delle differenze sostanziali rispetto all'architettura definita da Microsoft-IBM. Fortunatamente, l'adozione in comune di alcuni livelli fondamentali, quali SOAP e WSDL, assieme alla flessibilità offerta dall'XML, consente una pacifica convivenza tra i diversi standard dei due gruppi industriali.

Si noti pure che alcuni livelli nell'architettura Sun-Oracle, come può dedursi dalla presenza dei campi vuoti in figura, sono definiti nell'architettura, ma, di fatto, non sono ancora supportati.

2.3.1 WS-CTX

La Specifica “Web Services Context”, in breve WS-CTX, nasce per supportare interazioni tra servizi Web che eseguono Activities correlate in un’applicazione principale. Un’Activity è definita come un insieme di lavori distribuiti che coinvolgono una o più parti (ad esempio servizi, componenti, oggetti). Ogni lavoro di un tale insieme ha necessità di essere correlato con gli altri all’interno dell’Attività: il meccanismo definito allo scopo da WS-CTX è il solito meccanismo della propagazione del Contesto. Tuttavia, rispetto alle Specifiche di BTP e WS-Transactions, WS-CAF sottolinea la possibilità di utilizzare detto Contesto non solo per correlare operazioni, ma anche per condividere informazioni di comune interesse tra gli esecutori. Ad esempio, il Contesto potrebbe essere utilizzato per correlare messaggi appartenenti alla stessa sessione[100] di elaborazione, ma anche per condividere *credenziali* di Sicurezza, connessioni a database, e più in generale, connessioni a generici dispositivi.

Ad ogni modo, il concetto di Attività va inteso come un’astrazione avente lo scopo di correlare lavori eseguiti da soggetti distinti in un oggetto unico: detta correlazione viene realizzata proprio dalla propagazione del Contesto agli esecutori dei lavori. Per esempio, quando nel seguito utilizzeremo la locuzione <<l’Attività è stata avviata>>, dovrà essere interpretata come <<i servizi ai quali è stato richiesto, nell’ambito dell’Attività, una certa elaborazione, la hanno avviata>>.

Il Contesto è gestito dal ContextService, il cui compito è quello di supportare la nozione astratta di Attività (il ciclo di vita che definiremo in seguito), permettendo che le informazioni ad essa correlate (stato dell’Attività e Contesto) siano propagate nell’ambiente di esecuzione quando necessario. Ad esempio, un sistema di gestione ad eventi potrebbe voler essere informato sull’inizio e la fine di un’Attività, così da poter distribuire questa informazione alle parti interessate. Allo stesso modo, un servizio di coordinazione transazionale ha necessità di conoscere quando l’Attività dei servizi sia terminata, al fine di poter avviare l’opportuno protocollo di coordinamento.

[100] Il concetto “sessione di elaborazione” è utilizzato con il significato che tradizionalmente assume nell’ambito delle applicazioni Web.

Gli autori della Specifica precisano che il framework definito da WS-CTX può essere personalizzato dalle applicazioni e dai servizi nella maniera che si ritiene più opportuna.

2.3.1.1 Modello Concettuale

In Figura 49 è mostrato un possibile modello concettuale estrapolato da WS-CTX. La parte del modello nella zona superiore della figura è molto simile a quanto già visto in BTP e WS-Coordination. L'unica differenza sostanziale consiste nell'attributo "status", che caratterizza un'**Attività**, la quale, pur essendo definita come un'entità astratta, assume in WS-CTX un ruolo concreto e centrale.

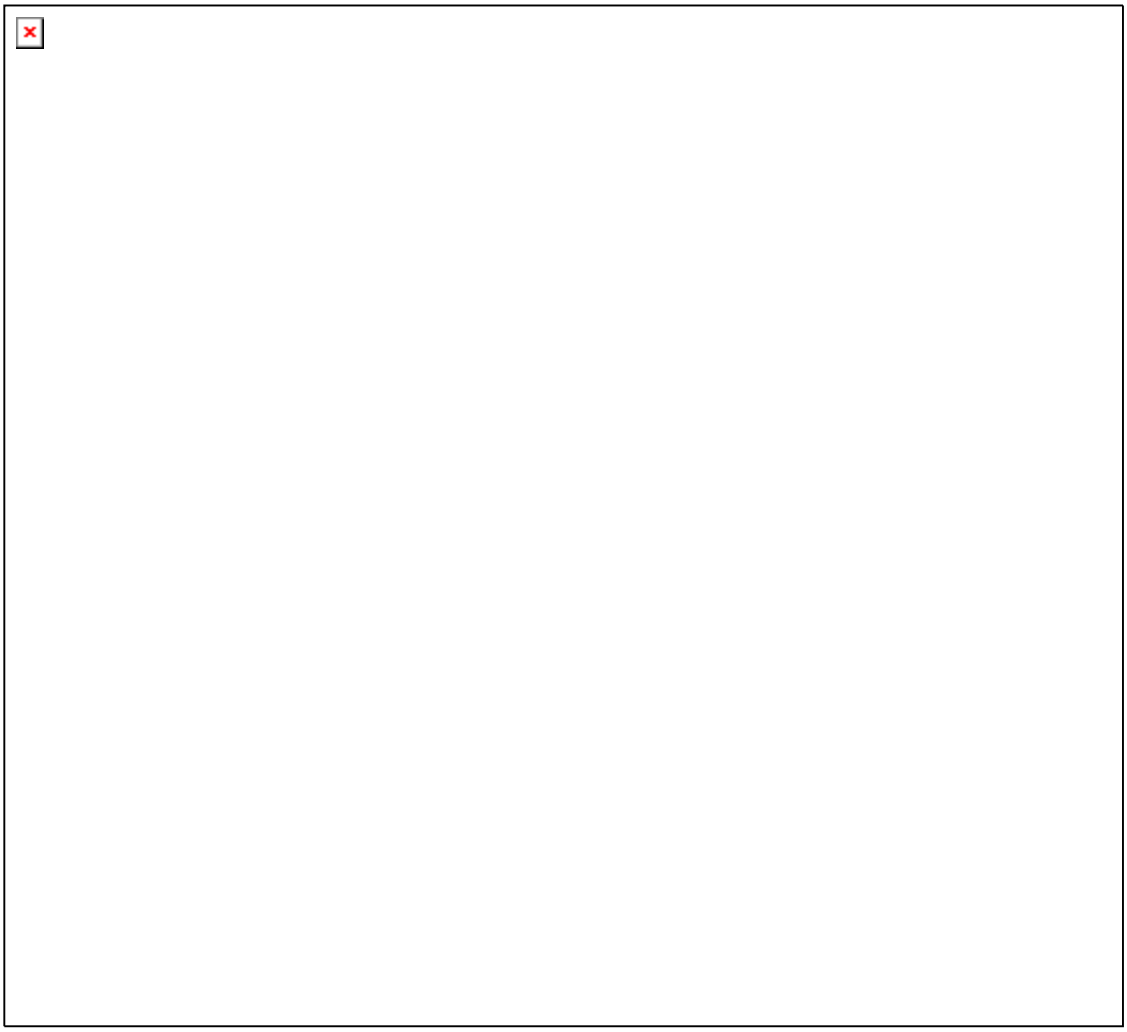

Figura 49 Modello Concettuale di WS-CTX

Un'Attività è rappresentata da un **Context**, il quale contiene le seguenti informazioni:

- Un identificatore di tipo URI chiamato "context-identifier". Garantisce l'unicità globale di ogni Attività, e può essere utilizzato per correlare i vari task che la realizzano. Obbligatorio.
- L'identificatore del Context Service responsabile della generazione del Context. Opzionale.
- Il tipo di Attività. Opzionale.
- Una lista di servizi che partecipano correntemente all'Attività. Opzionale.
- Una lista di Attività figlie, all'interno dell'elemento "child-contexts". Opzionale.
- Un timeout di "expired". Indica per quanto tempo le informazioni del Contesto saranno valide: dopo tale tempo, potranno essere dimenticate. Questo serve affinché un'Implementazione del Context Service, in assenza di ulteriori stimoli esterni, possa terminare un'Attività piuttosto che lasciarla avviata per sempre.

Oltre alle suddette informazioni, il Context può essere esteso per trasportarne altre, a seconda delle particolari necessità dell'Implementazione: ad esempio, in un sistema transazionale potrà trasportare l'URI del coordinatore, mentre in un sistema per lo scambio Sicuro dei dati, potrà contenere la chiave pubblica del mittente.

L'estensione dei campi previsti, così come l'aggiornamento, può essere operata in qualsiasi momento e da qualsiasi servizio partecipante all'Attività, dove per partecipante si intenda qualunque servizio coinvolto in qualche maniera nell'Attività stessa. In Figura 49 il generico servizio partecipante è rappresentato dall'entità **Service**.

Il **ContextService**, come detto, rappresenta l'entità che gestisce il ciclo di vita di un'Attività: mantiene e fornisce il Context quando richiesto; e informa le ALSes registrate su quando l'Attività inizia e termina, permettendo loro di influenzarne il "risultato di completamento". Vedremo in seguito cosa questo significhi, ma è opportuno notare sin d'ora, come sottolineato dalla Specifica, che il ContextService non possiede capacità di coordinazione.

Potendo gestire più Attività contemporaneamente, il ContextService mantiene un repository dei Contesti ad esse associati. WS-CTX precisa che può essere implementato come un servizio indipendente, oppure aggiungendo semantica al comportamento dei

partecipanti, e precisa che potrebbe essere un servizio web collocato in un sito remoto, o nello stesso sito degli utilizzatori.

Il ContextService viene controllato dall'applicazione utente, detta **ClientApplication**, che compone i singoli servizi partecipanti, detti ApplicationService (AS), richiedendone la collaborazione all'interno di un'Attività mediante messaggi applicativi. Come già osservato, per richiedere lavori nell'ambito di una specifica Attività, i messaggi applicativi devono contenere il Context che la rappresenta. Un messaggio applicativo può contenere il Context vero e proprio, oppure può più semplicemente indicare l'URI dal quale ricavarlo[101]: esso infatti può essere gestito come una risorsa Web grazie ad un'opportuna interfaccia, denominata ContextManager, che descriveremo in seguito.

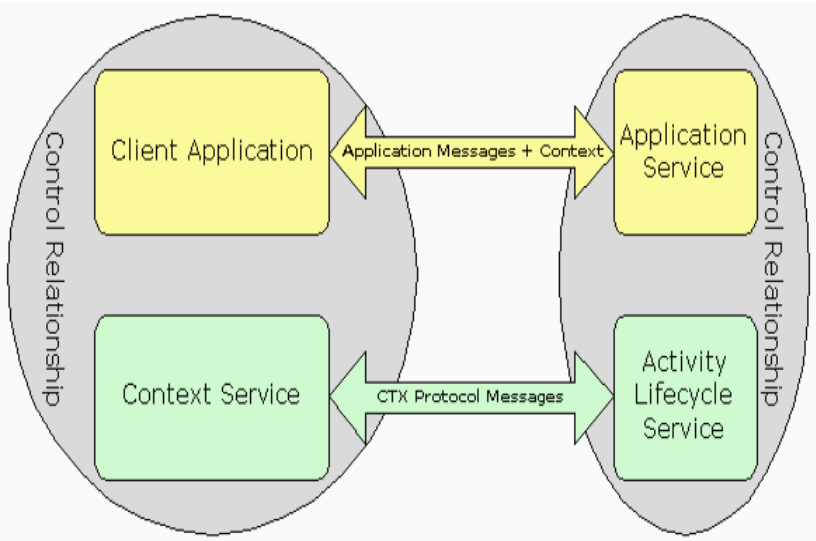

Figura 50 Tipi di messaggi scambiati tra i componenti di WS-CTX
- tratta da Riferimenti[8 p12] -

L'entità **ApplicationService** rappresenta i partecipanti che svolgono le funzionalità applicative dell'Attività, definendola concretamente. L'entità **ActivityLifecycleService**[102] (ALS) rappresenta, invece, i partecipanti che sono coinvolti nel ciclo di vita dell'Attività. Un ALS viene informata quando l'Attività inizia, quando deve essere portata a termine, e quando è stata completata. Tali notifiche provengono dal ContextService con cui l'ALS risulta registrata, e riguardano tutte le

[101] WS-CTX specifica che la scelta deve essere dell'utente.
[102] L'ALS sarebbe potuta essere rappresentata più propriamente come un ruolo dell'AS, ma WS-CTX la specifica come un'entità distinta.

Attività da questo create. Tramite un ALS, inoltre, un ContextService potrebbe registrarsi come partecipante presso un altro ContextService, a seconda dei bisogni dell'applicazione.

Un **ALS** è sempre associato ad un AS, in una relazione di "controllo" non specificata da WS-CTX e, come consentito a tutte le entità, può sempre utilizzare il Context relativo ad un'Attività per personalizzarlo e propagarlo a sua volta verso altri servizi, secondo le proprie esigenze: l'unico vincolo imposto da WS-CTX è che tale "personalizzazione" deve essere, nel caso di un'ALS, un'estensione del Context originario, piuttosto che una manipolazione[103].

Ad esempio, per un'Attività transazionale la ClientApplication potrebbe essere il servizio *richiedente*, ovvero il servizio che compone l'Attività, e l'AS il servizio coordinatore. Quest'ultimo sarebbe informato, tramite l'ALS associata, sulla terminazione dei lavori relativi all'Attività, e potrebbe così avviare l'opportuno protocollo di coordinamento con i partecipanti alla transazione. Inoltre, detto coordinatore, potrebbe estendere il Context inserendo il proprio identificatore e quello dei partecipanti.

Si noti che nella terminologia qui adottata, anche il coordinatore viene denominato "partecipante". WS-CTX, infatti, fa riferimento ad un'Attività generica, non necessariamente transazionale, nei confronti della quale un coordinatore è un servizio come tutti gli altri.

Le ALSes possono essere raggruppate secondo le particolari esigenze applicative. Ogni gruppo viene identificato da un URI denominato ALSConfigurationIdentifier. Tale identificatore deve essere indicato nel messaggio con cui l'ALS viene registrato presso il ContextService, e nei messaggi con cui la ClientApplication "pilota" lo stato delle Attività. Il ContextService notificherà solo le ALSes del gruppo richiesto.

Un'Attività può essere creata (Active), avviata (Completing) e completata (Completed). Viene creata su richiesta di un servizio Web, e viene posta

[103] Il vincolo imposto ad un ALS potrebbe sembrare insensato: probabilmente rispecchia un requisito di interoperabilità (tra le ALSes) non richiesto alle entità più esterne nell'architettura. L'ALS, insieme al ContextService, rappresenta infatti un'entità "propria" di WS-CTX, mentre AS e ClientApplication rappresentano, in realtà, elementi applicativi specifici.

automaticamente nello stato di Active. Il creante viene sempre informato quando l'Attività inizia e termina, e ha pieno accesso al Contesto che la rappresenta.

Nello stato di Active sono definiti dei sotto-stati di completamento, ognuno dei quali rappresenta lo stato nel quale si richiede che l'Attività venga completata. Prima di avviare il completamento di un'Attività, lo stato di completamento detto può variare tra Success e Fail, un numero di volte arbitrario. Nel caso in cui tale stato venga portato a FailOnly, invece, non potrà più variare.

Dallo stato di Active, l'Attività passerà nello stato di Completing, indicante che le operazioni di completamento sono in corso, ed infine transiterà nello stato di Completed. Il risultato di completamento ("outcame") di un'Attività, sarà comunicato al creante mediante un messaggio di notifica. Tale risultato potrà essere di successo (Success) o di insuccesso (Fail o FailOnly), e potrà, se necessario, essere utilizzato per determinare il flusso di controllo di altre Attività.

Il ciclo di vita di un'Attività, come entità astratta, è mostrato nella figura seguente:

Figura 51 StateChart Diagram di un'Attività

Il significato dei sotto-stati di completamento, del risultato di completamento, e degli stati di un'Attività, dovrà essere concretizzato dalla particolare applicazione. WS-CTX precisa infatti che un'Attività può durare molto tempo (minuti, ore o giorni), e durante il suo ciclo di vita può richiedere periodi di coordinazione dei suoi costituenti (i partecipanti che saranno definiti in WS-CF), o può avere la necessità di interagire con altri servizi, come un gestore transazionale o un servizio per gestire la Sicurezza; ma

non fornisce la definizione concreta di cosa sia un'Attività, poiché tale definizione non può prescindere dall'applicazione nella quale viene utilizzata.

Per esempio, un'applicazione di immagazzinamento-vendita può essere costituita da diverse attività, quali attività per determinare i quantitativi di merce disponibile, per effettuare nuovi ordinativi, per ottenere la consegna della merce ordinata etc. Analogamente, un servizio di vendita libri on-line, può essere decomposto in attività per consentire all'utente di inserire libri nel carrello, attività per confermare l'acquisto, attività per effettuare il pagamento mediante carta di credito, etc.

In entrambi i casi, la definizione concreta di Attività dipende dalla particolare applicazione, e dalla necessità di questa di condividere informazioni tra i vari componenti che realizzano l'applicazione stessa. Ad esempio, per l'applicazione di immagazzinamento-vendita potrebbe essere conveniente condividere il prezzo della merce in magazzino, mentre per il servizio di vendita libri on-line potrebbe essere utile condividere un inventario sulla quantità di libri correntemente disponibili.

Infine, le **Attività** possono essere tra di loro "innestate", in maniera simile a quanto già visto per le transazioni. In tal modo è possibile correlare gruppi di lavori, ognuno rappresentato da una singola Attività, in una struttura gerarchica. Seppure non indicato esplicitamente da WS-CTX, dalla specifica dei protocolli che tratteremo in seguito, può dedursi che le conseguenze della relazione di "innesto" sono essenzialmente tre:

a) affinché l'Attività "padre" possa terminare con successo, devono prima essere terminate tutte le Attività "figlie" (indipendentemente dallo stato di completamento di queste);
b) se si richiede il completamento dell'Attività "padre" in uno stato di fallimento, il ContextService completerà automaticamente tutte le Attività "figlie" nello stato di FailOnly;
c) il tempo di vita di un'Attività "figlia" viene automaticamente settato pari a quello dell'Attività "padre".

2.3.1.2 Protocolli

Le interfacce corrispondenti ai vari ruoli definiti dai protocolli di seguito descritti sono riassunte nel modello di Figura 52 a pag.199. Questo può essere considerato come un modello di secondo livello del Modello Concettuale spiegato nel paragrafo precedente.

2.3.1.2.1 Gestione del Context come risorsa Web

Affinché il Context possa essere gestito come una risorsa Web, ottenibile ad uno specifico URL, è stato definito il ruolo **ContextManager** del ContextService. Tale ruolo accetta i messaggi:

- *getContents*: il mittente richiede, specificando l'URI, l'intero contenuto del Context corrispondente;
- *setContents*: il mittente richiede che il contenuto corrente del Context venga sostituito con le informazioni di contesto fornite in questo messaggio.

Dall'altro lato, il generico servizio che attua nel ruolo di **ContextRespondant**, potrà ricevere in risposta uno dei messaggi di seguito elencati:

- *contents*: contiene il Context, relativo ad una precedente richiesta di *getContents*;
- *contentsSet*: riscontro, indicante che il contenuto del Context è stato aggiornato con successo in seguito ad una richiesta di *setContents*;
- *unknownContextFault*: il Context indicato nella richiesta non è stato individuato;
- *generalFault*: indica l'occorrenza di un errore imprecisato.

2.3.1.2.2 Arruolamento di ALSes

Per permettere l'arruolamento di un'ALS presso il ContextService, questo espone una specifica interfaccia, denominata **ALSRegistrar**. L'arruolante, per avere riscontro sull'esito dell'operazione, deve implementare invece l'interfaccia **ALSRegistrant**.

Il servizio che vuole arruolare un ALS ne deve fornire l'indirizzo, nel messaggio *enlistALS*, assieme all'ALSConfigurationIdentifier che identifica il gruppo di ALSes per cui si vuole l'arruolamento. In seguito riceverà una delle tre notifiche seguenti:

- *enlisted*: arruolamento riuscito con successo;
- *invalidALSFault*: l'ALS specificato nella richiesta non era valido;
- *generalFault*: indica l'occorrenza di un errore imprecisato.

In caso di successo, l'ALS sarà arruolato con tutte le Attività via via create dal ContextService che ha accettato l'arruolamento. Affinché un ALS arruolato non riceva più notifiche relative al ciclo di vita delle Attività di un certo ContextService, deve essere inviato all'ALSRegistrar un messaggio di *delistALS*, specificando l'indirizzo dell'ALS da "dearruolare" e l'ALSConfigurationIdentifier del gruppo di ALSes. In caso di successo dell'operazione, in risposta sarà ottenuta la notifica di *delisted*; in caso di insuccesso, un messaggio di *invalidALSFault* o di *generalFault*.

2.3.1.2.3 Protocollo tra ALS e ContextService

Come precedentemente discusso, ogni ALS registrata con un ContextService viene da questo invocata durante il ciclo di vita delle Attività, per riceverne notifiche sull'avanzamento. In dette invocazioni è sempre "propagato" anche il Contesto, che eventualmente può essere modificato dall'ALS secondo propria convenienza. L'ALS potrebbe sfruttare tali notifiche per effettuare del lavoro applicativo specifico: ad esempio, quando un'Attività sia terminata, potrebbe avviare un particolare protocollo di coordinamento.

Affinché possano essere scambiati messaggi tra ALS e ContexService, WS-CTX definisce due ruoli specifici, cui corrispondono altrettante interfacce:

- **ALS-Int**: deve essere implementata da un'ALS per ricevere le notifiche.
- **ALSRespondant**: interfaccia esposta dal ContextService, per ricevere riscontri sulle notifiche effettuate.

I messaggi che il ContextService può inviare ad un'ALS, contestualizzati con il Context dell'Attività a cui si riferiscono, sono:

- *begin*: una nuova Attività è stata creata e portata nello stato di Active.
- *completeWithStatus*: indica che l'Attività è entrata nello stato di Completing, e ne specifica lo stato di completamento corrente (Success, Fail o FailOnly).
- *complete*: l'Attività è stata completata, ed è transitata nello stato di Completed. Tale messaggio è inviato dal Context Service per dare alle ALSes l'opportunità di effettuare eventuali operazioni di chiusura, ad esempio operazioni di *clean-up*.
- *getIdentity*: utilizzato per richiede l'identità dell'ALS specificata nel messaggio.

I messaggi inviati in risposta all'ALSRespondant, sempre contestualizzati, sono invece:

- *begun*: messaggio di riscontro per il *begin*.
- *completedWithStatus*: messaggio di riscontro per il *completeWithStatus*. Indica lo stato di completamento precedentemente richiesto.
- *completed*: messaggio di riscontro per il *complete*.

- *identity*: messaggio utilizzato per ritornare al ContextService l'identificatore univoco dell'ALS che ha ricevuto la richiesta *getIdentity*.
- *invalidALSFault*: l'ALS risponde con questo messaggio ad una precedente notifica non valida nel contesto di "appartenenza" (quello in cui l'ALS risulta arruolata).
- *validContextExpectedFault*: il Context ricevuto nella notifica non era valido.
- *generalFault*: indica l'occorrenza di un errore imprecisato.

WS-CTX specifica la sequenza di messaggi ammessa come segue:

Il Context Service invia il messaggio di *begin* all'ALS, che risponderà con un messaggio di riscontro *begun*, o con messaggi di errore.

Se il Context Service non riceve errori, invierà il messaggio di *complete* o *completeWithStatus* all'ALS, che potrà rispondere eventualmente con il corrispondente messaggio di riscontro o con messaggi di errore.

Infine, il Context Service invierà il messaggio di *complete* all'ALS, che potrà rispondere, come al solito, mediante riscontro o messaggi di errore.

WS-CTX non specifica, invece, come il ContextService debba comportarsi rispetto alla "collezione" di ALSes, poiché esso comunica solo delle notifiche, e in ogni caso, non specifica un comportamento differente del ContextService, in funzione delle diverse risposte producibili da un'ALS nella sequenza di messaggi lecita. Ad esempio, cosa dovrebbe accadere se un'ALS non dovesse rispondere ad una notifica?

2.3.1.2.4 Protocollo tra ClientApplication e ContextService

Affinché lo stato di un'Attività possa essere "pilotato" da un'applicazione client, è necessario che il ContextService esponga un'appropriata interfaccia. L'interfaccia in questione è stata denominata CTXService. La ClientApplication può utilizzare i messaggi da questa previsti specificando in tutti l'ALSConfigurationIdentifier del set di ALSes a cui si riferisce: solo le ALSes registrate con lo stesso ALSConfigurationIdentifier riceveranno, dal ContextService, le notifiche che diremo di seguito. Dall'altro lato, affinché la ClientApplication possa ricevere notifiche sulle operazioni richieste mediante CTXService, deve implementare l'interfaccia UserCTXService. I messaggi accettati dal ContextService nel ruolo di **CTXService** sono:

- *begin*: la ClientApplication richiede la creazione di una nuova Attività. Il ContextService la crea e ne inizializza il Contesto, assegnandole un nuovo identificatore. Ogni ALS precedentemente arruolato viene notificato a sua volta mediante un messaggio di *begin*. Se nel messaggio di *begin* comunicato dalla ClientApplication era presente anche un Context, la nuova Attività sarà creata come una sotto-Attività di quella corrispondente a detto Context.

  Il parametro di "timeout", passato con il messaggio di *begin*, indica il tempo di vita dell'Attività. Oltre questo tempo il ContextService potrà portarla automaticamente allo stato di Completed con Fail. I valori che possono essere indicati per il parametro di "timeout" sono:

  - un valore positivo: indica il tempo di vita in secondi;
  - -1: tempo di vita infinito;
  - 0: tempo di vita pari al valore specificato con l'ultima chiamata a *setTimeout*.

  Nel caso fosse indicato un valore differente da quelli ammissibili, il ContextService ritornerebbe un messaggio di *timeoutOutOfRangeFault*.

  In mancanza di errori, il ContextService porta l'Attività nello stato di Active, e ritorna un messaggio di riscontro *begun* allo UserCTXService.

- *complete*: la ClientApplication richiede che l'Attività sia portata a completamento con il valore del CompletitionStatus specificato nell'ultima chiamata a *setCompletitionStatus*, oppure, se tale messaggio non era stato inviato, con lo stato di default, corrispondente allo stato di Fail. Quando viene inviato il messaggio di *complete*, se il CompletitionStatus è Success, ma ci sono delle Attività "figlie" ancora nello stato di Active, il ContextService ritorna un messaggio di errore *childActivityPendingFault*: in tal caso, per completare l'Attività, l'applicazione dovrà prima completare tutte le sotto-Attività, oppure dovrà richiedere il completamento dell'Attività "padre" nello stato di completamento con Fail o con FailOnly.

  Se lo stato di completamento corrente è Fail o FailOnly, e viene richiesto il *complete*, il ContextService porta automaticamente tutti gli stati di completamento delle Attività "figlie" pure a FailOnly.

  In ogni caso, il ContextService invia un messaggio di *completeWithStatus* a tutte le ALSes registrate, seguito da un messaggio di *complete*, e infine invia un messaggio di *completed* allo UserCTXService, con l'indicazione dello stato di completamento effettivo[104].

  L'implementazione di un ContextService può imporre delle restrizioni su chi può richiedere la terminazione di un'Attività. Nel caso in cui tali restrizioni non siano soddisfatte, potrà rispondere con un messaggio di *noPermissionFault.*

  Se l'Attività non può terminare nello stato di completamento richiesto dalla ClientApplication (ad esempio è marcata come FailOnly ed è stato richiesto il Success), allora il ContextService invierà a questa un messaggio di *activityCompleted*.

- *competeWithStatus*: equivale al messaggio di *setCompletitionStatus* seguito dal messaggio di *complete*.
- *setCompletitionStatus*: la ClientApplication richiede che lo stato di completamento dell'Attività sia portato a quello indicato nel messaggio. Tale messaggio può essere inviato molteplici volte durante il ciclo di vita

[104]Lo stato di completamento "effettivo" può essere differente da quello richiesto solo nel caso esso sia di FailOnly. Si noti, sia per il messaggio di *complete*, sia per quello di *completeWithStatus*, che in caso di ricezione di messaggi di errore non viene specificato come il ContextService debba comportarsi.

dell'Attività, cambiando ogni volta lo stato di completamento che sarà utilizzato dal ContextService quando si richiederà il *complete*. Se quest'ultima operazione non sarà richiamata durante il tempo di vita dell'Attività, lo stato di completamento sarà portato automaticamente a quello di default (Fail).

Se lo stato di completamento viene settato senza errori, la ClientApplication otterrà in risposta il messaggio di riscontro *completitionStatusSet*.

- *getCompletitionStatus*: la ClientApplication richiede il valore dello stato di completamento effettivo. In Risposta potrà ottenere uno dei seguenti valori:
    - Success: tutti i lavori dell'Attività sono stati eseguiti. Tale stato può variare.
    - Fail: sono occorsi degli errori specifici dell'applicazione, ovvero non è stato possibile eseguire tutto il lavoro richiesto nell'abito dell'Attività. Tale stato può variare.
    - Fail_Only: va interpretato come lo stato di Fail, ma senza possibilità di variazioni ulteriori. Pertanto l'Attività potrà avere come unico risultato (*outcame*) quello di fallimento.
    - Unknown: il Context Service non può, momentaneamente, determinare con certezza lo stato di completamento dell'Attività. E' una condizione transitoria.
- *getStatus*: la ClientApplication richiede lo stato corrente dell'Attività. In risposta otterrà un messaggio di *status* che potrà indicare:
    - Active: l'Attività è iniziata, e non è ancora partita la fase di completamento.
    - Completing: è partita la fase di completamento, ma l'Attività ha bisogno di ulteriore lavoro prima di essere completata.
    - Completed: l'Attività associata è stata completata.
    - No_Activity: indicato dopo che l'Attività è stata completata, o prima che sia stata avviata.
    - Unknown: indica una condizione transiente, che occorre quando il Context Service non riesce a determinare lo stato corrente. Una ulteriore richiesta di *getStatus* può ritornare uno stato differente, e

un'Implementazione potrebbe scegliere di ritentare la richiesta in maniera trasparente all'utente.

- *getActivityName*: la ClientApplication richiede che gli sia ritornata una stringa stampabile che descriva l'Attività.
- *getContext*: la ClientApplication, indicando l'identificatore univoco dell'Attività, richiede che gli sia ritornato l'URI del Context a questa associato.
- *setTimeout*: la ClientApplication richiede che il tempo di vita dell'Attività specificata nel messaggio in questione, e quello di tutte le Attività che eventualmente saranno create come Attività "figlie" di questa, sia settato al tempo indicato nel messaggio. Se viene richiesto un valore accettabile, il ContextService risponderà con un messaggio di riscontro *timeoutSet*.
- *getTimeout*: la ClientApplication richiede il valore corrente del parametro di "timeout".

I messaggi di risposta, ritornati dal ContextService alla ClientApplication, previsti dall'interfaccia **UserCTXService** sono: *begun*; *completed(status)*; *completitionStatusSet*; *completitionStatus*; *status*; *activityName*; *context*; *timeoutSet*; *timeout*.

A tale elenco vanno aggiunti i messaggi di errore:

- *invalidStateFault*: ritornato quando si richieda la creazione di una sotto-Attività "figlia" di un'Attività che abbia stato di completamento FailOnly, oppure quando, sempre per un'Attività con stato di completamento FailOnly, sia ricevuto un messaggio *setCompletitionStatus* che richieda una variazione di tale stato.
- *invalidActivityFault*: ritornato quando sia richiesto un *begin*, un *complete*, o un *setCompletitionStatus* per un'Attività che sia già Completing o Completed.
- *timeoutOutOfRangeFault*;
- *childActivityPendingFault*;
- *noActivityFault*: ritornato quando non ci sono Attività associate con il Context ricevuto in una precedente richiesta.
- *noPermissionFault*;

- *validContextExpectedFault*: ritornato quando il Context indicato in una precedente richiesta non era valido.
- *generalFault*: ritornato per segnalare errori imprecisati. Eventualmente può specificare il tipo di errore nel campo “AssertionType” in esso contenuto.

Il significato dei messaggi non specificati esplicitamente per la UserCTXService, è stato implicitamente definito nella descrizione dell’interfaccia CTXService.

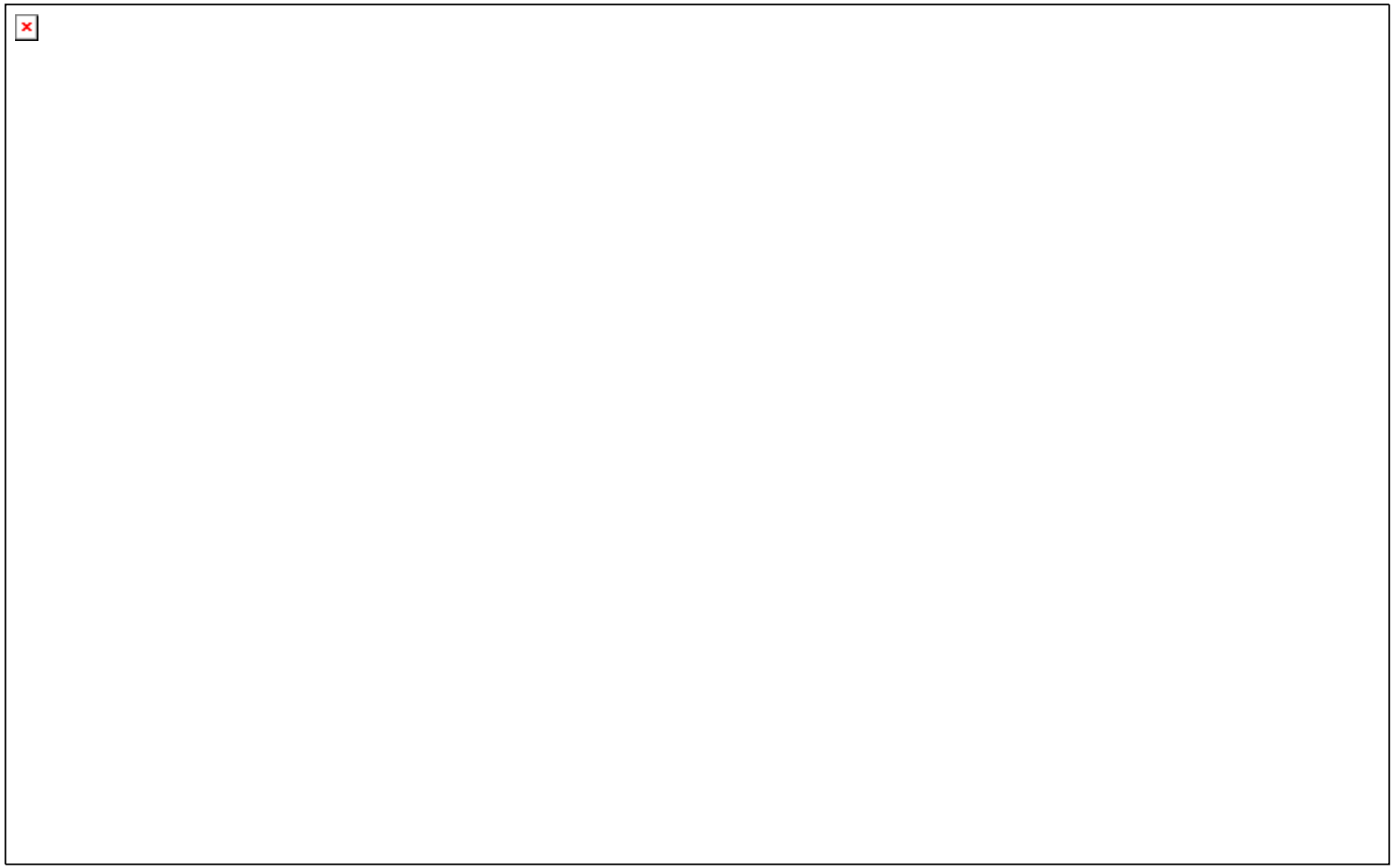

Figura 52 Modello delle interfacce di WS-CTX

In Figura 53 viene riportato un semplice esempio di interazioni conformi WS-CTX, in cui si ipotizza sia coinvolta una sola ALS. Nel Sequence Diagram si ipotizza pure che il ContextService, in seguito all'invio dei messaggi di *completeWithStatus* e di *complete*, ottenga i rispettivi messaggi di riscontro, prima che informi la ClientApplication sul completamento dell'Attività. Come precedentemente osservato, WS-CTX non specifica per il ContextService un comportamento preciso in funzione delle risposte fornite dalle ALSes, e dunque è lecito supporre che queste siano scelte implementative.

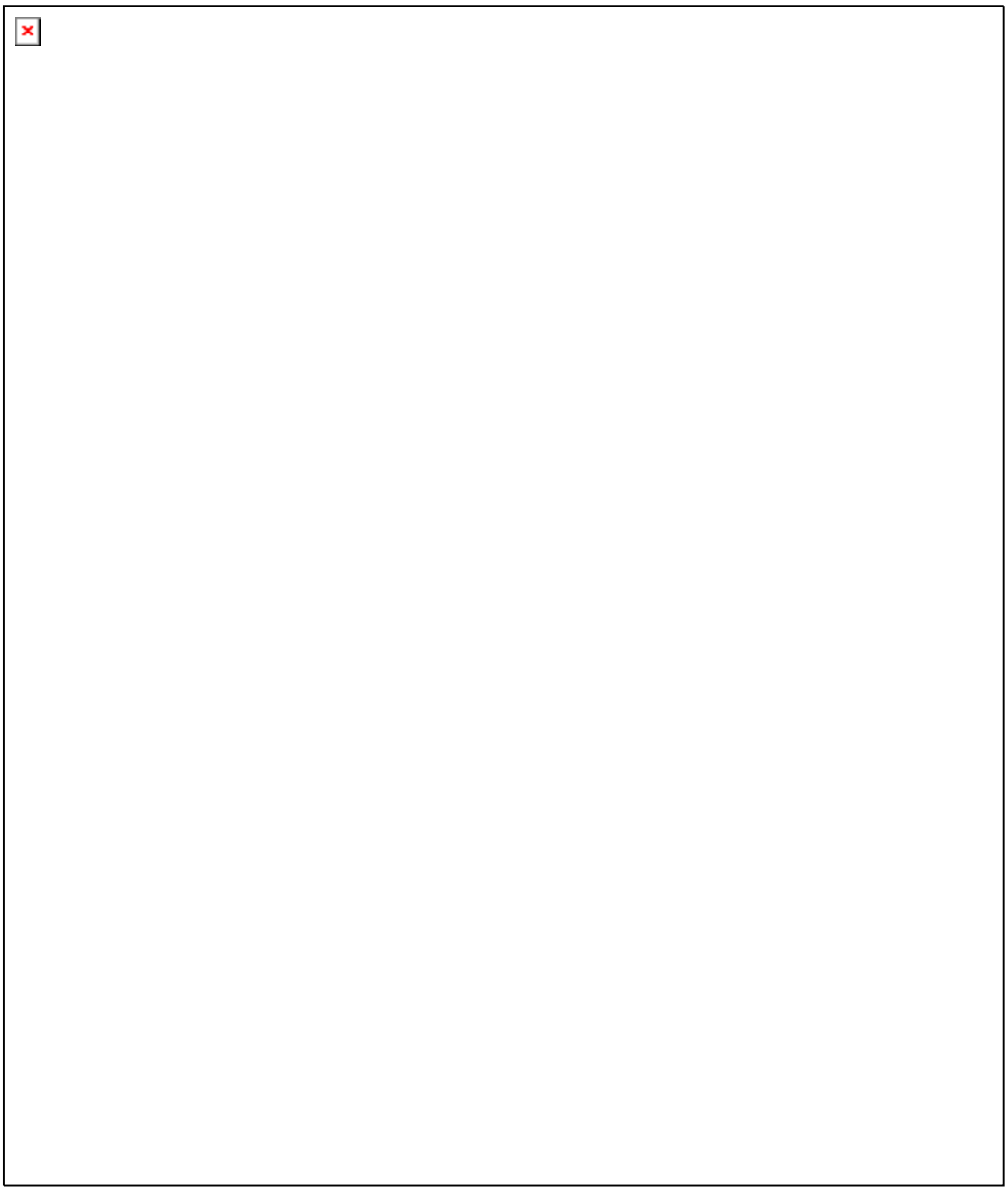

Figura 53 Semplici interazioni conformi WS-CTX

2.3.1.3 Binding

La Specifica si rivolge esplicitamente alla tecnologia dei WebServices. Precisa che per la contestualizzazione dei messaggi nell'ambito di una certa Attività, il corrispondente Context, o semplicemente l'URI dal quale è possibile ottenerlo, devono essere inseriti nell'header SOAP di tali messaggi. In generale il Context, oltre che all'interno di altri messaggi, può essere propagato anche come messaggio autonomo. In entrambi i casi l'attributo "mustUnderstand", previsto dal protocollo SOAP, deve essere posto a "true", in maniera che il ricevente sia consapevole di dover interpretare un messaggio nel particolare ambito di WS-CAF.

WS-CTX specifica che le modalità di scambio dei messaggi, e le modalità di esposizione dei metodi forniti dai vari servizi (tramite WSDL), sono una scelta Implementativa: i messaggi possono essere scambiati sia mediante il modello "request/reply" sincrono, sia mediante il modello "request/reply" asincrono, e possono essere codificati secondo il modello "document" o secondo il modello "RPC"[105].

Affinché ciò sia possibile, WS-CTX richiede che tutti i messaggi contengano l'indirizzo del mittente, in maniera che il ricevente possa conoscere sempre a chi rispondere, e un identificatore del messaggio stesso, in maniera che richiesta e risposta possano tra loro essere correlate dalla parte del mittente. La Specifica precisa che tali requisiti sono necessari in attesa che la "comunità dei WebServices" concordi su degli standard[106] per l'Addressing e il Reliable Messaging (indirizzamento e "scambio affidabile dei messaggi").

[105] I modelli citati sono definiti nelle ultime versioni della Specifica SOAP.
[106] Corrispondono proprio ai due livelli non specificati nell'architettura dei WebServices mostrata in Figura 48 a pag.182.

2.3.1.4 Failure Handling

Le condizioni di errore sono gestite tramite messaggi integrati nei vari protocolli precedentemente descritti. Tuttavia gli autori ammettono che le condizioni contemplate sono solo alcune; per tutte le altre, mettono a disposizione il generico messaggio *generalFault*.

2.3.2 WS-CF

La Specifica “Web Services Coordination Framework”, in breve WS-CF, estende WS-CTX definendo un framework di supporto per tutte quelle applicazioni distribuite che necessitano di una qualche forma di coordinamento, quali applicazioni per la composizione transazionale, per la gestione del controllo di flusso, della Sicurezza, di attività business-to-business, o applicazioni di *caching* e replicazione.

Definisce il concetto di “Coordinazione” come “l’atto del coordinatore di disseminare informazioni ad un certo numero di partecipanti”, ma non definisce gli specifici protocolli che la realizzano, poiché assume che sia impossibile coprire tutte le necessità applicative: si pone come obiettivo la definizione di meccanismi generici, estensibili secondo convenienza, con cui poter realizzare protocolli di coordinamento specifici per particolari problematiche. Alcuni di questi protocolli saranno definiti invece da WS-TXM, trattata nel prossimo capitolo, ma chiunque potrebbe definirne di nuovi. Eventualmente, protocolli definiti da terze parti potrebbero essere adattati per rispettare le specifiche stabilite da WS-CF.

Utilizzando i modelli definiti da WS-CF e WS-CTX, è possibile dunque realizzare un’applicazione come composizione di molteplici Attività generiche, all’interno delle quali siano eventualmente eseguiti degli opportuni protocolli di coordinamento.

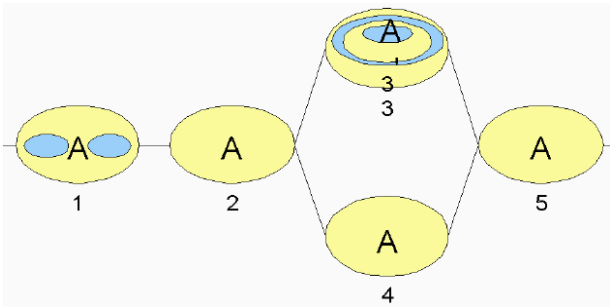

Figura 54 Esempio di composizione di Attività Coordinate e non
- tratta da Riferimenti[9 p13] -

Ad esempio, in Figura 54 sono mostrate una serie di Attività cooperanti durante il ciclo di vita di un'applicazione. Le ellissi scure rappresentano i confini delle Coordinazioni, mentre quelle chiare delimitano i confini delle Attività.

Nell'esempio:

- L'Attività A1 utilizza la Coordinazione in due punti della sua esecuzione. Nel primo potrebbe utilizzare, ad esempio, un protocollo 2PC, mentre nel secondo, un protocollo a tre fasi potrebbe essere più appropriato.[107]
- L'Attività A2 non utilizza nessuna Coordinazione.
- L'Attività A3 utilizza una Coordinazione e, al suo interno, un'Attività innestata A3', che a sua volta esegue un'ulteriore Coordinazione.

2.3.2.1 Modello Concettuale

WS-CF aggiunge al concetto di Attività, definito da WS-CTX, il concetto di Coordinazione. Come suddetto, una **Coordinazione** rappresenta semplicemente l'atto di un Agente, il Coordinator, di disseminare informazioni secondo opportuni protocolli ad un certo numero di partecipanti, detti CoordinationParticipant.

Un CoordinationParticipant rappresenta un'Agente da coordinare. Alla ricezione di un messaggio di coordinazione, questo può effettuare qualsiasi lavoro specifico per l'applicazione: ad esempio, può eseguire nuove Attività, Attività di "compensazione", operazioni di modifica su database, avvio di altre Coordinazioni, etc.

Come notato nell'esempio introduttivo, in ogni Attività possono essere eseguite molteplici Coordinazioni. Un'Attività cui è associato un Coordinator viene detta CoordinatedActivity. Quest'ultima, come l'entità da cui deriva, rappresenta un concetto astratto, che viene rappresentato concretamente da un Contesto. Il Contesto per un'Attività Coordinata è detto CoordinationContext. Esso deriva dal Context definito per un'Activity, al quale aggiunge alcune informazioni proprie: in particolare l'indirizzo, e il tipo di Coordinator.

[107] Nell'esempio, il lettore noti che anche all'interno della stessa Attività sono possibili Coordinazioni di tipo differente (in A1, 2PC prima e 3PC poi).

Detta specializzazione fornisce, dunque, un Contesto ampliato, che potrà essere propagato tra i servizi partecipanti per stabilire l'ambiente di esecuzione. Infatti, affinché i riceventi siano consapevoli di prendere parte ad una specifica Attività Coordinata, il CoordinationContext dovrà essere propagato nei messaggi applicativi scambiati tra i servizi cooperanti.

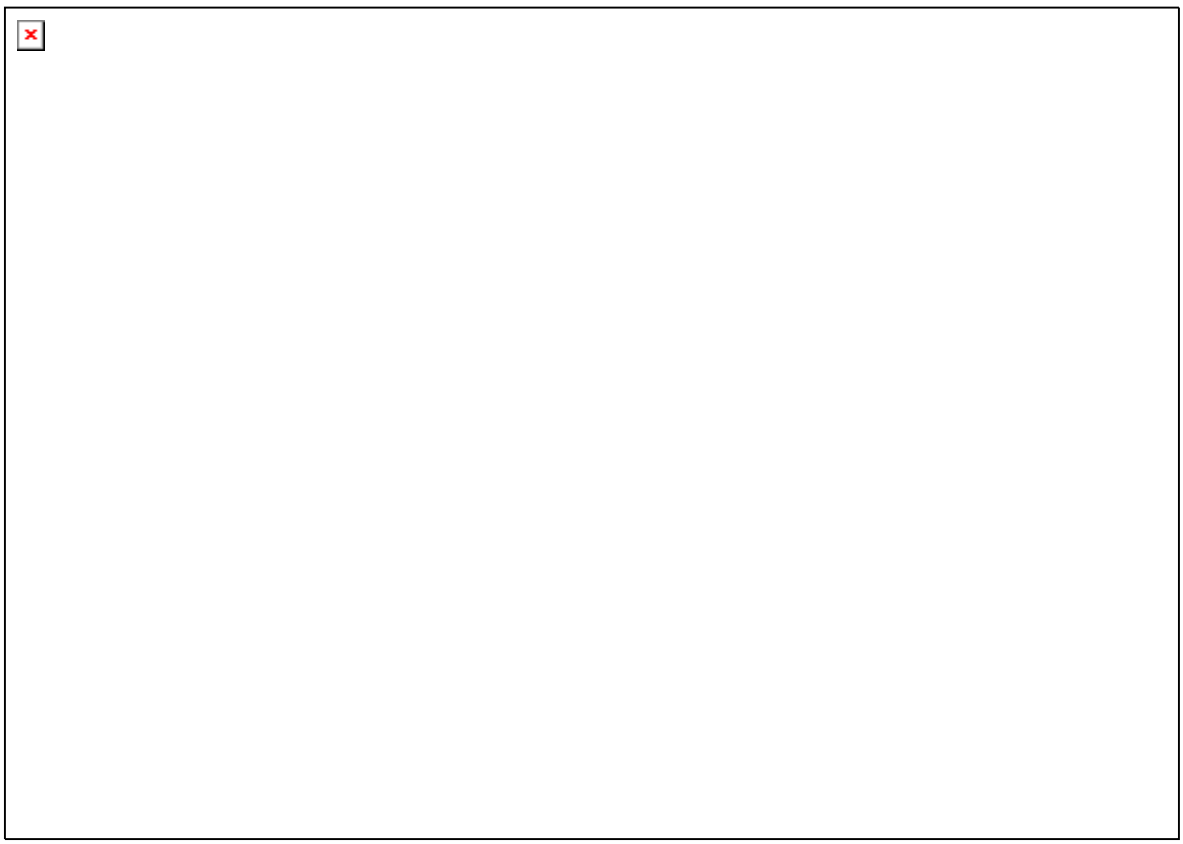

Figura 55 Modello Concettuale di WS-CF

Ogni Attività Coordinata è gestita da un **Coordinator**. Come risulterà più chiaro dalla lettura del prossimo paragrafo, il Coordinator rappresenta anch'esso un'entità astratta, poichè solo alcune delle relazioni in cui è coinvolto sono stabilite concretamente: in pratica solo la relazione con cui permette l'arruolamento dei partecipanti. Per le altre, sarà il **CoordinationService** a specializzare la sintassi e il significato dei messaggi previsti, per definire uno specifico modello di coordinamento, e particolari protocolli per tale modello. In altri termini, l'entità Coordinator rappresenta l'infrastruttura di base per qualsiasi CoordinationService, mentre quest'ultimo ne rappresenta una particolare specializzazione per supportare un particolare modello di coordinamento. Per esempio, un CoordinationService per transazioni ACID specializzerà il Coordinator per definire

un protocollo 2PC, la cui sequenza di coordinamento include messaggi di Prepare, Commit e Rollback.

Diverse Implementazioni di CoordinationService, relative ad uno stesso modello di coordinamento o a modelli diversi, possono coesistere nello stesso dominio applicativo, e ogni modello può supportare molteplici protocolli di coordinamento. Affinché un servizio partecipante possa essere coordinato, deve essere registrato presso un CoordinationService, e deve supportare il protocollo di coordinamento scelto tra quelli disponibili.

La realizzazione di un servizio di coordinamento, dunque, corrisponde all'Implementazione di uno specifico CoordinationService, il quale, utilizzando quanto già definito da WS-CF, realizza a sua volta il generico Coordinator. Per chiarezza, nel seguito di questo testo, utilizzeremo il termine "Coordinator" per indicare un generico CoordinationService, e il termine "CoordinationService" per indicare una specifica realizzazione di Coordinator.

Alla definizione di uno specifico CoordinationService dovrà essere associata quella di specifici CoordinationParticipant e ClientApplication. Tuttavia, per supportare servizi di coordinazione già esistenti, che dunque possono avere già definito le interfacce di coordinatore e partecipante e l'insieme dei messaggi, ad una Implementazione compatibile con WS-CF si richiede solo di implementare l'ALS. Il coordinatore associato a tale ALS potrà così partecipare alle Attività e personalizzare il Contesto di base fornito da WS-CTX. In tal caso WS-CF assume che le interfacce e i protocolli per questi servizi siano definiti altrove, e conosciuti dagli utenti dei servizi.

Un Coordinator può eseguire la coordinazione al completamento dell'Attività, oppure in un qualsiasi momento del ciclo di vita di questa, stabilito dinamicamente su richiesta della ClientApplication. Nel primo caso, sarà l'ALS associata al Coordinator, denominata CoordinationServiceALS, ad indicare al Coordinator quando l'Attività sia terminata. Nel secondo, invece, sarà un opportuno servizio utente, nel ruolo di ClientApplication, che, mediante uno specifico messaggio, indicherà al coordinatore quando avviare la Coordinazione.

Ad esempio, quando il CoordinationServiceALS sia avvisato dal ContextService sull'inizio dell'Attività, potrebbe avviare una specifica istanza di CoordinationService,

mentre quando sia avvisata del completamento, potrebbe richiedere a tale istanza l'esecuzione della Coordinazione.

In ogni caso, dovrà essere la Specifica del particolare modello di CoordinationService a stabilire quando, e se, sia possibile avviare una Coordinazione.

Nell'intento della Specifica, il concetto di CoordinatedActivity può essere utilizzato in generale, per raggruppare Attività correlate anche quando queste che non siano di tipo transazionale. A pagina 219 sarà riportato un esempio di questo tipo.

Negoziazione del protocollo di coordinazione

Essendo possibile che un servizio Web supporti differenti modelli di coordinamento, ovvero diverse specializzazioni di Coordinator (ad esempio con differente livello di qualità), è necessaria una qualche forma di "negoziazione", che permetta di stabilire quale particolare modello debba essere utilizzato. Se un servizio non dovesse supportare il modello di CoordinationService richiesto, l'applicazione potrebbe valutare se sia il caso di continuare ad utilizzarlo. Ad esempio, un'applicazione transazionale potrebbe non voler lavorare con servizi che non supportano la semantica transazionale.

Affinché la negoziazione sia possibile, è necessario, prima di tutto, identificare ogni protocollo univocamente: WS-CF richiede che ogni protocollo di coordinamento sia univocamente identificato da un URI, ma precisa che non è suo compito definire come l'URI possa essere associato allo specifico protocollo che rappresenta.

WS-CF non definisce come i servizi possano scambiarsi informazioni circa i modelli supportati (ciò che Microsoft ed IBM ottengono mediante WS-Policy), e non definisce nessun protocollo per la negoziazione. Tuttavia impone delle limitazioni a questi ultimi. In particolare richiede che un protocollo di negoziazione permetta lo scambio dei suoi messaggi indipendentemente dallo scambio dei messaggi ordinari, e sottolinea l'importanza che la negoziazione non richieda requisiti addizionali sul protocollo di Trasporto.

WS-CF suggerisce inoltre che, per conoscere il modello di coordinazione supportato da un particolare CoordinationService, può essere utilizzato il messaggio di *identity* previsto per l'ALS ad esso associata, ma precisa che sono consentiti anche altri meccanismi proprietari.

Interposizione

Per evitare sovraccarichi per un singolo coordinatore, WS-CF permette che una Coordinazione sia gestita tramite coordinatori subordinati (SubordinateCoordinator), mediante la tecnica dell' Interposizione: il coordinatore deve semplicemente registrarsi come partecipante presso un altro coordinatore.

Ogni coordinatore di questo tipo gestirà dunque una sotto-CoordinatedActivity, e avrà un proprio CoorinationContext, poiché, rispetto a quello del coordinatore "padre", dovrà cambiare almeno il campo contenente l'indirizzo del coordinatore.

Inoltre, poiché un coordinatore subordinato deve eseguire un protocollo di coordinamento con i propri arruolati, WS-CF richiede che esso abbia il proprio file di log, e il corrispondente sottosistema per il recovery in caso di *failures*: il subordinato dovrà ricordare sufficienti informazioni per il recovery, sia del lavoro che esegue come partecipante, sia del lavoro che esegue come coordinatore.

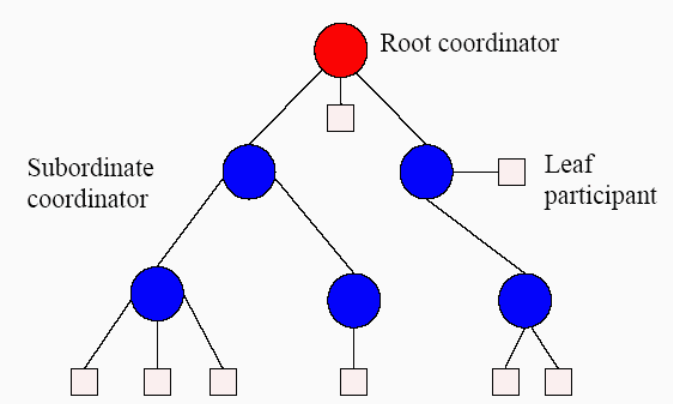

Figura 56 Interposizione di coordinatori
- tratta da Riferimenti[9 p30] -

In un'applicazione distribuita di grandi dimensioni, se necessario, può essere creato un albero di coordinatori e di partecipanti potenzialmente illimitato.

WS-CF non richiede che l'Interposizione sia necessariamente supportata, ma lascia la scelta alle singole Implementazioni. Inoltre, l'Interposizione nulla implica su eventuali Attività innestate, e viceversa: sono due concetti distinti, il primo riguardante il comportamento di un partecipante, il secondo riguardante il comportamento del

ContextService. Eventualmente, il coordinatore di un'Attività innestata potrebbe essere anche un Interposto, ma non necessariamente.

Supporto per il Recovery

A differenza dei sistemi transazionali classici, in cui i meccanismi per il recovery vogliono garantire solo la consistenza dei dati, WS-CF assume che le applicazioni che utilizzeranno un CoordinationService richiederanno tipicamente anche dei meccanismi per recuperare lo stato delle Attività alle quali partecipano, affinché, anche in caso di *failures*, possano proseguire senza dover riprendere tutto dall'inizio.

A tal proposito WS-CF definisce dei messaggi generici mediante i quali, nel caso in cui dovesse avere problemi durante l'esecuzione della Coordinazione, e conseguentemente dovesse perdere informazioni circa lo stato di avanzamento della stessa, un partecipante possa tentare di ripristinare lo stato "dimenticato" richiedendolo direttamente al coordinatore. Detti messaggi sono gestiti mediante delle opportune interfacce, cui corrispondono nuovi ruoli definiti da WS-CF. Li descriveremo in dettaglio nel paragrafo "Recovery Handling" a pag.220.

WS-CF specifica inoltre che:

- La persistenza degli oggetti non è una responsabilità del CoorinationService.
- Il recovery deve essere guidato dall'applicazione, poiché questa dà significato ai partecipanti ed ai messaggi scambiati. Le stesse informazioni da memorizzare dipendono dall'applicazione: queste devono essere tali da poter ripristinare uno stato consistente degli oggetti applicativi anche in caso di *failures*. Per garantire la consistenza, dunque, si richiede logica applicativa non solo per guidare le Attività durante il funzionamento normale, ma anche durante le operazioni di recovery.
- Le tecniche impiegate per garantire la persistenza sono una scelta implementativa.
- La semantica dei messaggi per il recovery deve essere definita dall'Implementazione, ed è strettamente legata al particolare protocollo di coordinamento. Pertanto, specificazioni circa quanto tempo il coordinatore e i partecipanti devono ricordare eventi di *failures*, non possono essere e non sono stabilite da WS-CF.

Responsabilità

La Specifica precisa le responsabilità dei fornitori[108] di servizi Web "conformi WS-CF", come di seguito esposto.

Fornitore del CoordinationServiceALS: questo servizio, specificato in WS-CTX, deve permettere che l'applicazione stabilisca i punti di inizio e fine per un'Attività Coordinata, e "piloti" il risultato dell'Attività. Inoltre deve collegarsi con lo specifico CoordinationService attraverso una relazione non definita da WS-CF.

Fornitore del CoordinationService: un CoordinationService fornisce un modo per coordinare un certo numero di task che hanno un interesse comune. Esempi sono servizi per Attività transazionali (come Java JTS o OMG OTS), per Attività transazionali estese/rilassate, o per Attività di tipo non transazionale. La definizione di un servizio di coordinazione deve comprendere quella dei relativi protocolli da esso supportati, dove per "protocollo" si intenda:

- Protocollo: definisce le caratteristiche del servizio di coordinazione e il contratto/obbligazioni per i partecipanti in un'Attività Coordinata.

Fornitore del CoordinationParticipant: un servizio Web che voglia partecipare in Attività Coordinata conforme WS-CF deve:

- Fornire un'Implementazione di CoordinationParticipant per rispondere ai messaggi provenienti dall'Implementazione del CoordinationService. Eventualmente più Implementazioni di CoordinationParticipant potrebbero supportare differenti livelli di Qualità del Servizio, ed essere utilizzate in maniera intercambiabile in funzione di quanto richiesto dallo specifico servizio di coordinazione utilizzato per l'Attività.
- Supportare le interfacce di CoordinationParticipant, e quelle per comunicare con il servizio che ha effettuato l'arruolamento. Quest'ultimo potrebbe essere il partecipante stesso, ma non necessariamente.

[108] Per "fornitore" deve intendersi il soggetto che espone il servizio Web.

2.3.2.1.1 Possibili estensioni future di WS-CF

Gli autori di WS-CF ammettono l'importanza di alcuni argomenti non trattati nella versione attuale della Specifica, e considerano l'eventualità di aggiungerli in future versioni. Gli argomenti in questione sono:

- **Sicurezza e confidenzialità**;
- **Audit trail**: mantenere un "log" delle azioni intraprese durante una transazione business è utile per una serie di ragioni, non ultima quella della "non rifiutabilità" in caso di azioni legali.
- **Garanzia di completamento del protocollo**: in caso di *failures* potrebbe essere conveniente rendere disponibile un *repository*, accessibile a tutti, che indichi quali Attività bisogna invocare e in quali circostanze (ad esempio, nel caso di caduta del coordinatore principale).
- **Quality of Service**: diversi modelli e diversi protocolli per uno stesso modello, ognuno dei quali avente un differente livello di Qualità di Servizio, dovrebbero essere supportati contemporaneamente.

Per il momento WS-CF assume che siano le Implementazioni a prendere provvedimenti in proposito, e non definisce esplicitamente nulla per supportare le caratteristiche elencate.

2.3.2.2 Protocolli

Come anticipato nella descrizione del Modello Concettuale, i vari protocolli definiti da WS-CF costituiscono un'infrastruttura generale, per la definizione di particolari modelli di coordinamento. Pertanto, essi possono essere considerati, analogamente al Coordinator, come dei protocolli astratti: stabiliscono la struttura dei messaggi, ma non il contenuto (il valore dei campi), che dovrà invece essere specificato dai particolari modelli di coordinamento. Volendo fare un'analogia con la programmazione orientata

agli oggetti, ogni messaggio WS-CF rappresenta una classe, e ogni modello di coordinazione, attribuendo un valore ai campi, ne utilizzerà una particolare istanza.

Le interfacce dei protocolli definiti da WS-CF sono riassunte nel Modello di Figura 57. In questo, l'interfaccia ErrorInt è stata da noi introdotta semplicemente per evidenziare un insieme di messaggi condivisi tra più interfacce. Come per il Modello Concettuale, l'entità Service rappresenta un generico servizio: può essere sostituita da un qualsiasi servizio definito da WS-CF, ad esempio da quello dell'applicazione client o da quello del partecipante, oppure può rappresentare un nuovo servizio, legato ai primi in qualche maniera.

Seppure l'XML permetta naturalmente l'estensione dei messaggi e dei protocolli mediante la semplice definizione di nuovi elementi, tale flessibilità può venire meno utilizzando XML Schema. In tal caso, infatti, affinché l'interprete XML non generi errori, eventuali campi di estensione devono essere previsti in anticipo, ed indicati nello Schema dei messaggi.

A tal proposito WS-CF prevede un apposito elemento strutturato in XML e denominato Qualificatore, che, incluso in un qualsiasi messaggio di protocollo, può fornire informazioni addizionali. Ad esempio, un'Implementazione potrebbe definire uno specifico Qualificatore con cui un partecipante, nelle operazioni di arruolamento in un'Attività transazionale, potrebbe indicare al coordinatore l'intervallo di tempo in cui rimarrà disponibile per eseguire la propria parte di lavoro.

Il concetto di Qualificatore lo abbiamo già incontrato nella descrizione di BTP ma, a differenza di quest'ultimo, WS-CF non definisce Qualificatori standard. Alcuni Qualificatori saranno invece definiti da WS-TXM, che descriveremo nel prossimo capitolo.

Figura 57 Modello delle Interfacce di WS-CF

2.3.2.2.1 Protocollo di Arruolamento

Affinché un generico servizio possa arruolare/"dearruolare" un partecipante in un'Attività Coordinata, WS-CF definisce due ruoli specifici:

- **ServiceCoordinator**: ruolo del Coordinator, o di un servizio ad esso associato in qualche modo (per semplicità, in Figura 57 è mostrato come ruolo del Coordinator).
- **ServiceRespondand**: ruolo del servizio che richiede l'arruolamento del partecipante.

Detti ruoli accettano i messaggi rappresentati in Figura 57 nelle omonime interfacce. Il significato dei messaggi è descritto di seguito:

- *addParticipant*: è inviato al ServiceCoordinator per richiedere la registrazione di uno specifico partecipante con uno specifico CoordinationService. Esso contiene pertanto un riferimento al partecipante in questione, e un Context, per identificare la particolare Attività Coordinata. Potendo il CoordinationService utilizzare più protocolli, in tale messaggio deve essere specificato anche il campo "protocolType", indicando l'URI del protocollo scelto.
  In risposta, il ServiceRespondant otterrà:
  - *invalidCoordinator*, se il CoordinationService indicato nel Context non potesse essere localizzato;
  - *invalidProtocol*, se il CoordinationService richiesto non dovesse supportare il protocollo specificato nella richiesta;
  - *duplicateParticipant*, se il partecipante fosse stato già arruolato, e se lo specifico CoordinationService non dovesse supportare arruolamenti duplicati;
  - *invalidParticipant*, se il partecipante risultasse non valido nell'ambito della Coordinazione;
  - *wrongState*, se l'Attività di coordinamento corrispondente al Context avesse già iniziato la fase di completamento, o fosse già completata del tutto;

Nel caso in cui la registrazione sia stata condotta con successo, sarà ritornato il messaggio di riscontro *participantAdded*, contenente l'indirizzo del ParticipantCoordinator che coordinerà il partecipante in questione.

- *removeParticipant*: inviato al ServiceCoordinator per cancellare la registrazione di un partecipante precedentemente arruolato. Tale messaggio contiene, oltre al Context, l'identificativo del partecipante in questione.
  In risposta, il ServiceRespondant otterrà:
  - *participantNotFound*, se il partecipante indicato non dovesse risultare registrato;
  - *invalidCoordinator*, se il coordinatore non potesse essere localizzato;
  - *wrongState*, se l'Attività corrispondente al Context avesse già iniziato la fase di completamento, o fosse stata già completata;

  Il messaggio di *removeParticipant* qui descritto potrebbe non essere supportato da tutte le Implementazione di Coordinator, poiché alcuni protocolli di coordinamento potrebbero non permettere questo tipo di operazione.
- *getParentCoordinator*: inviato al ServiceCoordinator per richiedere l'indirizzo dell' eventuale Coordinator "padre" di quello indicato nel Context associato al messaggio. In risposta si otterrà un messaggio di *parentCoordinator*, indicante detto indirizzo, o un indirizzo "vuoto", nel caso in cui il Coordinator in questione non abbia un "padre".
  Se il Coordinator corrispondente al Context non potesse essere localizzato, in risposta si otterrebbe un messaggio di *invalidCoordinator*.
- *getQualifiers*: inviato al Coordinator per richiedergli la lista di tutti i Qualificatori correntemente supportati. In risposta si otterrà detta lista in un messaggio di *qualifiers*.
  Se il Coordinator corrispondente al Context non potesse essere localizzato, in risposta si otterrebbe un messaggio di *invalidCoordinator*.

Si noti che il servizio che effettuerà l'arruolamento non è precisato. Sono dunque possibili sia arruolamenti di tipo "Pull" che di tipo "Push". Tuttavia, utilizzando l'arruolamento di tipo "Push", nel caso in cui il servizio esecutore crei un CoordinationParticipant dinamicamente, servirebbe un meccanismo standard con cui

l'esecutore possa comunicare l'indirizzo di detto CoordinationParticipant al richiedente, meccanismo che non è stato definito[109]. Si noti pure che nel Context la lista dei partecipanti è opzionale, per cui, utilizzando il modello "Pull", un servizio potrebbe dearruolarsi in maniera trasparente al servizio del richiedente, essendo, questo, distinto dal servizio coordinatore.

2.3.2.2.2 Protocollo tra coordinatore e applicazione client

Alcuni protocolli di coordinamento prevedono che la Coordinazione sia avviata sotto esplicito comando dell'applicazione. Per supportare questo tipo di operazione, WS-CF definisce i messaggi con cui la ClientApplication può richiedere l'avvio della Coordinazione e lo stato corrente del coordinatore, e definisce i corrispondeti ruoli (interfacce), rispettivamente **ClientRespondant** per la ClientApplication, e **ClientCoordinator** per il Coordinator.

I ruoli in questione accettano i messaggi rappresentati in Figura 57, nelle omonime interfacce. Il loro significato viene esposto di seguito:

- *coordinate*: se il protocollo di coordinazione, specificato dall'URI, consente questo tipo di richiesta, il coordinatore inizierà l'esecuzione del protocollo con i partecipanti correntemente arruolati, purchè la richiesta sia stata ricevuta prima della terminazione del ciclo di vita dell'Attività Coordinata. Il coordinatore può essere invocato più volte durante il ciclo di vita dell'Attività, ed è possibile che diversi messaggi di coordinamento siano inviati ad ogni distinta invocazione. Una volta che i partecipanti abbiano processato un messaggio di coordinamento e prodotto ognuno un risultato ("outcome"), è compito del coordinatore tradurre tali risultati in un risultato unico per l'Attività Coordinata, risultato che viene comunicato al ClientRespondant assieme al messaggio di riscontro *coordinated*.
- *getStatus*: richiede lo stato del coordinatore. In risposta si otterrà un messaggio di *status*, che potrà indicare un valore di stato specificato per il ContextService,

[109] Lo stesso problema viene risolto in BTP introducendo il concetto di Identitificativo.

o un valore di stato specificato dal particolare modello/protocollo di coordinazione.

2.3.2.2.3 Coordinazione dei partecipanti

Le interazioni di richiesta/risposta di un generico protocollo di coordinazione tra Coordinator e CoordinationParticipant vengono classificati da WS-CF come:

- Interazioni su iniziativa del Coordinator: utilizzate per i protocolli di coordinazione tradizionali. Interazioni di questo tipo, ad esempio, sono quelle costituite dalle coppie “prepare/prepared” o “committ/committed” del classico 2PC.
- Interazioni su iniziativa del CoordinationParticipant: utilizzate per alcuni protocolli di coordinazione che permettono ai partecipanti di prendere decisioni autonome, o di formulare previsioni circa le notifiche che riceveranno dal coordinatore.

Per lo scambio dei messaggi su iniziativa del Coordinator verso il CoordinationParticipant, sono stati definiti rispettivamente i ruoli di CoordinatorParticipant e Participant.

Un **Participant** accetta i seguenti messaggi:

- *getStatus*: richiede lo stato del Participant.
- *assertionType*: contiene un messaggio di coordinazione specifico. Ad esempio, potrebbe contenere un messaggio di “committ” per un protocollo di tipo 2PC.
- *getIdentity*: richiede l’identificatore univoco del Participant.

I messaggi di risposta, accettati dal **CoordinatorParticipant**, sono:

- *status*: fornisce lo stato del Participant.
- *assertionType*: contiene la risposta ad un messaggio di coordinazione specifico. Potrebbe contenere un semplice messaggio di riscontro.
- *identity*: contiene l’identificatore univoco del Participant.
- *wrongState*: <non specificato>.
- *generalFault*: indica un errore generico.

Eventuali altri messaggi di errore possono essere definiti come specifiche istanze del messaggio *assertionType*.

Per lo scambio di messaggi su iniziativa del CoordinationParticipant, WS-CF definisce altri due ruoli:

- **PartecipantCoordinator**: rappresenta il coordinatore. Il suo indirizzo è noto al partecipante in seguito al processo di registrazione.
- **PartecipantRespondant**: rappresenta il partecipante.

Un partecipante può inviare un messaggio di *setResponse*, con cui può indicare un'eventuale decisione autonoma o una previsione di notificazione da parte del coordinatore.

Al coordinatore si richiede di ricordare la decisione del partecipante fino a quando gli avrebbe inviato il messaggio di *assertionType* concorde con tale decisione. Nel caso in cui le due non fossero concordi, è compito dell'Implementazione decidere cosa fare.

Dopo aver memorizzato con successo la decisione ricevuta, il coordinatore ritornerà un messaggio di riscontro *responseSet* al partecipante.

Possibili messaggi di errore, che potrebbero essere scambiati in queste interazioni, sono:

- *unknownCoordinator*: inviato dal partecipante se questo non riconosce l'identità del coordinatore.
- *protocolViolation*: inviato dal coordinatore verso un partecipante che abbia inviato un messaggio incompatibile con lo stato corrente del coordinatore.
- *wrongState*: inviato dal coordinatore ad un partecipante, quando il primo si rifiuti di accettare un messaggio precedentemente inviato dal secondo.

Tutti i messaggi descritti sono riassunti nel modello delle interfacce mostrato in Figura 57.

2.3.2.3 Esempio di utilizzo di WS-CF

Un esempio di utilizzo concreto dei meccanismi definiti da WS-CF è fornito dalla Specifica stessa. Esso mostra come sia possibile coordinare il flusso di lavoro di più Attività mediante un protocollo opportunamente definito. Nell'esempio, il coordinatore dell'Attività principale avvia l'esecuzione di altre Attività, ognuna responsabile per effettuare una certa unità di lavoro, ed eventualmente per fornire un risultato. Il ciclo di vita di ogni Attività è guidato da un singolo partecipante. Considerando il coordinatore e i partecipanti al posto delle Attività che rappresentano, il protocollo prevede tre messaggi:

- **start**: inviato dall'Attività "padre" alle Attività "figlie" (tramite il messaggio di *assertionType*). Tale messaggio può contenere delle informazioni addizionali per parametrizzare l'avvio dell'Attività.
- **start_ack**: inviato da un'Attività "figlia" come riscontro per un messaggio di *start* (tramite il messaggio di *assertionType*).
- **outcome**: inviato da un'Attività "figlia" all'Attività "padre" per indicare che l'esecuzione è stata completa (tramite il messaggio di *setResponse*) e per fornire il risultato dell'Attività stessa (tramite il campo "AssertionType" del messaggio di *setResponse*).

In Figura 58 viene mostrato uno scenario in cui l'Attività principale <a> coordina l'esecuzione parallela delle Attività <b> e <c>, e in seguito, sulla base del risultato ottenuto da queste, avvia l'Attività <d>.

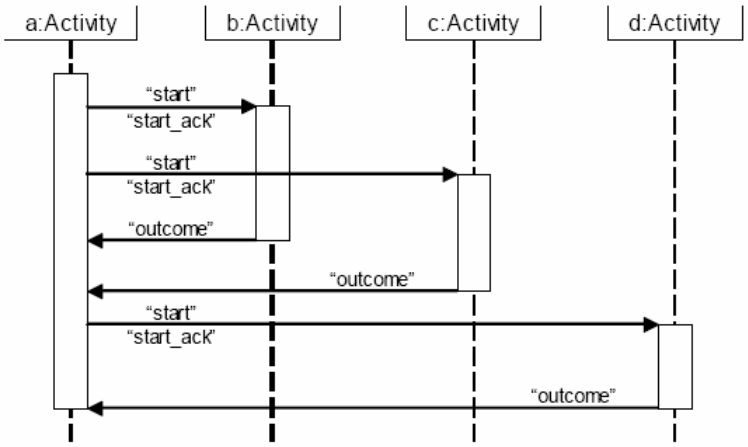

Figura 58 Esempio di personalizzazione di WS-CF per coordinare il flusso di esecuzione di Attività.
- tratta da Riferimenti[9 p36] -

2.3.2.4 Recovery Handling

Per supportare il recovery, WS-CF definisce due ruoli specifici: **RecoveryCoordinator** e **RecoveryParticipant**, ripettivamente per il coordinatore, e per il servizio partecipante. Quest'ultimo potrebbe essere sostituito da un altro servizio, appositamente utilizzato nelle operazioni di recovery per guidare una nuova istanza dello stesso partecipante "caduto", o un servizio partecipante completamente nuovo.

Il significato dei messaggi corrispondenti ai ruoli suddetti è riportato di seguito:

- *recover*: messaggio utilizzato da un partecipante per richiedere al coordinatore che inizi le operazioni di recovery. Può anche indicare al coordinatore che, dopo il recovery, il partecipante in questione non risiede più al precedente indirizzo.
  In risposta, il RecoveryParticipant può ottenere:
  - *unknownCoordinator*, se il coordinatore indicato non può essere localizzato;
  - *wrongState*, se lo stato del coordinatore è tale che il recovery non è consentito al momento della richiesta.

  In assenza di errori, sarà ritornato un messaggio di *recoverResponse*, indicante lo stato corrente dell'Attività Coordinata.

- *getStatus*: utilizzato per richiedere lo stato del coordinatore. In risposta si otterrà un messaggio di *status*, che potrà indicare un valore di stato specificato per il ContextService, o un valore di stato specificato dal particolare modello/protocollo di coordinamanto.

2.3.2.5 Binding e Failure Handling

Per quanto concerne questi due argomenti, WS-CF non aggiunge ulteriori specificazioni a quanto già definito in WS-CTX.

2.3.3 WS-TXM

La Specifica “Web Services Transaction Management”, o più semplicemente WS-TXM, basandosi su quanto definito in WS-CF e WS-CTX, definisce tre specifici modelli di coordinazione, ognuno indirizzato ad una differente tipologia di casi d’uso per relazioni business-to-business. Tuttavia, assume che altri modelli possano essere richiesti in futuro, e quando dovessero essere disponibili, si propone di incorporarli tra quelli già previsti.

I tre modelli presenti nella versione attuale della Specifica sono:

- **ACID Transactions**: un modello per transazioni tradizionali, progettato per l’interoperabilità con le infrastrutture transazionali esistenti.
- **Long Runnig Action (LRA)**: un modello per gestire un’Attività, o un gruppo di Attività, che non richiedano necessariamente le garanzie ACID. Una LRA possiede ancora la proprietà del “tutto o niente”, ovvero un *failure* non si traduce in un lavoro parziale, ma, i partecipanti, per assicurare l’Atomicità, possono utilizzare sia il recovery in avanti (compensazione) che quello all’indietro. L’Isolation è considerata pertanto una responsabilità della particolare Implementazione applicativa.
- **Business Process transaction (BP)**: per gestire un’Attività, o un gruppo di Attività, modellabili come un unico processo applicativo la cui struttura è costituita da una collezione di transazioni ACID o LRA, a seconda dei requisiti applicativi.

La Specifica, per mostrare le situazioni in cui possano essere utilizzati convenientemente i tre modelli sopra elencati, presenta anche alcuni scenari di casi d’uso. Per consentire al lettore una maggiore comprensione, li discuteremo alla fine del capitolo.

Utilizzo di WS-CF e WS-CTX

I tre modelli definiti da WS-TXM sono stati costruiti sulla base di WS-CTX e WS-CF. Definiscono infatti specifici Coordinator e CoordinationParticipant, ed estendono il CoordinationContext con informazioni proprie, per ottenere uno specifico Contesto transazionale. Ogni modello estende il concetto di Attività Coordinata per ottenere quello di Transazione, allo stesso modo con cui WS-CF deriva il concetto di CoordinatedActivity da quello di Attività.

In proposito, WS-TXM specifica:

- Una Transazione ha lo stesso tempo di vita dell'Attività a cui è associata.
- I partecipanti nei vari modelli transazionali sono istanze di CoorinationParticipant secondo quanto specificato in WS-CF. Tuttavia, essendo questo un requisito opzionale della specifica di WS-CF, il partecipante può essere definito anche in altro modo.
- un Servizio che voglia utilizzare uno dei protocolli definiti nell'ambito di WS-TXM per il ruolo di partecipante, deve sempre indicarne il corrispondente URI, nel messaggio di *addParticipant*, all'atto della registrazione con lo specifico servizio coordinatore.

Inoltre, come previsto in WS-CF, i coordinatori potranno arruolarsi come partecipanti presso altri coordinatori, e dunque comportarsi da subordinati. In questo modo, ad esempio, un coordinatore può rappresentare una serie di Attività locali al suo dominio, o può tradurre un protocollo transazionale "neutrale" in un protocollo transazionale specifico. WS-CTX, tuttavia, considera l'aspetto dell'Interposizione come opzionale, e non richiede che sia supportato da tutte le Implementazioni.

Nella descrizione dei tre modelli, come per tutte le altre Specifiche sinora trattate, anche per WS-TXM saranno forniti i "Modelli delle Interfacce", in cui saranno riassunti tutti i messaggi previsti dai vari protocolli che si andranno a discutere. Per semplicità, non sarà mostrato come tali messaggi derivino da quelli definiti in WS-CF: al livello di astrazione da noi adottato, del resto, questo tipo di informazione non è essenziale. Alcuni richiami ai messaggi definiti da WS-CF e specializzati in WS-TXM saranno comunque forniti nel corso della descrizione.

2.3.3.1 ACID Transactions

La possibilità di interoperare con i sistemi transazionali esistenti è una caratteristica di notevole importanza: molti di questi sistemi fungono attualmente da "dorsale" per applicazioni di livello *enterprise*, e si vorrà che continuino ad essere tali anche per applicazioni basate su WebServices. Tali sistemi, infatti, sono tipicamente coinvolti in attività business-to-business, direttamente o indirettamente, e ne costituiscono una parte essenziale. Pertanto, l'abilità a supportare contemporaneamente questi due ambienti, molto probabilmente, sarà uno degli aspetti chiave per il successo dei WebServices anche in ambito transazionale. Del resto, seppure le transazioni ACID non siano appropriate per tutti i domini applicativi, esistono delle particolari applicazioni in cui risultano insostituibili: ad esempio, per applicazioni finanziarie.

Per supportare questo tipo di applicazioni, gli autori di WS-TXM hanno progettato il modello transazionale oggetto del presente capitolo, specializzando il concetto di Attività Coordinata in quello di Transazione ACID (nel seguito semplicemente Transazione).

Utilizzo di WS-CF

Oltre alle considerazioni generali, precedentemente riportate nella sezione introduttiva e valide per tutti i protocolli definiti da WS-TXM, il modello ACID Transactions impone le seguenti restrizioni:

- Un partecipante non può ritirarsi autonomamente da una Transazione: una volta arruolato, dovrà rimanere vincolato fino al termine. Pertanto, non è lecito inviare un messaggio di *removeParticipant* al coordinatore. Se ciò dovesse accadere, il coordinatore dovrà ritornare un messaggio di errore *wrongState*.
- La fine di una Transazione induce automaticamente l'esecuzione dei protocolli di coordinamento. Non è lecito richiedere esplicitamente al coordinatore l'operazione di *coordinate*. In tal caso il coordinatore dovrà ritornare un messaggio di *notCoordinated*.

2.3.3.1.1 Modello Concettuale

Una ACID-Transaction è una particolare Attività Coordinata, e come tale è sempre legata ad un coordinatore: questo eseguirà automaticamente i protocolli di coordinamento previsti quando l'Attività sarà termina.

Il coordinatore, nello specifico, è detto ACID-Coordinator. Esso è legato al ContextService che gestisce l'Attività mediante una specifica ALS, denominata ACIDTxALS. Questa sarà informata dal ContextService circa l'inizio dell'Attività, nel qual caso potrà avviare un'istanza del coordinatore, e sarà informata circa la terminazione dell'Attività, informazione che passerà a sua volta al coordinatore creato, il quale potrà eseguire la coordinazione dei CoordinationParticipants.

In ogni caso, la relazione di controllo tra ACIDTxALS e ACID-Coordinator non è specificata da WS-TXM, e rimane una scelta implementativa: gli autori della Specifica, pur non esplicitamente, hanno assunto che le due entità in questione siano fornite normalmente dallo stesso soggetto.

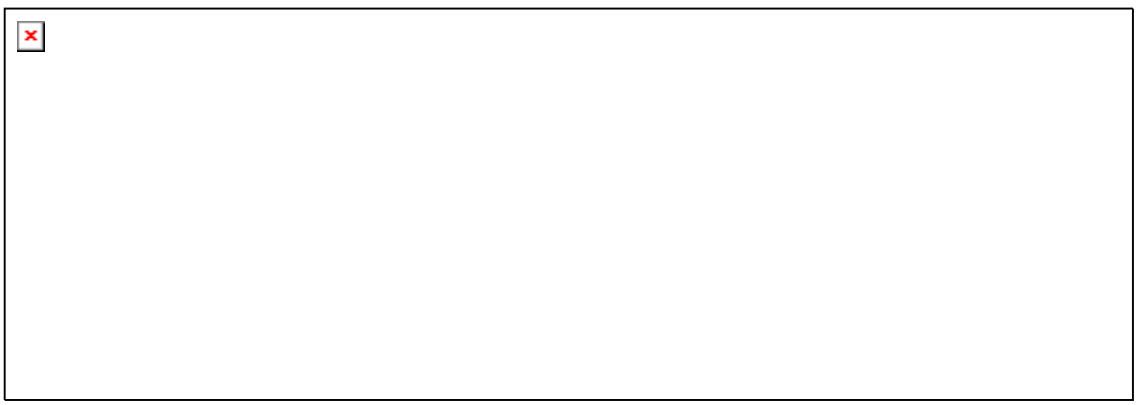

Figura 59 Modello Concettuale ACID-Transactions

Il modello utilizza un proprio Contesto, derivato dal CoordinationContext, che specializza solo nel nome (non aggiunge nuove informazioni). Prevede due protocolli, rispettivamente 2PC e Synchronization, entrambi costruiti specializzando le interfacce già definite da WS-CF.

2.3.3.1.2 Protocolli

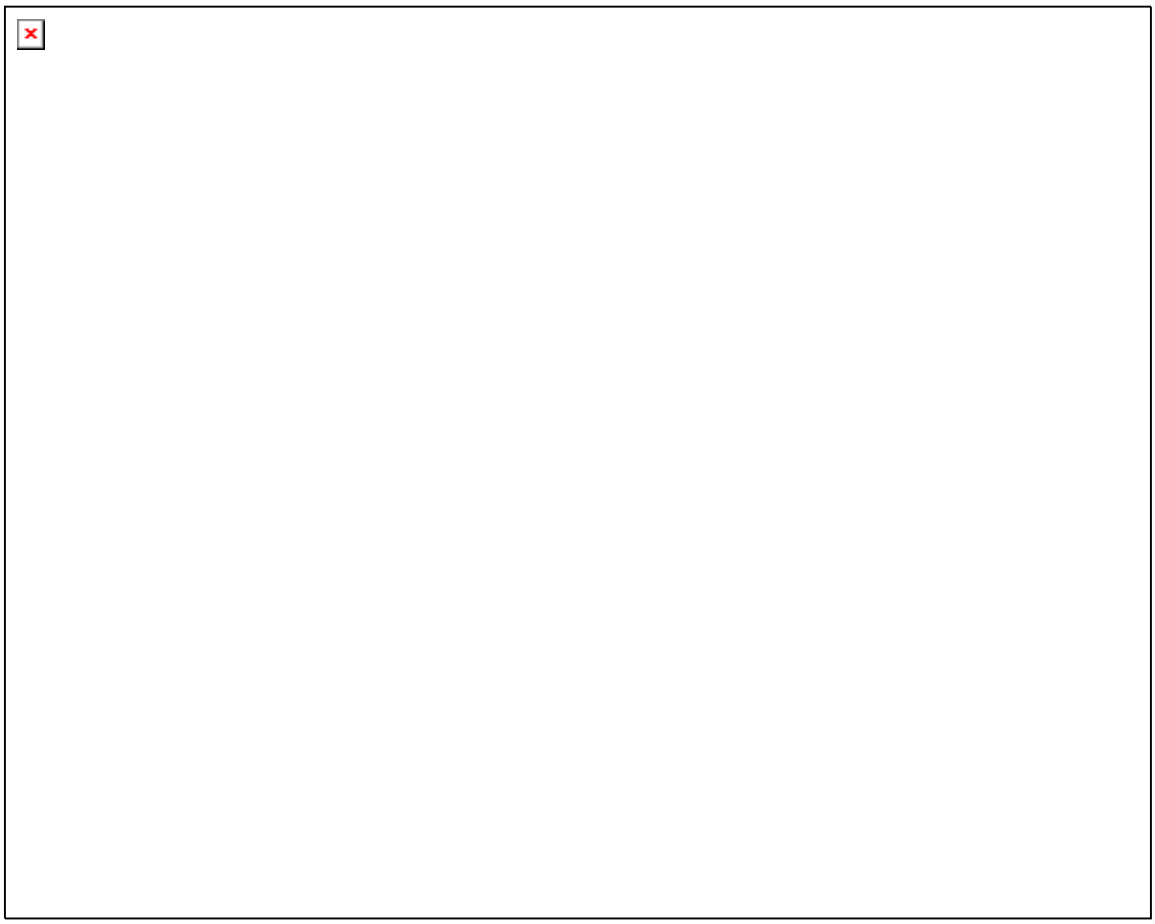

Figura 60 Modello delle Interfacce di ACID-Transactions

2.3.3.1.2.1 2PC

Il modello ACID-Transactions utilizza un protocollo 2PC tradizionale con alcune varianti ed ottimizzazioni. Le ottimizzazioni previste le abbiamo già incontrate e discusse nella descrizione delle precedenti Specifiche trattate. Viene adottato[110] infatti il modello dell' "Abort Presunto", che la Specifica chiama di "**Presumed rollback**"; l'ottimizzazione **One-phase**, che evita l'esecuzione della fase di Prepare quando la transazione coinvolga un solo partecipante; e l'ottimizzazione **Read-only**, che evita l'esecuzione della fase di Commit quando un'operazione sia di sola lettura.

Sono previste inoltre decisioni autonome dei partecipanti. Un partecipante che abbia passato con successo la fase di Prepare, è abilitato a prendere una decisione autonoma di Commit o Rollback. In tal caso la decisione deve essere ricordata localmente. Se la successiva comunicazione del coordinatore è contraria alla decisione già presa, si è in presenza di un risultato "non atomico", detto anche "heuristic" poiché conseguente ad una decisione euristica (previsione). I possibili risultati "heuristic" sono:

- **heuristic rollback**: l'operazione di Commit è fallita perché almeno un partecipante ha preso autonomamente la decisione di Rollback.
- **heuristic commit**: l'operazione di Rollback è fallita (ad esempio perché dopo il Prepare il coordinatore non è riuscito a scrivere sul suo log) poiché tutti i partecipanti hanno deciso autonomamente per il Commit.
- **heuristic mixed**: alcuni partecipanti hanno deciso autonomamente per il Commit mentre altri per il RollBack.
- **heuristic hazard**: non si conosce la decisione di alcuni partecipanti. Le decisioni note sono tuttavia tutte di Commit o di Rollback.

[110] Per quanto riguarda l'adozione del modello dell' "Abort Presunto", in realtà, essa viene solo citata in WS-TXM, in quanto in nessuna parte della Specifica vengono esplicitate le "conseguenze" di tale adozione. In proposito la Specifica si limita semplicemente a dire che il coordinatore della transazione non ha necessità di memorizzare informazioni persistentemente fino a quando non decida per il Commit, e null'altro.

Passando al significato dei messaggi previsti, e dunque dello stesso protocollo 2PC, esso è definito come segue:

- *prepare*: viene inviato dal coordinatore nella fase di Preparing. Il partecipante può rispondere con *voteReadOnly*, *voteCommit* o *voteRollback*. Se la risposta è un *voteCommit*, possono essere utilizzati dei Qualificatori aggiuntivi per fornire ulteriori informazioni. Se il partecipante funge da coordinatore subordinato, e non riesce a determinare lo stato di qualche suo partecipante, deve rispondere con un messaggio di *heuristicHazardFault*; se invece verifica che alcuni suoi partecipanti hanno già votato per il Commit e altri per il Rollback, dovrà rispondere con un messaggio di *heuristicMixedFault*.
- *rollback*: il coordinatore si trova nella fase di Cancelling. Se il partecipante riceve questo messaggio dopo un *prepare*, qualsiasi errore che dovesse occorrere a questo punto causerebbe un'euristica. Se il partecipante funge da coordinatore subordinato, e non riesce a determinare lo stato di qualche suo partecipante, deve rispondere con un messaggio di *heuristicHazardFault*; se invece verifica che alcuni suoi partecipanti hanno già votato per il Commit e altri per il Rollback, dovrà rispondere con un messaggio di *heuristicMixedFault*. Se il partecipante effettua il Commit piuttosto che il Rollback, dovrà inviare un messaggio di *heuristicCommitFault*. In tutti gli altri casi, il partecipante potrà inviare un messaggio di riscontro *rolledback*.
- *commit*: il coordinatore è nella fase di Confirming. Se dopo la ricezione di questo messaggio il partecipante dovesse incontrare qualche errore, si otterrebbe un'euristica. Se il partecipante funge da coordinatore subordinato, e non riesce a determinare lo stato di qualche suo partecipante, deve rispondere con un messaggio di *heuristicHazardFault*; se invece verifica che alcuni suoi partecipanti hanno già votato per il Commit e altri per il Rollback, dovrà rispondere con un messaggio di *heuristicMixedFault*. Se il partecipante effettua il Rollback piuttosto che il Commit, dovrà inviare un messaggio di *heuristicRollbackFault*. In tutti gli altri casi, il partecipante potrà inviare un messaggio di riscontro *committed*.

- *onePhaseCommit*: se il partecipante funge da coordinatore subordinato, e non riesce a determinare lo stato di qualche suo partecipante, deve rispondere con un messaggio di *heuristicHazardFault*; se invece verifica che alcuni suoi partecipanti hanno già votato per il Commit e altri per il Rollback, dovrà rispondere con un messaggio di *heuristicMixedFault*. Se il partecipante effettua il Rollback piuttosto che il Commit, dovrà inviare un messaggio di *heuristicRollbackFault*. In tutti gli altri casi, il partecipante potrà inviare un messaggio di riscontro di *commtted* o di *rolledback*.
- *forgetHeuristic*: il partecipante riceve questo messaggio quando, in seguito ad un messaggio di *prepare*, abbia preso una decisione autonoma contraria a quella del coordinatore. Quando un partecipante prende una decisione autonoma, deve sempre ricordarla fino alla ricezione del messaggio di *forgetHeuristic*, per essere sicuro che il coordinatore sia consapevole della situazione. A questo messaggio il partecipante risponderà con un messaggio di riscontro[111] *heuristicForgotten*.

Dal significato dei messaggi sopra esposto, si deduce facilmente che il protocollo differisce dal 2PC tradizionale solo per quanto riguarda le decisioni autonome dei partecipanti.

In particolare, va ricordato che un partecipante che prenda una decisione autonoma, deve memorizzarla persistentemente. Quando riceva la decisione del coordinatore, in caso sia contraria a quella già presa localmente, il partecipante dovrà rispondere con un messaggio di "heuristic" (a seconda dei casi). A questo punto dovrà ancora attendere che il coordinatore ritorni un messaggio di *forgetHeuristic* (dimentica), con cui gli comunica che è consapevole del problema.

[111] Non viene specificato per quanto tempo il coordinatore dovrà attendere il messaggio di *heuristicForgotten*: è compito dell'Implementazione definire, eventualmente, Qualificatori in proposito.

Comportamento del coordinatore e del partecipante

La figura seguente mostra le transizioni di stato del coordinatore per il protocollo 2PC. Per un partecipante vale lo stesso diagramma.

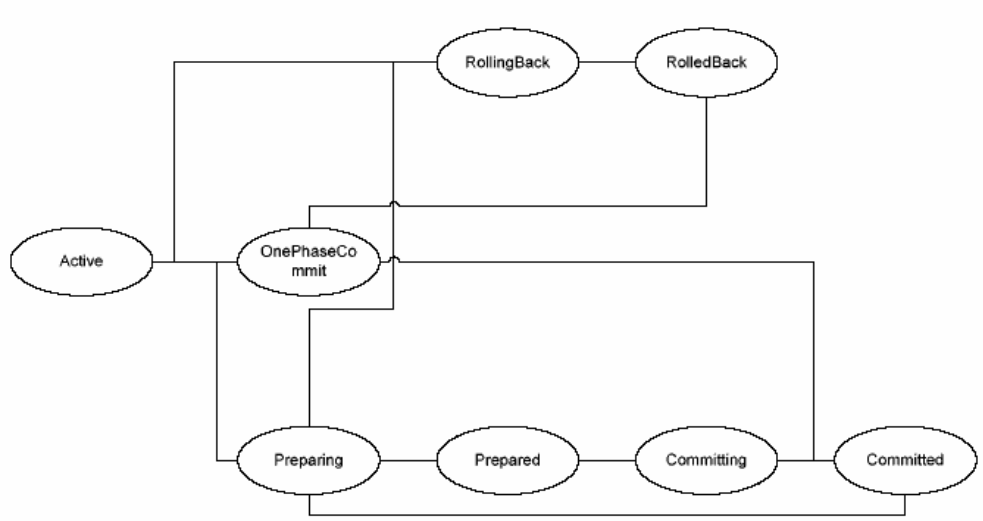

Figura 61 StateDiagram per un coordinatore o un partecipante 2PC.
Le transizioni di stato vanno lette da sinistra verso destra, senza possibilità di ritorno
- tratta da Riferimenti[10 p17] -

La Transazione inizia quando viene inviato un messaggio di *begin* dall'ActivityService all'ALS associata al protocollo ACID-transactions (ACIDTxALS), la quale crea il corrispondente coordinatore. Il coordinatore parte dallo stato di Active, e possiede un *tempo di vita* pari a quello della Transazione associata. Questo significa che se scade il tempo di vita della Transazione, come conseguenza, l'ActivityService la porterà al completamento, e il coordinatore eseguirà la Coordinazione concordemente con lo stato di terminazione raggiunto.

Se la Transazione viene portata a conclusione nello stato di Success, allora l'ActivityService invierà il corrispondente messaggio di *completeWithStatus* all'ACIDTxALS, la quale, di conseguenza, tenterà per il Commit della Transazione.

Se con la transazione è stato arruolato un solo partecipante, il coordinatore inizierà il "protocollo" OnePhaseCommit, e transiterà nello stato di RolledBack o Committed, a seconda della decisione restituita dal partecipante.

Se invece ci sono più partecipanti, il coordinatore passerà alla fase di Preparing, e invierà un messaggio di *prepare* a tutti i partecipanti. Se tutti i partecipanti dovessero essere "read-only", la transazione è completata, e il coordinatore transiterà nello stato di Committed, mentre il "completition status" dell'Attività associata sarà Success. Alternativamente, qualsiasi messaggio di errore ricevuto da un partecipante, o l'indicazione di un partecipante non preparato, causerà la transizione del coordinatore nello stato di RollingBack, nel quale invierà un messaggio di *rollback* a tutti i partecipanti. Subito dopo, esso transiterà nello stato di RolledBack, mentre il "completition status" dell'Attività sarà Failure.

Assumendo che tutti i partecipanti abbiano "preparato" con successo, il coordinatore entrerà nello stato di Prepared, nel quale prenderà la decisione finale. In tal caso dovrà ricordare sufficienti informazioni, in maniera persistente, per assicurare che la decisione possa essere condotta a termine anche in caso di *failures*. Quando il coordinatore avvierà la seconda fase del protocollo, passerà nello stato di Committing, in cui raccoglierà i vari riscontri di *committed*, e in seguito transiterà nello stato di Committed.

Estensione del messaggio di "status" di WS-CF

Alla richiesta di *getStatus* indirizzata ad un coordinatore (come previsto dal ruolo ClientCoordinator definito in WS-CF) o ad un partecipante (come previsto dal ruolo Participant definito in WS-CF), questi possono rispondere con un messaggio di *status*, indicante il proprio stato corrente. Oltre agli stati già definiti in WS-CF, un coordinatore, o un partecipante del protocollo 2PC, può indicare uno tra i valori di stato definiti da WS-TXM:

- RollbackOnly: indica che il coordinatore o il partecipante è in grado di eseguire solo il Rollback, qualunque sia la richiesta;
- RollingBack;
- RolledBack: è uno stato transiente, poiché il protocollo 2PC utilizza l'ottimizzazione dell' "Abort Presunto". Dopo tale transitorio sarà restituito lo stato di NoActivity;
- Committing;
- Committed;

- HeuristicRollback: tutti i partecipanti hanno effettuato il Rollback, mentre era stato richiesto di effettuare il Commit;
- HeuristicCommit: tutti i partecipanti hanno effettuato il Commit, mentre era stato richiesto di effettuare il Rollback;
- HeuristicHazard: il risultato di alcuni partecipanti non è noto;
- HeuristicMixed; alcuni partecipanti hanno effettuato il Commit e altri il Rollback, ma tutti i risultati sono noti;
- Preparing;
- Prepared;

In Figura 62 si può vedere un esempio di interazioni conformi con il protocollo 2PC appena definito. Tale diagramma, per semplicità, non mostra le interazioni con il ContextService e con la ACIDTxALS, supponendo tali entità inglobate nell'ACIDCoordinator. Il lettore noti l'utilizzo della "modalità Pull", scelta per l'arruolamento del partecipante, che avviene in seguito alla ricezione del messaggio applicativo, il quale, contenendo il Contesto della transazione, fornisce il necessario riferimento al coordinatore.

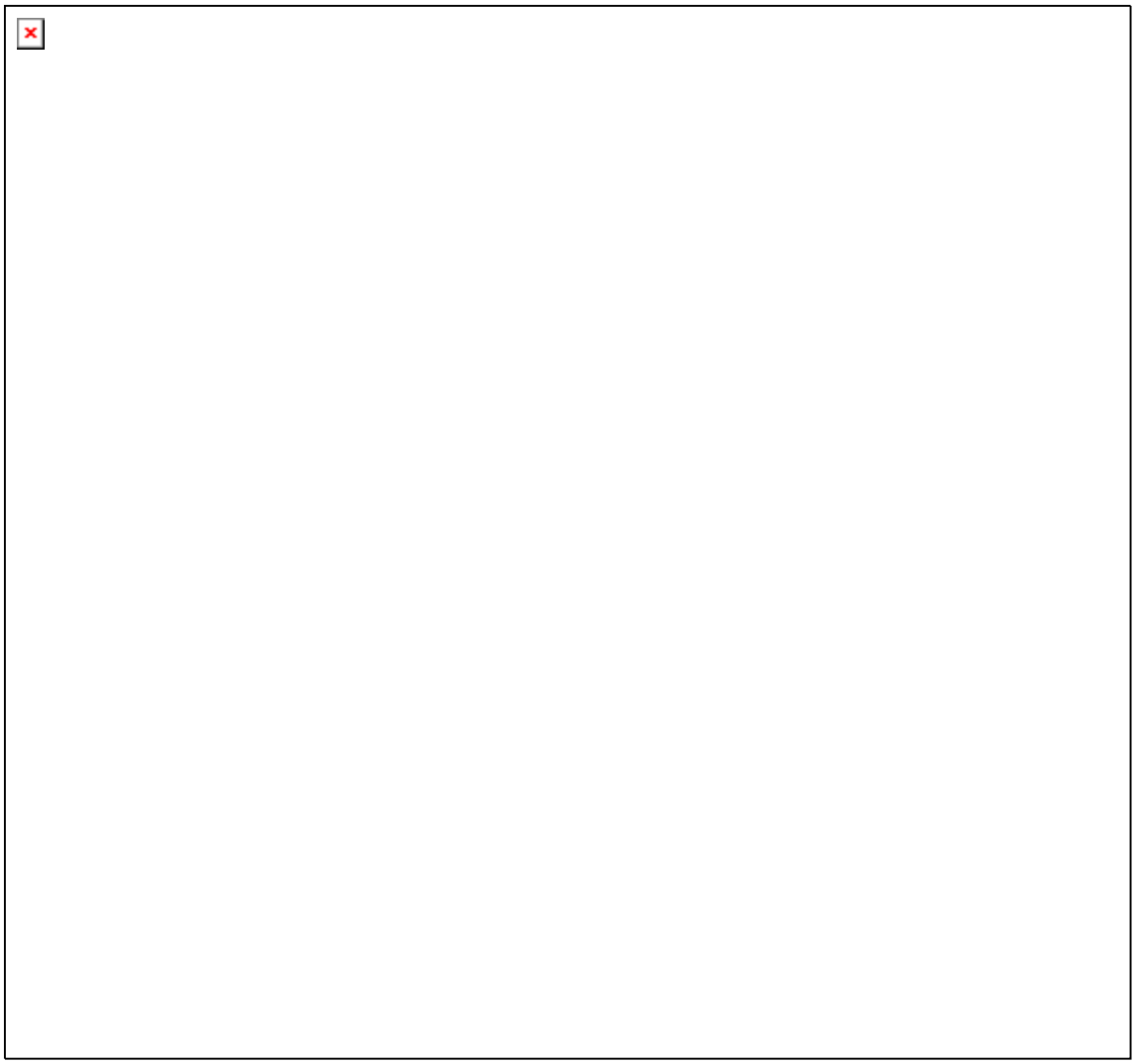

Figura 62 Esempio di interazioni conformi al modello ACID-Transactions

2.3.3.1.2.2 Synchronization

Un partecipante può registrarsi per questo protocollo se non vuole prendere parte al 2PC, ma vuole essere informato quando esso inizi, e quando sia stato completato.

Il protocollo in questione permette tuttavia che un partecipante di questo tipo possa influenzare la decisione del coordinatore. Se infatti quest'ultimo dovesse ricevere un messaggio di errore da un partecipante Synchronization, dovrà prendere una decisione globale di Rollback.

I messaggi inviati dal coordinatore sono:

- *beforeCompletion*: l'esecuzione del protocollo 2PC sta per essere avviata. Indica pure lo stato di avvio, "committing" o " rollingback". Se lo stato è di "committing", e se il partecipante non ha problemi, quest'ultimo dovrà restituire un messaggio di *success*.
- *afterCompletion*: l'esecuzione del protocollo 2PC è stata completata. Indica pure lo stato di terminazione, "committed" o "rolledback".

Nella figura seguente sono mostrate le transizioni di stato del coordinatore, nel caso in cui l'esecuzione del protocollo 2PC sia avviata per il Commit. In caso contrario, il protocollo prevede che al partecipante Synchronization sia inviato solo il messaggio *afterCompletion*.

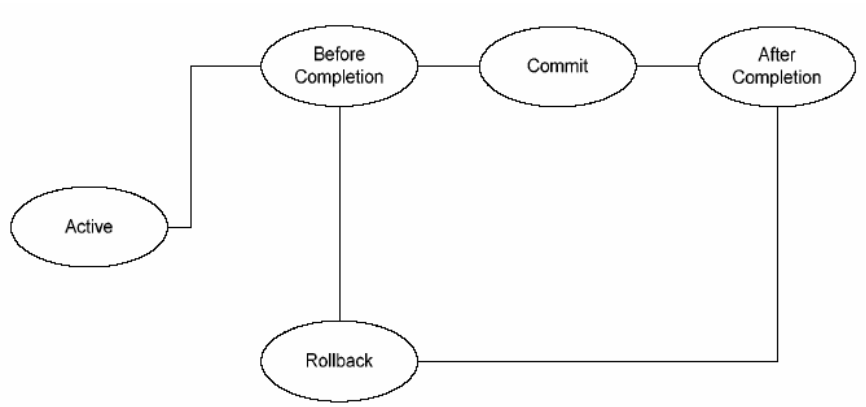

Figura 63 Transizioni di stato per un Coordinatore che esegue il protocollo Synchronization.
Le transizioni di stato vanno lette da sinistra verso destra, senza possibilità di ritorno.
- tratta da Riferimenti[10 p24] -

Dallo stato di Active il coordinatore passa nello stato BeforeCompletion, e invia un messaggio omonimo a tutti i partecipanti Synchronization arruolati. Ogni errore ricevuto in questa fase da un “Synchronization” forza il Roolback della Transazione. Assumendo che non siano restituiti errori, il coordinatore passa nello stato di Commit, esegue il protocollo 2PC, e una volta completato, passa nello stato AfterCompletition, in cui invia un messaggio omonimo a tutti i partecipanti Synchronization. Il verificarsi di un errore a questo punto non ha più effetto sul risultato della Transazione, e deve essere gestito dall’Implementazione.

2.3.3.1.3 Recovery Handling e Interposizione

Essendo WS-TXM costruito sopra WS-CF, è possibile l’Interposizione: ogni partecipante può dunque essere un coordinatore subordinato.

WS-TXM specifica che ogni partecipante semplice, o coordinatore subordinato, è responsabile affinché ricordi sufficienti informazioni per completare una transazione anche in caso di fallimenti. Tuttavia richiede esplicitamente che un partecipante memorizzi una decisione autonoma fino a quando non riceva il messaggio di *forgetHeuristic* dal coordinatore. Inoltre, come già osservato nella nota 110, le informazioni da memorizzare relativamente al modello di “Abort Presunto” non vengono esplicitate.

Per il recovery possono venire in aiuto i meccanismi definiti in WS-CF, che permettono di determinare lo stato corrente dei partecipanti e della Transazione.

WS-TXM assume inoltre che il recovery non sia sempre facilmente automatizzabile, e dunque che l’intervento umano possa essere richiesto come ordinaria amministrazione. Nel caso in cui il recovery non sia possibile del tutto, la Specifica precisa che tale condizione deve essere “loggata” e/o indicata all’amministratore di sistema, affinché questi possa prendere gli opportuni provvedimenti.

2.3.3.2 Long Running Action (LRA)

Gli autori di WS-TXM ritengono che il modello per transazioni ACID di tipo tradizionale non sia adatto per tutti i tipi di applicazioni. In particolare, ritengono che sia inadatto per molte applicazioni di tipo business-to-business per i seguenti motivi:

a) Molte applicazioni business-to-business richiedono un supporto transazionale che garantisca risultati consistenti, ma più spesso richiedono anche computazioni di lunga durata, tra sistemi lascamente accoppiati e tra componenti che non condividono dati, locazione o amministrazione. Risulta pertanto difficile utilizzare le transazioni ACID tradizionali. Un'applicazione di lunga durata, ad esempio, è quella di un sistema di prenotazione di un posto in aereo: il servizio dell'agenzia può riservare un posto ad un individuo per un certo periodo di tempo, ma, se nel frattempo l'individuo non dovesse confermare la prenotazione, la riserva non risulterebbe ulteriormente valida.
b) I meccanismi disponibili con i sistemi transazionali tradizionali consentono una composizione di transazioni sequenziale e concorrente. Sono sufficienti se l'applicazione può essere rappresentata come una singola transazione principale, ma, frequentemente, questo non è il caso per i servizi Web.
c) Le transazioni tradizionali sono meglio concepite per essere di corta durata, e per apportare cambiamenti stabili sullo stato dei sistemi. Esse sono meno bene concepite per strutturare funzioni applicative di lunga durata, che normalmente possono durare minuti, ore, ed anche giorni. Infatti, l'utilizzo del "locking" per un così lungo periodo, potrebbe ridurre la concorrenza a livelli inaccettabili.
d) Nel modello tradizionale, se una transazione di lunga durata richiedesse il Rollback, sarebbe necessario cancellare, con operazioni di undò, una grande quantità di lavoro già effettuato.[112]

Inoltre, ci sono certe classi di applicazioni[113] per cui è noto che le risorse acquisite all'interno di una transazione possono essere rilasciate "prima", piuttosto che alla fine

[112] Sarebbe invece preferibile un modello che tenga conto della lunga durata di una transazione business, ad esempio, supportando il recovery anche nella fase di Composizione (vedi requisito "RC5 Recovery anche nella fase di Composizione").

della transazione. Per questo tipo di applicazioni, precisa WS-TXM, nel caso in cui la transazione vada in Abort, per riportare il sistema ad uno stato consistente potrebbero essere più opportune delle "attività di compensazione".

Sulla base delle precedenti considerazioni, WS-TXM introduce il modello LRA, che rivolge ad applicazioni che richiedono:

- interazioni di lunga durata;
- Commit "negoziabile" a run-time;
- rilassamento della proprietà di Isolamento.

Il modello non assicura implicitamente l'Atomicità, ma definisce le condizioni e le modalità con cui le azioni di compensazione possono essere invocate per cercare di ottenerla, o quanto meno per cercare di ripristinare una qualche forma di consistenza. La compensazione, che potrà essere di "recovery in avanti o all'indietro", sarà tipicamente specifica per l'applicazione, e potrebbe essere non necessaria del tutto. In altri termini, come i servizi facciano il loro lavoro e come questo possa essere compensato, è un problema applicativo. Anche problemi cruciali come l'Isolation e la Durabiliry sono scelte implementative.

Utilizzo di WS-CF

Oltre alle considerazioni generali precedentemente descritte nella sezione introduttiva di WS-TXM, e valide per tutti i protocolli definiti dalla Specifica, il modello LRA impone la seguente restrizione:

- Non è lecito richiedere al Coordinator l'operazione di *coordinate*. In tal caso il Coordinator dovrà rispondere con un messaggio di *notCoordinated*.

Questa restrizione è analoga a quella imposta al modello ACID-Transactions. Si noti, invece, che a differenza di quanto accade per un "partecipante ACID", un partecipante LRA (Compensator) non è vincolato alla transazione, e può ritirarsi da una LRA, in ogni momento prima che questa sia conclusa, semplicemente inviando un messaggio di *removeParticipant* al coordinatore.

[113] La Specifica non fa esempi in proposito.

2.3.3.2.1 Modello Concettuale

Una LRA è una particolare Attività Coordinata, e come tale è sempre legata ad un coordinatore: analogamente a quanto accade per una ACID-Transaction, questo significa che quando l'Attività termina il protocollo LRA, che descriveremo in seguito, viene automaticamente eseguito, sia che i lavori già effettuati debbano essere accettati, siano che debbano essere compensati.

Il coordinatore per una LRA è detto LRA-Coordinator. Questo è legato al ContextService che gestisce l'Attività mediante un'apposita ALS.

I partecipanti in una Coordinazione LRA sono detti Compensator, poiché, appunto, vengono invocati nel caso in cui siano necessarie operazioni di compensazione. Riprendendo l'esempio sul sistema di prenotazione di un posto in aereo, alla richiesta di un utente l'agenzia potrebbe rispondere con un approccio ottimistico, ovvero riservando il posto e sottraendo il costo dal conto dell'utente; nel caso in cui l'utente disdica la prenotazione, l'operazione di compensazione potrebbe corrispondere alla cancellazione della prenotazione e al rimborso del cliente.

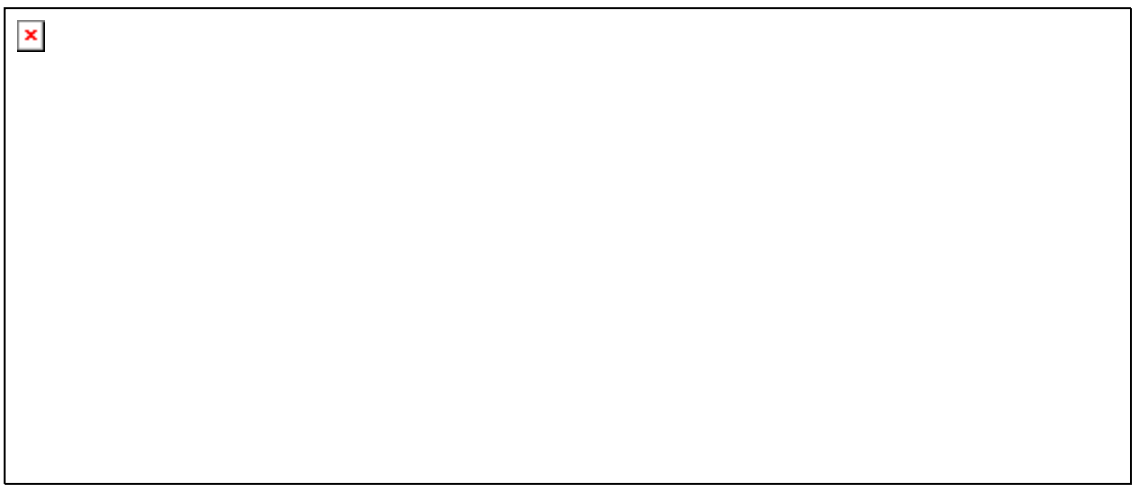

Figura 64 Modello Concettuale LRA

Quando un Servizio effettua del lavoro all'interno di una LRA, e tale lavoro potrebbe dover essere compensato in seguito, arruola un Compensator come partecipante del LRA-Coordinator associato. Il Coordinatore invocherà il Compensator al termine

dell'Attività, indicandogli se dovrà compensare (l'Attività è stata completata con Fail) o meno (l'Attività è stata completata con Success).

Le eventuali operazioni di compensazione potrebbero essere delle operazioni che effettuano l'*undò* del lavoro già eseguito dal servizio Web come parte dell'Attività, oppure potrebbero essere delle operazioni di "recupero", dovute al fatto che tale servizio non è stato in grado di eseguire il lavoro richiesto.

Eventualmente, per effettuare l'operazione di compensazione, il Compensator potrebbe richiamare a sua volta altre LRA, aventi anch'esse dei compensatori. In proposito WS-TXM fornisce un esempio che qui di seguito riproponiamo.

Consideriamo un'applicazione per la prenotazione di un viaggio, strutturata come nella figura seguente:

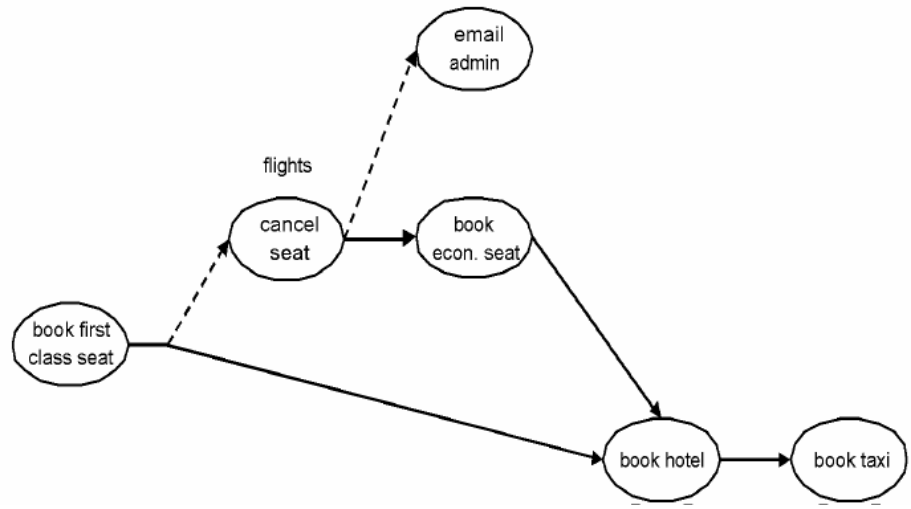

Figura 65 Esempio di un'applicazione con molteplici Attività di compensazione.
- tratta da Riferimenti[10 p29] -

Se l'Attività di prenotazione di un posto in prima classe dovesse fallire, sarebbe avviata l'Attività di compensazione "cancel seat". Se anche questa dovesse fallire, sarebbe avviata un'ulteriore Attività di compensazione per avvisare l'amministratore tramite email. Se invece l'annullamento della prenotazione avesse successo, inizierebbe un'altra Attività, con l'obiettivo di prenotare un posto di classe economica. Nel caso si riesca ad ottenere un posto, in prima o in seconda classe, infine, sarebbero avviate le Attività per la prenotazione dell'hotel e del taxi.

La capacità di compensazione, per un'Attività business automatizzata, come per una qualsiasi altra attività business, potrebbe essere, in generale, una capacità solo transiente. Per quanto riguarda LRAs innestate, invece, WS-TXM richiede che il lavoro effettuato al loro interno rimanga compensabile fino al completamento dell'Attività "padre"[114].

A riguardo consideriamo lo scenario "Arranging a Night-Out" precedentemente trattato[115], e "mappiamo" le varie attività con le LRAs mostrate in Figura 66. Quando LRA1 sarà conclusa con successo, i Compensators registrati con il coordinatore di LRA1 saranno notificati con un messaggio di *complete* (secondo il protocollo LRA), in seguito al quale potranno scegliere[116] di arruolarsi con il coordinatore relativo ad LRA5, che potrà invocarle nel caso in cui l'intera Attività debba concludersi con un Abort. Solo dopo che LRA4 sia terminata, le singole LRAs saranno avvisate dal coordinatore dell'LRA principale (LRA5) su cosa fare (compensare o meno), ma, fino a quel momento, tutte, LRA1 compresa, dovranno rimanere compensabili.

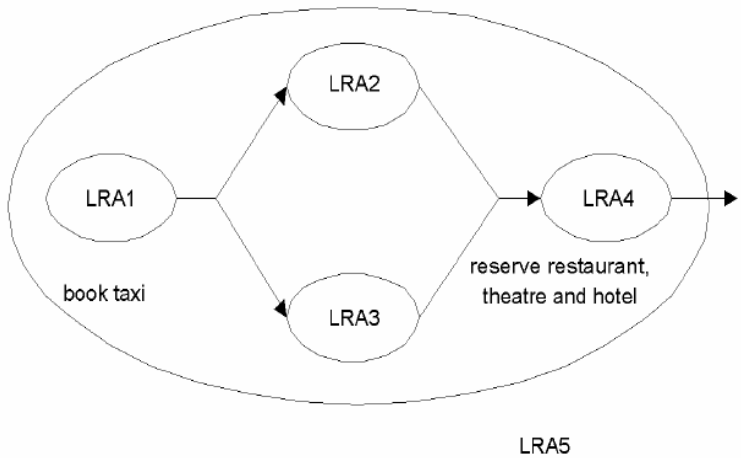

Figura 66 Esempio di LRAs innestate.
- tratta da Riferimenti[10 p27] -

[114] WS-TXM non è chiara in proposito, poichè richiede che il lavoro effettuato all'interno di Attività innestate rimanga compensabile fino a quando l' "Attività padre" le informi che non sia più *necessario*. Ma non specifica come questa "non ulteriore necessità" sia comunicata ai partecipanti, e l'esempio non è chiarificante, in quanto, nella versione originale, riporta: <<Quando LRA1 sia stata eseguita con successo, le relative operazioni di compensazione potranno essere "passate" ad LRA5, che potrà invocarle nel caso in cui l'intera Attività debba concludersi con un Abort>>.
La nostra è solo una possibile interpretazione, ma probabilmente è quella più verosimile: in proposito si ricordi che il "Protocollo di Arruolamento" definito da WS-CF prevede il messaggio di *getParentCoordinator*.
[115] Lo scenario è stato trattato a pag.41.
[116] La Specifica assume che questa sia una scelta implementativa.

Il modello LRA specializza il CoordinationContext nell'LRA-Context, aggiungendo le seguenti informazioni:

- un identificativo univoco per l'LRA;
- una lista di eventuali coordinatori subordinati.

Poiché, in generale, un servizio Web può essere o meno compensabile, è compito dell'utente utilizzare l'LRA-Context per richiedere che tutti i servizi invocati all'interno dello *scope* dell'LRA siano compensabili: può essere richiesto utilizzando l'attributo "mustUnderstand", previsto da SOAP, in maniera appropriata. Ovviamente, miscelando servizi compensabili e non, l'utente otterrà un'Attività business che, complessivamente, non sarà compensabile mediante il modello LRA.

2.3.3.2.2 Protocollo LRA

Figura 67 Modello delle Interfacce LRA

Il protocollo è molto semplice. Esso prevede, al termine di una LRA, che ogni compensatore registrato sia invocato dal coordinatore con uno dei seguenti messaggi:

- *complete*: l'LRA è stata completata con Success. Se l'LRA è innestata, allora i Compensators con essa arruolati possono arruolare[117] se stessi (o nuovi Compensators) con l'LRA "padre", altrimenti possono effettuare le necessarie operazioni di *cleanup*.
- *compensate*: l'LRA è stata completa con Fail. Tutti i compensatori registrati con l'LRA saranno invocati in ordine inverso a quello con cui sono stati arruolati. Il coordinatore potrà dimenticare le informazioni dei compensatori che indicheranno di aver operato correttamente inviando un messaggio di riscontro *compensated*, mentre per gli altri potrà ritentare l'invocazione (possibilmente dopo un periodo di tempo) oppure potrà "loggare" una "violazione di compensazione". Un compensatore che non possa compensare deve mantenere le sue informazioni fino a

[117] La Specifica utilizza il termine "propagare" al posto del termine "arruolare" da noi utilizzato, e non precisa il significato di questa "propagazione", nè fornisce esempi da cui possa essere dedotto in maniera univoca. Abbiamo già incontrato il problema sulla stessa questione (vedi nota 114 a pag.240).

quando non riceva il messaggio di *forget* (dimentica), indicante che il coordinatore è consapevole del problema.

Nella figura seguente è mostrato un semplice Sequence Diagram, in cui, come al solito, si suppone che sia coinvolto un solo partecipante. Nel diagramma, per semplicità, si suppone pure che l'LRA-Coordinator contenga in sè anche il ContextService e l'LRA-ALS, cosicché non vengono mostrate le interazioni con tali entità.

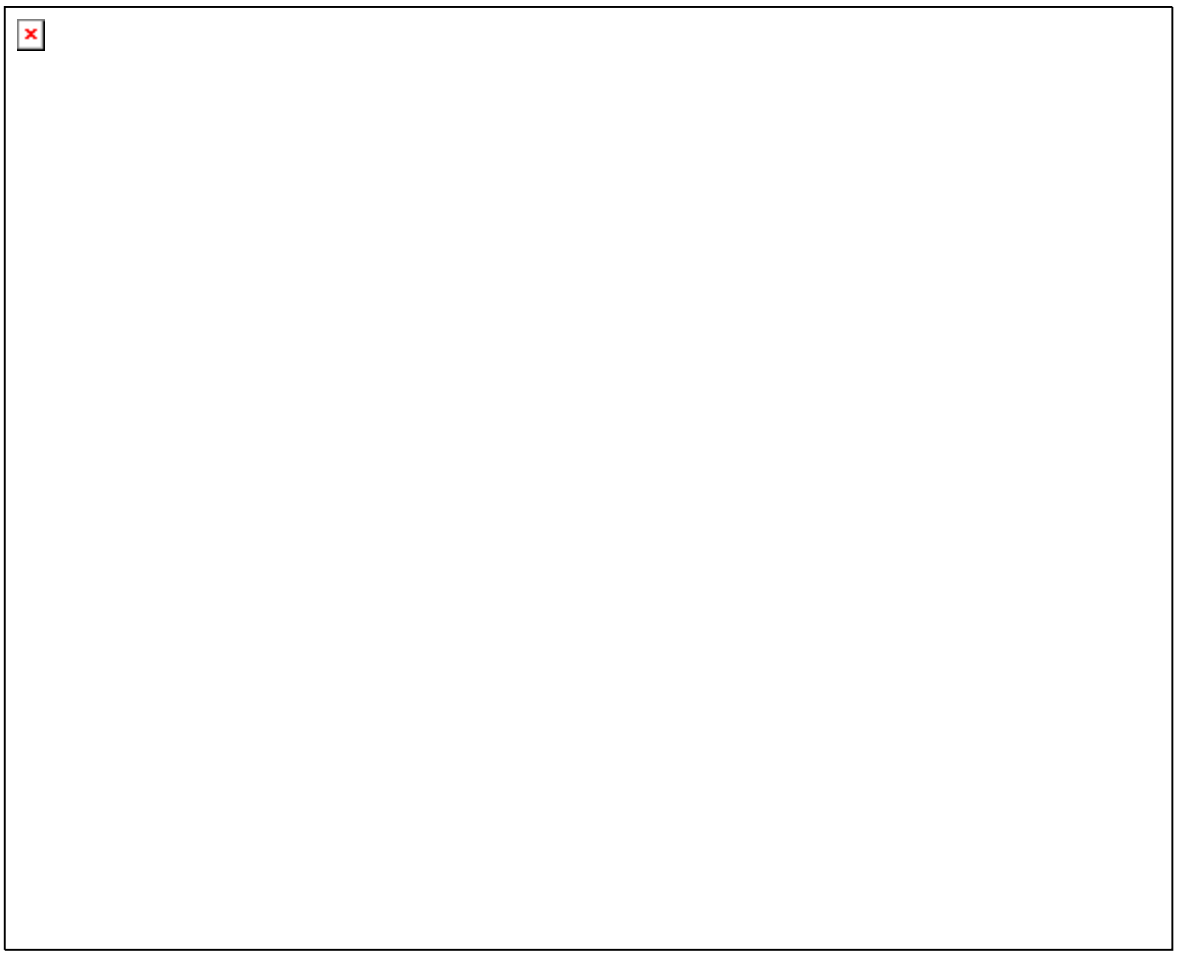

Figura 68 Esempio di interazioni conformi LRA

Composizione Dinamica delle Attività

WS-TXM fornisce un esempio di Composizione Dinamica, in cui l'applicazione consiste nella prenotazione di alcuni servizi dalle compagnie che offrono i costi minori.

Nell'esempio si suppone che l'utente voglia prenotare un taxi, un volo aereo, e un'assicurazione sul volo. Assumendo che ogni operazione di prenotazione sia eseguita

all'interno di una specifica LRA, l'applicazione può essere schematizzata come in Figura 69.

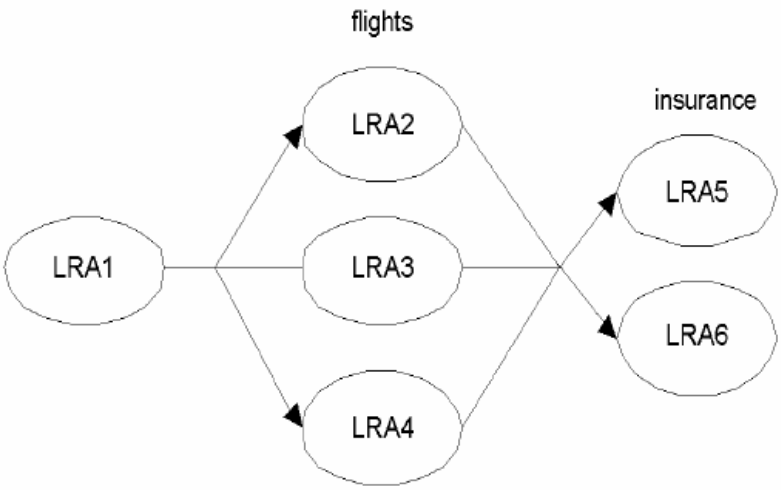

Figura 69 Esempio di composizione dinamica mediante il modello LRA
- tratta da Riferimenti[10 p34] -

L'utente vuole per prima cosa prenotare un taxi per l'aeroporto (LRA1). In seguito vuole contattare tre differenti compagnie aeree per cercare l'offerta migliore. Avvia dunque LRA2 con la prima compagnia, e con essa prenota un posto a 150€. Mentre LRA2 è ancora attiva, avvia LRA3 con la seconda compagnia, e ottiene una prenotazione per 160€, che dunque viene subito annullata (richiedendo la fine dell'LRA con Fail). Infine, l'utente avvia LRA4 con la terza compagnia, ottenendo un offerta per 120€, così annulla LRA2 mentre conferma LRA4. La stessa tecnica viene utilizzata in seguito per scegliere la compagnia assicuratrice più conveniente (LRA5 e LRA6).

In generale, un'applicazione può utilizzare diverse LRAs, sequenzialmente o concorrentemente, a seconda delle necessità. L'applicazione potrebbe essere strutturata in maniera tale che, mentre alcune LRAs siano ancora nello stato di Active, essa richieda lavori nell'ambito di altre LRAs. Dal risultato di queste ultime, l'applicazione potrebbe decidere, tra le prime, quali confermare e quali compensare. Così, se ogni LRA corrispondesse ad una certa unità di lavoro, l'applicazione potrebbe scegliere in tempo reale quali lavori confermare.

Il modello LRA supporta dunque la Composizione Dinamica delle Attività. Si noti che lo stesso discorso non può essere fatto relativamente ad una singola LRA, come accade

per una BusinessActivity in WS-Transactions, poiché un singolo LRA-Coordinator dovrà trattare tutti i compensatori arruolati allo stesso modo (*complete* o *compensate*).

Estensione del messaggio di "status" di WS-CF

Oltre ai valori di stato definiti da WS-CF, alle richieste di *getStatus* un Compensator può rispondere con gli ulteriori valori di seguito elencati:

- Compensating;
- Compensated: compensazione avvenuta con successo.
- FailedToCompensate: il compensatore non è riuscito a compensare. Esso rimane in questo strato, in cui deve mantenere informazioni circa il lavoro che doveva compensare, fino a quando non riceve un messaggio di "forget" dal coordinatore.
- Completing;
- Completed;
- FailedToComplete;

Si ricorda che ad un messaggio di *getStatus*, come previsto da WS-CF per il ruolo di Participant, quest'ultimo può rispondere con un messaggio di *status* indicante il proprio stato corrente.

Definizione di Qualificatori

Quando un compensatore viene arruolato con una LRA, l'entità che effettua l'arruolamento può fornire una serie di Qualificatori, che possono essere usati dal coordinatore e dall'applicazione business per influenzare il risultato finale dell'Attività in questione. Uno di questi è definito da WS-TXM, e corrisponde al Qualificatore **TimeLimit**: esprime il tempo massimo, in secondi, in cui il compensatore garantisce che potrà compensare il lavoro fatto dal servizio associato. Scaduto tale tempo, per compensare potrebbe essere necessario avviare altre Attività, oppure potrebbe essere necessario l'intervento di un operatore umano, a seconda di come è strutturata l'applicazione.

2.3.3.2.3 Recovery Handling e Interposizione

Il modello LRA usa un protocollo definito come “presumed nothing”: i compensatori non effettuano operazioni per default, ma attendono sempre comunicazioni dal coordinatore. Pertanto, ogni volta che un compensatore viene arruolato con una LRA, il coordinatore deve prendere informazioni a sufficienza su di esso, da mantenere persistentemente, in maniera tale che, anche in caso di fallimenti, il compensatore possa essere contattato quando l’LRA sia termina.

Ad ogni servizio si richiede invece di memorizzare un numero sufficiente di informazioni affinché la compensazione sia possibile, e, nel caso in cui non riesca a compensare, che memorizzi tale evento fino alla ricezione del messaggio di *forget*.

I partecipanti o i coordinatori, per il recovery, possono utilizzare i meccanismi definiti da WS-CF, per determinare lo stato corrente dell’LRA ed agire di conseguenza. In ogni caso è sempre possibile che il recovery non possa essere effettuato. Tale evento, secondo WS-TXM, deve essere “loggato”, in qualche maniera, e posto all’attenzione dell’amministratore. In generale, in un ambiente su larga scala e in presenza di *failures* di lunga durata, la Specifica assume che il recovery possa non essere automatizzabile, e che, per riportare l’applicazione alla consistenza, possa essere essenziale l’intervento umano.

Essendo WS-TXM costruito sopra WS-CF, l’Interposizione è ammessa: così dei compensatori possono anche essere dei coordinatori subordinati, ad esempio per incrementare le performance, o per federare un ambiente multi-dominio in domini separati. Evidentemente, quanto detto per un compensatore e per un coordinatore, vale anche per un coordinatore subordinato.

2.3.3.3 Business Process transaction (BP)

Per supportare relazioni multi-dominio tra servizi Web, su larga scala, e di lunga durata, WS-TXM fornisce il modello Business Process. Questo definisce tutta una serie di protocolli, adatti sia per interazioni sincrone che per interazioni asincrone (la risposta ad una richiesta può richiedere molto tempo).

A differenza degli altri modelli definiti da WS-TXM, in BP gioca un ruolo fondamentale l'intervento umano. Si assume infatti che questo non debba essere relegato alle sole operazioni derivanti da eventi anomali, ma che possa essere una parte importante anche nella gestione e nel monitoraggio dell'intero Processo Business. Pertanto, seppure il modello descriva i partecipanti come servizi Web, è possibile che molte implementazioni di partecipanti interagiscano direttamente, e ordinariamente, con un operatore umano. La natura asincrona di BP permette tempi arbitrariamente lunghi tra richieste e rispose, proprio per supportare questo tipo di interazioni. Si pensi, ad esempio, ai sistemi informatici comunemente utilizzati dalle agenzie di viaggi, in cui è sempre presente un operatore che esercita il ruolo di intermediario tra il cliente e l'agenzia stessa.

Questa assunzione può essere interpretata anche come una diretta conseguenza dell'obiettivo del modello BP: coordinare molteplici Domini Business, di natura indipendente e intrinsecamente variabile, produce interazioni complesse, molto spesso non gestibili in maniera predicibile e/o automatizzabile.

2.3.3.3.1 Modello Concettuale

Un Processo Business rappresenta, come per gli altri modelli di WS-TXM, un particolare tipo di Attività Coordinata. Pertanto è sempre associato ad un coordinatore, che nel caso specifico è stato denominato BPCoordinator.

Questo particolare tipo di Attività è costituita da un insieme di Task, ognuno appartenente ad un Dominio Business. Ogni Dominio Business è responsabile per l'esecuzione di un singolo Task, ed è suddivisibile in sotto-domini, ricorsivamente.

Un Business Task può rappresentare l'elaborazione congiunta di diversi servizi, ma WS-TXM assume che sia una singola unità di lavoro compensabile: ad esempio, può essere una transazione ACID, una LRA, un altro Processo Business, ma può anche essere definito secondo modelli forniti da terze parti. Come la compensazione sia realizzata, è un problema applicativo.

Ogni Business Task, e dunque il Dominio Business cui è associato, viene gestito da un opportuno Agente coordinatore, subordinato del BPCoordinator principale. Tale Agente è responsabile per il proprio dominio, e può essere, in accordo al Business Task che gestisce, un ACIDCoordinator, un coordinatore LRA, un altro BPCoordinator, o può agire da traduttore per un altro protocollo, quale, ad esempio, il protocollo BT dell'OASIS. In ogni caso, la sua particolare natura è invisibile al BPCoordinator, poiché a questo i vari subordinati appaiono sempre come semplici partecipanti. Lo stesso ruolo di interposto non è necessariamente un ruolo prefissato staticamente, anzi, sotto certe condizioni, qualsiasi partecipante potrebbe diventare un coordinatore subordinato: quando e dove sia conveniente usare l'Interposizione è una decisione di progetto.

Come detto, un Processo Business è coordinato da un **BPCoordinator**. Questo può ricevere informazioni su iniziativa dei singoli Task circa il completamento della loro esecuzione (con successo o meno), o può richiedere esplicitamente il loro stato corrente.

Inoltre, periodicamente può richiedere che ogni Task stabilisca un checkpoint, ovvero un punto dal quale poter riprendere l'esecuzione in caso di *failures*, senza dover ripartire dall'inizio. Un Processo Business può terminare con successo, se tutto il lavoro che doveva essere eseguito nel suo Scope è stato eseguito, o con insuccesso, nel qual caso tutto il lavoro già fatto deve essere compensato.

Una differenza sostanziale con i sistemi transazionali classici è che BP è ottimistico, ovvero assume che i casi di *failures* siano poco probabili, risolubili anche fuori linea, se necessario, e spesso richiedendo l'intervento umano. Anzi, quando non sia possibile automatizzare le azioni di recovery/compensazione, quest'utlimo potrebbe essere l'unico "mezzo" a disposizione.

Nel caso in cui, invece, l'automatizzazione sia possibile, ma non sia possibile eseguire correttamente le operazioni di compensazione o di recovery previste, WS-TXM richiede che l'evento sia opportunamente "loggato" e posto all'attenzione dell'amministratore.

Le transizioni di stato di un Processo Business sono mostrate nel diagramma di Figura 70. Qualsiasi BusinessTask, indipendentemente dal proprio modello, deve poter essere interpretato secondo questo diagramma.

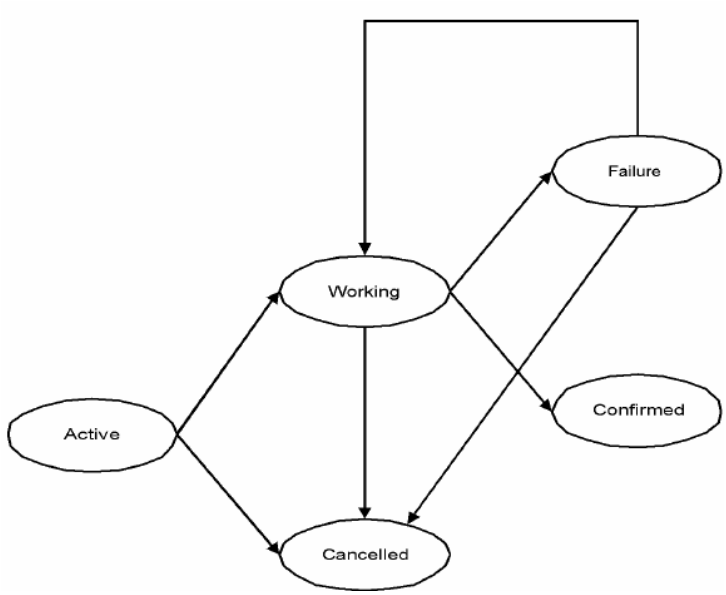

Figura 70 State Diagram di un Processo Business o di un BusinessTask
- tratta da Riferimenti[10 p37] -

Una volta creato, il Processo Business entrerà nello stato di Active. Da qui potrà passare nello stato Cancelled, nel quale non sarà eseguito alcun lavoro. Più spesso transiterà nello stato di Working, in cui potrà rimanere per tutto il tempo necessario alla normale esecuzione. Da tale stato potrà uscire per portarsi nello stato di Confirmed, se tutto il lavoro è stato eseguito senza problemi, nello stato di Cancelled, in cui saranno "disfatti" tutti i lavori già eseguiti, o nello stato di Failure, nel quale potranno essere intraprese azioni di recovery/compensazione.

A differenza del modello LRA, in BP il concetto di compensazione viene applicato al Processo, indipendentemente dai singoli task componenti: se un task riesce a compensare, allora può riportarsi nello stato di Working, se tale è lo stato del Processo

Business. Inoltre, , poiché abbiamo detto che l'intervento umano può essere richiesto ordinariamente per eseguire un'azione compensatrice, questa, in BP, potrà durare un tempo arbitrariamente lungo, e ogni task dovrebbe essere strutturato di conseguenza.

Un possibile Modello Concettuale di BP è mostrato in Figura 71. In esso, la relazione "parent-child" tra BusinessProcess e BusinessTask non va interpretata necessariamente come una relazione di "innesto" secondo WS-CTX, ma, più semplicemente, come una aggregazione logica.

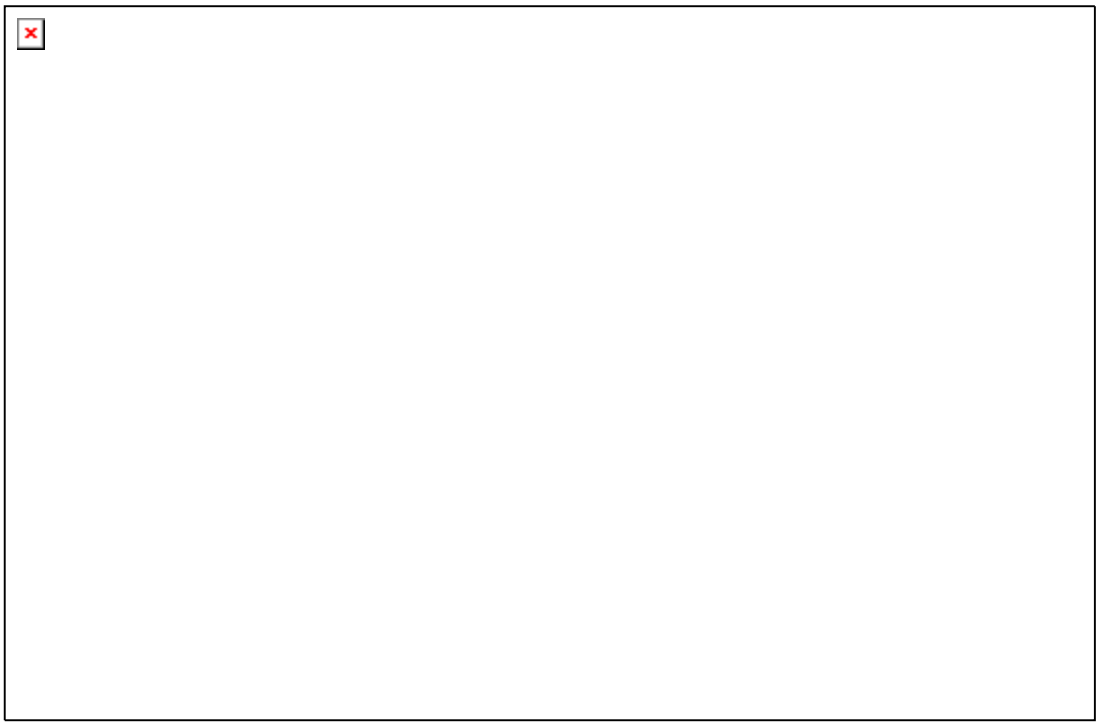

Figura 71 Modello Concettuale di BP

E' opportuno sottolineare che il SubordinateCoordinator, in accordo alla descrizione precedentemente data, non deriva dal BPCoordinator, ma dal generico Coordinator. Ciò non toglie, essendo il BPCoordinator una specializzazione di Coordinator, che il coordinatore subordinato possa essere a sua volta un coordinatore BP.

Si noti pure che l'unica informazione aggiuntiva nel BPContext, rispetto al CoordinationContext, è l'identificativo del processo business.

2.3.3.3.2 Protocolli

WS-TXM classifica i protocolli per Processi Business in due gruppi: protocolli guidati dal Dominio Business, e protocolli guidati dal coordinatore.

Al primo gruppo appartengono i protocolli:

- **Terminate-notification**: utilizzato per informare i partecipanti sullo stato di completamento del Processo Business (Confirmed o Cancelled).
- **BusinessProcess**: utilizzato da un partecipante per comunicare l'impossibilità ad eseguire il lavoro richiestogli. Solitamente è il coordinatore BP a registrarsi presso il partecipante per questo protocollo, in maniera da poter intraprendere delle azioni di compensazione.

Al secondo gruppo, invece, appartengono i protocolli:

- **Checkpoint**: utilizzato dal coordinatore per richiedere a tutti i partecipanti di stabilire un "punto di ripristino", univocamente individuato da un identificatore, dal quale poter ripartire successivamente, su sua richiesta, in caso di *failures*.
- **Restart**: utilizzato dal coordinatore per comunicare ad ogni partecipante che dovrà ripartire da un "punto di ripristino" specificato.
- **WorkStatus**: utilizzato dal coordinatore per richiedere ad ogni partecipante lo stato di avanzamento del lavoro richiestogli nello *scope* del Processo Business.
- **Completion**: utilizzato dal coordinatore per richiedere a tutti i partecipanti la conferma o la cancellazione del lavoro eseguito nell'ambito del Processo.

A prescindere dal particolare protocollo utilizzato, nel modello definito da BP, un partecipante può dimettersi da un Processo Business in qualsiasi momento prima del completamento dell'Attività ad esso associata, semplicemente inviando un messaggio di *removeParticipant* al BPCoordinator.

Figura 72 Modello delle Interfacce di BP

2.3.3.3.2.1 Terminate-notification[118]

Un servizio che voglia essere notificato sulla terminazione di un Processo Business, può registrarsi presso il coordinatore BP come **TerminatorParticipant**.

Il Terminator infatti, che rappresenta un particolare ruolo Participant secondo WS-CF, sarà contattato automaticamente dal coordinatore, mediante il protocollo **Terminate-notification**, quando il Processo Business entrerà in uno stato di completamento (Confirmed o Cancelled).

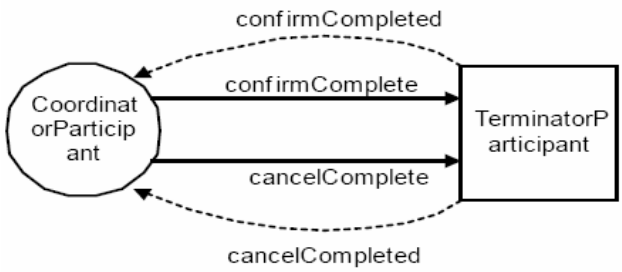

Figura 73 Protocollo Terminate-notification
- tratta da Riferimenti[10 p40] -

Il TerminatorParticipant accetta i seguenti messaggi:

- *confirmComplete*: il coordinatore informa il partecipante che il Processo Business può essere completato con successo. Il TerminatorParticipant dovrà rispondere con un messaggio di riscontro *confirmCompleted*, mentre ogni altra risposta sarà ignorata dal coordinatore.
- *cancelComplete*: il coordinatore BP informa il partecipante che il Processo Business può solo essere annullato. Il TerminatorParticipant dovrà rispondere con un messaggio di riscontro *cancelCompleted*, mentre ogni altra risposta sarà ignorata dal coordinatore.

[118] Nella descrizione fornita dalla Specifica, gli autori trattano questo protocollo come un "protocollo di completamento", ovvero un protocollo mediante il quale l'applicazione può richiedere la conferma o la cancellazione del Processo Business. Di fatto, con la definizione dei messaggi data, il Terminator viene semplicemente notificato sullo stato di completamento del Processo. Tuttavia il Terminator, supponendo che "gestisca" un processo diverso da quello per il quale riceve la notifica, potrebbe portare al completamento quest'altro processo sfruttando proprio detta notifica. In proposito si veda l'esempio riportato a pag.262.

WS-TXM specifica che non è nei suoi scopi decidere se debba esserci o meno un solo TerminatorParticipant, poiché questa è una decisione applicativa.

2.3.3.3.2.2 BusinessProcess

Se un partecipante verifica che non può eseguire correttamente il proprio lavoro, può causare il fallimento dell'intero Processo Business (lo stato di completamento del Processo è portato a FailOnly), oppure può dare l'opportunità al proprio coordinatore "padre" di intraprendere qualche azione di compensazione. Questa compensazione sarà specifica per il Processo Business, e potrebbe consistere, ad esempio, nella rimozione del partecipante dal Processo, o nel richiedere al partecipante qualche lavoro alternativo.

Affinché un coordinatore subordinato possa comunicare al "padre" la propria impossibilità ad eseguire il lavoro richiestogli, il "padre" dovrà essere registrato presso detto subordinato come **BusinessProcessParticipant**, e dovrà utilizzare il protocollo **BusinessProcess**. Quest'ultimo potrà essere eseguito ogni volta sia necessario, e l'esecuzione potrà essere richiesta dall'applicazione client inviando lo specifico messaggio di *coordinate* (definito in WS-CF) al coordinatore subordinato.

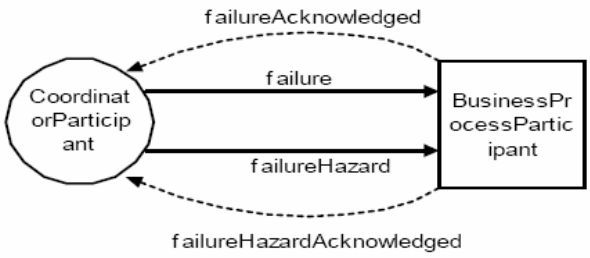

Figura 74 Protocollo BusinessProcess
- tratta da Riferimenti[10 p42] -

Il BusinessProcessParticipant accetta i messaggi:

- *failure*: il subordinato invia questo messaggio a tutti i partecipanti, e dunque anche al coordinatore "padre", per indicare che non può eseguire il lavoro richiestogli.

- *failureHazard*: come prima, e in più il subordinato non è capace di cancellare completamente il lavoro fatto.

Il coordinatore subordinato otterrà in risposta il corrispondente messaggio di riscontro mostrato in Figura 74. Qualsiasi altra risposta sarà ignorata.

2.3.3.3.2.3 Checkpoint

L'applicazione client, nel ruolo di ClientRespondant, può avviare l'esecuzione di questo protocollo in qualsiasi momento, inviando al BPCoordinator (CoordinatorParticipant) un messaggio di *createCheckpoint* (specializzazione del messaggio di *coordinate* definito in WS-CF).

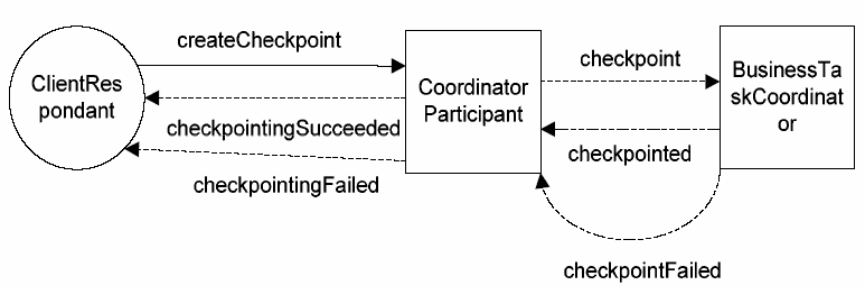

Figura 75 Protocollo Checkpoint
- tratta da Riferimenti[10 p44] -

In seguito alla richiesta del ClientRespondant, il coordinatore BP invierà un messaggio di ***checkpoint*** a tutti i partecipanti registrati come BusinessTaskCoordinator per questo specifico protocollo. Alla ricezione del messaggio detto, il partecipante dovrà creare un "punto di ripristino" consistente, associandovi l'identificatore univoco fornito nel messaggio stesso. Il checkpoint così creato dovrà essere mantenuto per tutta la durata del Processo Business. Se il checkpoint è stato creato senza problemi, il partecipante ritornerà il messaggio di riscontro *checkpointed*, altrimenti ritornerà *checkpointFailed*. L'eventualità che il checkpoint non possa essere creato, non ha comunque implicazioni obbligatorie sul Processo Business.

Se un *checkpointed* è stato ottenuto da tutti partecipanti, allora il coordinatore invierà al ClientRespondant il messaggio di *checkpointingSucceeded*, altrimenti invierà il messaggio di *checkpointingFailed*.

2.3.3.3.2.4 Restart

L'applicazione client, nel ruolo di ClientRespondant, può avviare l'esecuzione di questo protocollo inviando al BPCoordinator (CoordinatorParticipant) un messaggio di *tryRestart* (specializzazione del messaggio di *coordinate* definito in WS-CF).

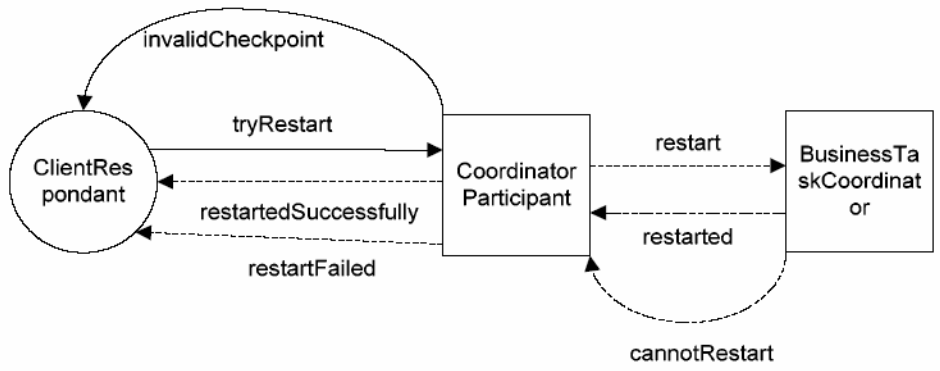

Figura 76 Protocollo Restart
- tratta da Riferimenti[10 p46] -

In seguito alla richiesta del ClientRespondant, il coordinatore BP invierà un messaggio di ***restart*** a tutti i partecipanti registrati come BusinessTaskCoordinator per questo specifico protocollo. Alla ricezione del messaggio, ogni partecipante dovrà ripartire dal checkpoint specificato, e dovrà "disfare" (*undone*) tutto il lavoro che abbia eseguito dopo questo punto. Se il partecipante non potesse ripartire dal punto specificato, dovrà rispondere con un messaggio di *cannotRestart*, altrimenti con un messaggio di *restarted*.

Collezionate le risposte, se un *restarted* è stato ottenuto da tutti i partecipanti, il coordinatore invierà al ClientRespondant un messaggio di *restartedSuccessfully*, includendo l'identificatore del checkpoint. Altrimenti invierà un messaggio di *restartFailed*, e lo stato di completamento dell'Attività che rappresenta il Processo Business sarà portato a FailOnly.

2.3.3.3.2.5 WorkStatus

Un'applicazione può utilizzare questo protocollo per sapere se il Processo Business, come Attività unica, possa completare con successo, ovvero se tutti i lavori richiesti siano stati effettuati (il processo business è entrato nello stato di completamento).

L'applicazione client, nel ruolo di ClientRespondant, può avviare l'esecuzione di questo protocollo in qualsiasi momento, inviando al BPCoordinator (CoordinatorParticipant) un messaggio di *getWorkStatus* (specializzazione del messaggio di *coordinate* definito in WS-CF).

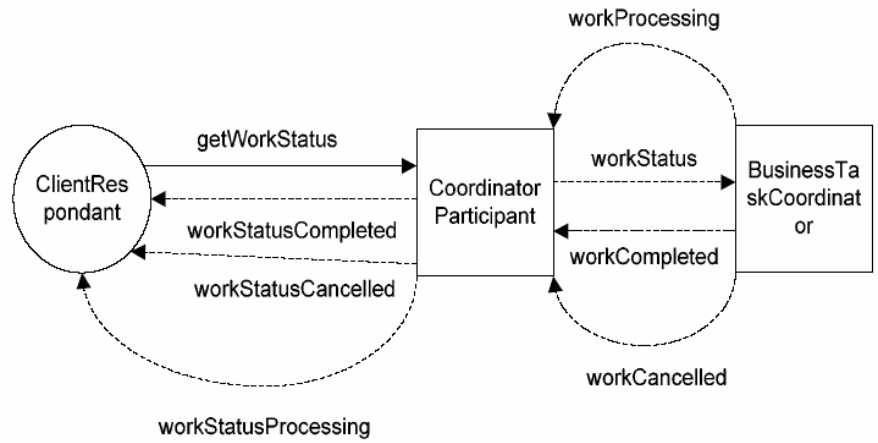

Figura 77 Protocollo WorkStatus
- tratta da Riferimenti[10 p48] -

In seguito alla richiesta del ClientRespondant, il coordinatore BP invierà un messaggio di ***workStatus*** a tutti i partecipanti registrati come BusinessTaskCoordinator per questo specifico protocollo. Ogni partecipante risponderà indicando lo stato corrente del Task gestito, che potrà essere Processing, Completed o Cancelled: se il partecipante è pronto per completare con successo, risponderà con *workCompleted*, mentre se ha già annullato il suo lavoro risponderà con *workCancelled*, e lo stato di completamento del Processo sarà settato a FailOnly. Se il partecipante sta ancora elaborando, risponderà con *workProcessing*.

Ai precedenti messaggi di risposta, WS-TXM suggerisce che possono essere aggiunti dei Qualificatori, ad esempio, per indicare quanto tempo dovrà durare l'elaborazione ancora in corso.

Dato che il lavoro effettuato da ogni dominio business può richiedere un tempo arbitrario per essere eseguito, è anche possibile che un partecipante informi spontaneamente il coordinatore quando ha completato (sia in caso di successo che di fallimento). In tal caso, il partecipante che ha completato, può inviare spontaneamente al coordinatore un messaggio di *workCompleted* o *workCancelled* (mediante *setResponse* definito in WS-CF).

Una volta ottenute le varie risposte, il coordinatore ritornerà al ClientRespondant un messaggio di:

- *workStatusCompleted*: se tutti i partecipanti hanno risposto con un *workCompleted*;
- *workStatusCancelled*: se tutti i partecipanti hanno risposto con un *workCancelled*;
- *workStatusProcessing*: se almeno un partecipante ha risposto con un *workProcessing*;

Nel secondo caso, inoltre, lo stato di completamento del Processo Business sarà portato a FailOnly.

Nel caso in cui tutti i Task abbiano completato, oppure “cancellato”, oltre alle azioni suddette, il coordinatore eseguirà anche il protocollo **Terminate-notification** precedentemente descritto.

2.3.3.3.2.6 Completion

Questo protocollo sarà eseguito automaticamente quando l'Attività che rappresenta il Processo Business sarà terminata. Pertanto, non è possibile avviarlo esplicitamente mediante una richiesta di *coordinate*, nel qual caso il coordinatore BP risponderà con un messaggio di *notCoordinated*.

Il partecipante accetta i messaggi:

- *confirm*: il coordinatore invia questo messaggio a tutti i partecipanti, quando lo stato di completamento dell'Attività business è Success, per richiedere la conferma di tutto il lavoro già effettuato nello Scope del Processo Business. Il partecipante che riceve il messaggio dovrà rispondere con un messaggio di:
  - *confirmed*: in caso di successo;
  - *cancelled*: se il lavoro non può essere confermato; in questo caso, tutto il lavoro già eseguito dal partecipante dovrà essere disfatto (undone);
  - *confirming*: se l'incapacità nel confermare è solo transiente;
  - *unknownResult*: se il partecipante non riesce a determinare se sia possibile confermare.
- *cancel*: il coordinatore invia questo messaggio a tutti i partecipanti, quando lo stato di completamento dell'Attività business è Fail o FailOnly, per richiedere l'annullamento di tutto il lavoro già effettuato nello scope del Processo Business. Il partecipante che riceve il messaggio dovrà rispondere con un messaggio di:
  - *cancelled*: in caso di successo;
  - *confirmed*: se era stato eseguito del lavoro che non può annullare;
  - *cancelling*: se l'incapacità nell'operazione di annullamento è solo transiente. Il coordinatore potrà informarsi in seguito;
  - *unknownResult*: se il partecipante non riesce a determinare correttamente se sia possibile annullare.

Il coordinatore, raccolte tutte le risposte dai partecipanti, notificherà il ClientRespondant con un messaggio di:

- *processConfirmed*: se tutti i lavori del Processo Business sono stati eseguiti con successo;
- *processCancelled*: se tutti i lavori del Processo Business sono stati annullati;
- *unknownResultOccurred*: se non è possibile stabilire il risultato di tutti i partecipanti; tale messaggio contiene gli identificativi di tutti i partecipanti incerti;
- *mixedResponse*: se alcuni partecipanti hanno annullato mentre altri hanno confermato; tale messaggio contiene gli identificativi di tutti i partecipanti, e per ognuno indica se ha annullato o confermato.

2.3.3.3.3 Esempi di utilizzo

In Figura 78 è riportato un esempio, ripreso da WS-TXM, che mostra come le varie entità precedentemente discusse possano essere organizzate in diversi Domini Business, tutti cooperanti in un unico Processo Business principale. Seguendo la direzione delle frecce, che indica la direzione del flusso dei messaggi, si può vedere come ogni partecipante (BusinessTaskCordinator) possa notificare il cordinatore "padre", nel ruolo di BusinessProcessParticipant, per concedergli l'opportunità di intraprendere azioni appropriate nel caso di *failures*.

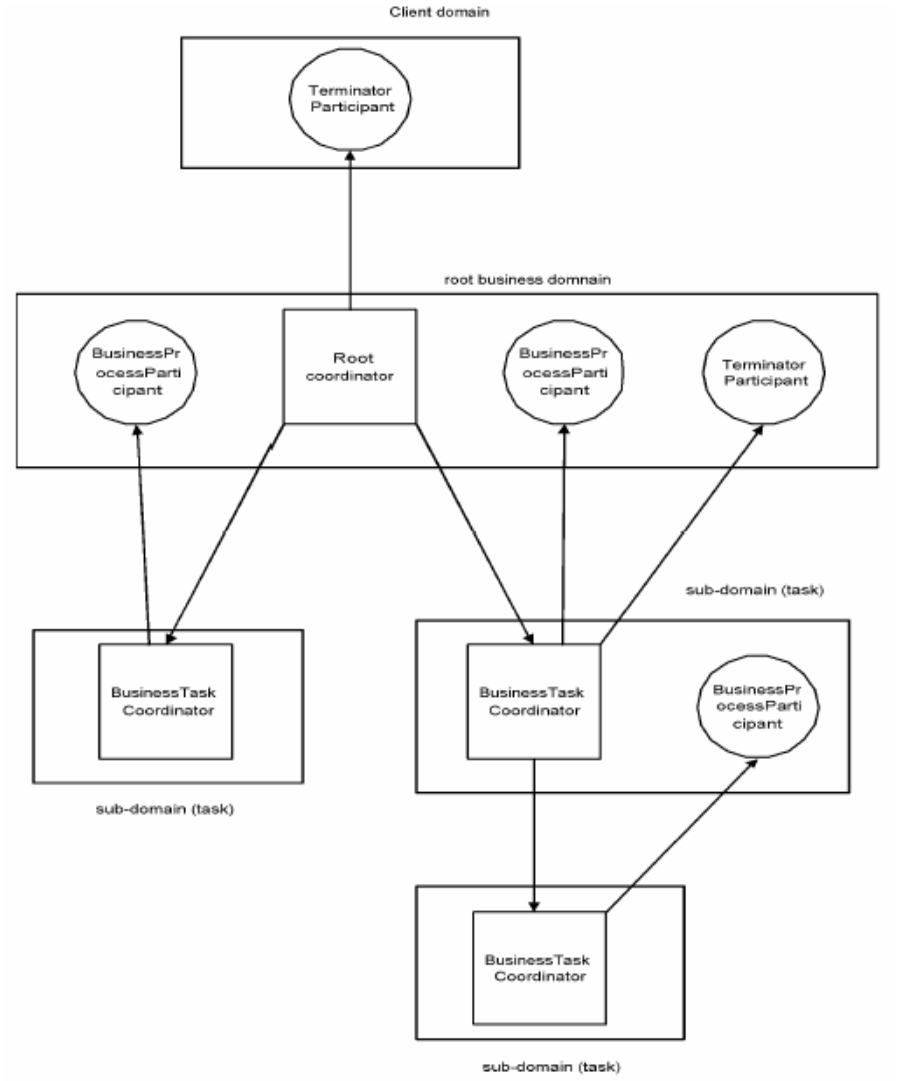

Figura 78 Esempio di Business Process
- tratta da Riferimenti [10 p53] -

Con l'organizzazione mostrata, a detta degli autori del modello di BP, è possibile incrementare il livello di tolleranza ai *business-fault*.

Un ulteriore esempio è mostrato nel Sequence Diagram in Figura 79. Trascurando come al solito le interazioni con i vari ContextService e ALSs, nell'esempio si suppone che l'applicazione client, dopo un certo tempo dall'invio della richiesta applicativa, controlli lo stato di avanzamento dell'esecuzione. Venendo a conoscenza che l'esecuzione non è stata ancora completata, decide di annullare l'intero Processo. L'esempio mostra anche i messaggi di notifica verso un eventuale partecipante Terminator. Quest'ultimo, ad esempio, potrebbe essere il coordinatore per un altro Business Task.

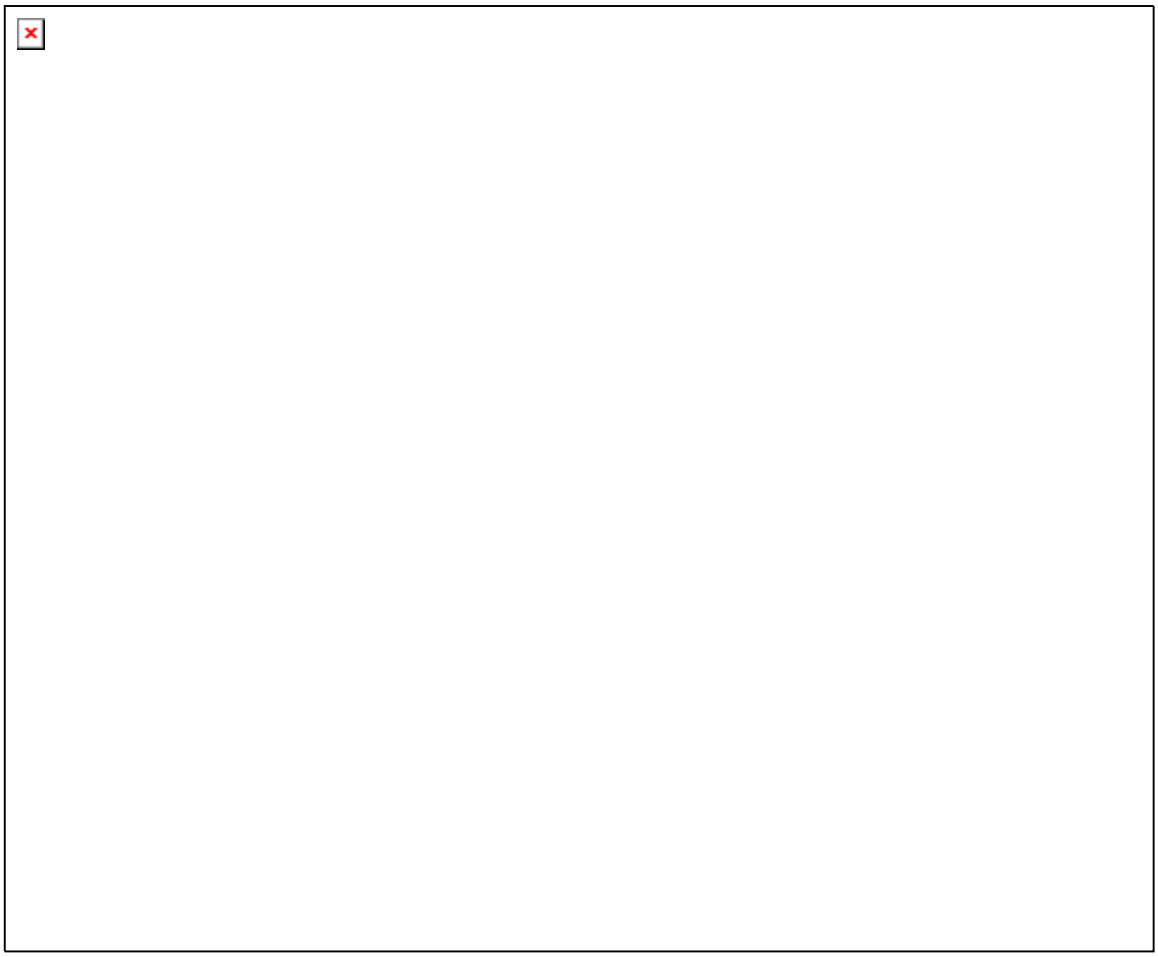

Figura 79 Interazioni conformi BP

2.3.3.3.4 Estensione del messaggio di "status" di WS-CF

Ad una richiesta di *getStatus*, come previsto dal ruolo Participant definito in WS-CF, un partecipante BP può ritornare un messaggio di *status* indicante il proprio stato corrente.

Oltre ai valori di stato già definiti in WS-CF, il partecipante BP potrà indicare uno dei seguenti ulteriori valori:

- Working: il BusinessTask è correntemente in elaborazione;
- Checkpointing;
- Checkpointed: il checkpoint richiesto è stato memorizzato;
- Restarting: il partecipante sta ripartendo da un checkpoint precedentemente salvato;
- Restarted;
- Cancelling;
- Cancelled;
- Confirming;
- Confirmed;
- Failure: si sono presentati degli eventi di *failures* nella normale esecuzione del BusinessTask, e questa non può più proseguire normalmente;

2.3.3.3.5 Definizione di nuovi Qualificatori

Quando un partecipante viene arruolato con un Processo Business, l'entità che effettua l'arruolamento può fornire un certo numero di Qualificatori, che possono essere utilizzati dal coordinatore e dall'applicazione, per influenzare il risultato dell'Attività associata al processo. I qualificatori correntemente definiti da WS-TXM per il modello BP sono:

- **workCompleted**: il partecipante è già pronto a completare il proprio lavoro con successo. Questo è equivalente all'invio, da parte del partecipante, dell'omonimo messaggio di "workCompleted".

- **workCancelled**: il partecipante non può effettuare il proprio lavoro (lo stato di completamento del Business Task è portato a FailOnly). Questo è equivalente all'invio, da parte del partecipante, dell'omonimo messaggio di *workCancelled*.
- **checkpointTimelimit**: indica l'intervallo di tempo entro il quale il partecipante cercherà di mantenere le informazioni sui "punti di ripristino", che eventualmente saranno richiesti mediante il protocollo Checkpoint, ma senza nessun obbligo e/o garanzia.

2.3.3.3.6 Recovery Handling e Interposizione

Gli autori di WS-TXM sottolineano che, a differenza dei sistemi transazionali classici, in cui il recovery è semplice e automatico, in un lungo Processo Business la parte più difficile è proprio quella di determinare quando un *failure* è occorso, e cosa fare in tale situazione. In BP si assume che ogni partecipante, nel ruolo di coordinatore subordinato o di partecipante semplice, sia responsabile per la memorizzazione di sufficienti informazioni per il recovery.

Per condurre le operazioni di recovery, possono venire in aiuto i meccanismi definiti in WS-CF per determinare lo stato corrente di un Processo Business (e agire di conseguenza), e possono essere utilizzati contemporaneamente i vari protocolli precedentemente definiti.

In ogni caso, quando il recovery non sia possibile, la Specifica richiede che tale situazione sia quantomeno "loggata" e posta all'attenzione dell'amministratore. Pertanto, nel peggiore dei casi, l'errore dovrebbe essere memorizzato e successivamente gestito dall'amministratore, mentre, nel migliore dei casi, dovrebbe essere possibile eseguire qualche forma di compensazione o di *undò* in maniera automatica.

2.3.3.4 Scenari di casi d'uso

In questo paragrafo discuteremo i quattro scenari forniti da WS-TXM nel capitolo "Use case scenarios" della Specifica, e cercheremo di capire come essi siano risolubili mediante i modelli precedentemente descritti.

"**Web services coordination**"

Non è un vero e proprio scenario, poiché gli autori, come il titolo indica, trattano la coordinazione di servizi Web in generale. Essi propongono, in alternativa a quello che chiamano "il sovrautilizzo del modello 2PC", i modelli di coordinazione "compensativi" definiti da WS-TXM, poiché, affermano, l'utilizzo di molteplici modelli specifici comporta almeno due vantaggi:

1. fornisce all'utente e al programmatore maggiore chiarezza e semplicità;
2. aiuta a categorizzare gli oggetti e le attività nel giusto modello transazionale;

Sostengono inoltre l'inutilità nel continuare ad utilizzare il modello 2PC poiché, applicandolo ai servizi Web, che combineranno spesso più tecnologie sottostanti in un'unica larga unità di lavoro, non si potranno mai avere le stesse garanzie ACID date dal 2PC applicato alle risorse.

Per quanto riguarda il punto (1), gli autori portano come esempio quello dei servizi ad una fase (servizi ordinari, non transazionali). Dicono che questi potrebbero essere utilizzati[119] nelle attività transazionali in due modi distinti:

- Secondo il 2PC classico, inserendoli in un contenitore a due fasi, che ignori la fase di Prepare.
- Adottando un modello transazionale esteso, specifico per tale tipo di servizi, che fornisca un appropriato supporto per il trattamento degli errori.

[119] Gli autori, implicitamente, assumono che esista la necessità di inserire servizi ad una fase in una transazione.

Utilizzando il 2PC, affermano, in caso di *failure* dopo il Commit, non sarà possibile effettuare l'*undò* di quanto "committato"; utilizzando un modello transazionale specifico, invece, seppure il servizio possa ancora fallire allo stesso modo, si eviterà almeno la confusione nel programmatore che, utilizzando un sistema transazionale tradizionale, si aspetterà sempre la proprietà del "tutti o nessuno", e inserirà servizi ad una e due fasi indistintamente nella stessa transazione.

Il medesimo concetto viene poi ribadito rispetto ai differenti requisiti richiesti ai meccanismi di recovery: mentre nei sistemi transazionali classici è importante soltanto che lo stato del sistema dopo il recovery sia consistente, per transazioni di lunga durata, affinché sia possibile l'intervento umano, è parimenti importante conoscere lo stato di progresso durante il recovery. Infatti, dicono, gli stessi meccanismi per il recovery sono molto differenti, poiché, nel caso dei servizi Web, il recovery automatico è molto meno possibile, e l'intervento umano potrebbe essere ordinaria amministrazione.

"**Timed transactions**":

Anche questo non è uno scenario vero e proprio, poiché gli autori non trattano un particolare caso d'uso. Lo scenario in questione fa invece riferimento alle molte transazioni business che devono avvenire entro un certo intervallo di tempo, trascorso il quale, tipicamente, interviene l'applicazione per portarle a termine in maniera specifica.

Per questo tipo di applicazioni potrebbero essere utilizzati tutti e tre i modelli definiti da WS-TXM, poiché questi, basandosi sul concetto di CoordinatedActivity dato in WS-CF, assegnano sempre, ad ogni Attività, un *tempo di vita* oltre il quale il coordinatore eseguirà automaticamente i protocolli di coordinamento previsti per concludere l'Attività..

"**Arranging a Night-Out**"

Come lo scenario "Web services coordination", anche questo viene proposto dagli autori di WS-TXM per motivare l'adozione del modello "compensativo". Lo scenario è già stato introdotto in 1.3.1, a cui il lettore potrà fare facilmente riferimento prima di continuare nella lettura di quanto segue. In questa sede vogliamo solo sottolineare alcune considerazioni fornite in proposito dagli autori di WS-TXM.

Essi affermano che, inserendo tutte le operazioni previste dallo scenario in una transazione unica, le risorse acquisite durante le prime attività (ad esempio t1) non saranno disponibili fino alla fine della transazione, e questo anche quando dette risorse non servano alle successive attività (quali t2, t3, etc,), risultando così ingiustamente indisponibili per altri clienti. Propongono pertanto l'utilizzo dell'approccio "compensativo", poiché con esso transazioni business di lunga durata possono essere strutturate come molteplici transazioni indipendenti di corta durata, consentendo così che un'Attività possa acquisire e utilizzare risorse solo per il tempo strettamente necessario. Tuttavia riconoscono che utilizzando tale approccio, pur assumendo che le operazioni di compensazione siano progettate correttamente, non è mai garantito che l'intera transazione abbia proprietà ACID: ad esempio, se prima del fallimento di un'Attività di lunga durata altre Attività utilizzassero per un proprio lavoro i risultati parziali, non ancora compensati, si otterrebbero delle letture "sporche".

Vediamo ora in che modo sia possibile "risolvere" lo scenario mediante i modelli compensativi forniti da WS-TXM.

Utilizzando il modello LRA, come proposto dagli autori di WS-TXM in Figura 66, si dovrebbe assegnare ad ogni Attività una LRA indipendente, e tutte le LRAs dovrebbero essere innestate in una LRA principale. In seguito all'ottenimento della prenotazione per il taxi mediante LRA1, il compensatore per tale Attività dovrebbe essere arruolato in LRA5. Se dunque una tra LRA2, LRA3, o LRA4 dovesse fallire, l'utente dovrebbe richiamare la compensazione delle altre LRAs, tra le tre, già eventualmente eseguite, e successivamente potrebbe avviare le altre due LRAs, sempre innestate in LRA5, per prenotare cinema e cena a domicilio, diciamole rispettivamente LRA6 ed LRA7.

Se LRA6 avesse successo, il compensatore per tale Attività dovrebbe essere arruolato ancora in LRA5. Se invece LRA6 dovesse fallire, o se in seguito al successo di LRA6 dovesse fallire LRA7, l'utente potrebbe richiedere l'annullamento dell'intera LRA5, affinché il coordinatore di questa invochi automaticamente i compensatori di LRA1 ed eventualmente, se già eseguita con successo, di LRA6. Quest'ultimo meccanismo potrebbe essere utilizzato anche per LRA2, LRA3 e LRA4, innestandole in una ulteriore LRA innestata a sua volta in LRA5.

Alternativamente si potrebbe utilizzare il modello BP. In tal caso ogni attività potrebbe essere rappresentata da BusinessTask, eventualmente di tipo ACID-Transaction, se necessario, e l'insieme di tali Task potrebbe essere innestato in un BusinessProcess principale. L'applicazione potrebbe ancora essere strutturata come nel caso precedente, e la logica della "compensazione" risulterebbe invariata.

Ricordando quanto detto nella descrizione di WS-CTX (cfr. punto (a) pag 189), a proposito del completamento con successo di un'Attività "padre", il quale richiede che siano prima completate tutte le Attività "figlie", si può facilmente dedurre che la composizione mediante Attività innestate è conveniente che sia una composizione locale. Infatti, supponendo che un servizio remoto sia richiamato all'interno di un'Attività client, diciamola A1, e supponendo che tale servizio definisca un'Attività A2 innestata in A1, l'applicazione dalla parte del cliente non potrebbe controllare l'Attività A2 per richiederne il completamento, almeno con i protocolli previsti, e di conseguenza non potrebbe richiedere il completamento dell'Attività A1 definita localmente. Nelle ipotesi fatte è preferibile, invece, che un Compensator sia registrato dal servizio provider presso A1, e che tale compensatore "piloti" il ciclo di vita di A2, senza che questa sia innestata in A1: così, dunque, A1 e A2 sarebbero gestite separatamente, ognuna con il proprio Scope.

Quanto detto vale per tutti i modelli definiti da WS-TXM, in particolare per i modelli LRA e BP, e giustifica il fatto che, in WS-CAF, un coordinatore subordinato non gestisca necessariamente un'Attività innestata.

La possibilità di innestare Attività può, dunque, essere interpretata più propriamente come una possibilità locale, concessa al richiedente. In questo modo rimane giustificato

pure il fatto che WS-CTX non definisca come più ContextServices possano collaborare per trattare Attività innestate e locate su nodi distinti.

"**Home entertainement system**"

Questo scenario, come il precedente, è già stato da noi introdotto nel paragrafo 1.3.2, a cui il lettore potrà fare riferimento prima di proseguire nella lettura.

Dalla descrizione data in WS-TXM non è chiaro quale sia il modello da adottare per la sua risoluzione.

Secondo la logica proposta dalla Specifica, ogni attività, nello scenario, corrisponde ad una singola Transazione. Il cliente richiede le prenotazioni mediante specifici messaggi applicativi, cui devono corrispondere i locks delle risorse prenotate. Le transazioni vengono eseguite in sequenza, e solo i locks confermati vengono mantenuti tra una transazione e la successiva.

Gli autori affermano che, in questo modo, ad ogni singolo passo l'insieme delle risorse bloccate dovrebbe avere la proprietà di Atomicità, e che, pertanto, non sarebbe necessaria nessuna azione di compensazione. Pertanto, si potrebbe supporre che lo scenario sia proposto come esempio d'uso per il modello ACID-Tx. Tuttavia, il lettore avrà notato che, con la logica esposta, ogni partecipante dovrebbe mantenere il lock temporaneo delle risorse confermate, anche dopo la terminazione della singola Transazione con cui tali risorse siano state prenotate. Questo significa che i partecipanti della Transazione al passo (n)-mo saranno anche i partecipanti della Transazione al passo (n+1)-mo. Ma questi partecipanti, alla fine dell'esecuzione della Transazione n-ma, dovranno sapere che le risorse confermate sono soggette ad un possibile annullamento nelle Transazioni successive, e dovranno sapere anche quale sia l'ultima Transazione della sequenza, la cui conferma sarebbe definitiva. Tuttavia, il modello ACID-Tx non fornisce protocolli per comunicare questo tipo di informazioni.

Anche il modello LRA tratta i partecipanti in maniera atomica, e dunque può essere preso in considerazione. Utilizzando questo modello, si potrebbe strutturare l'applicazione in maniera analoga a quanto detto per lo scenario precedente, "Arranging a Night-Out": ogni Attività sarebbe una LRA, e tutte le LRAs potrebbero essere innestate in una LRA principale. In questo caso il "lock delle risorse" andrebbe inteso in

termini di riserva, piuttosto che di Isolamento. La logica dell'applicazione potrebbe essere gestita esattamente come nel caso del modello ACID-Tx, e la presenza di una LRA principale renderebbe consapevole ogni singolo partecipante sul fatto che le operazioni richieste in una sotto-LRA potrebbero essere successivamente annullate nello Scope della LRA "padre". Tuttavia, anche utilizzando la logica descritta, un partecipante alla LRA n-ma dovrebbe sapere che sarà arruolato nella LRA (n+1)-ma, e per le stesse risorse già riservate nella LRA n-ma. Evidentemente, questo richiede della semantica aggiuntiva, non prevista dal modello in questione.

In definitiva, entrambi i modelli potrebbero essere utilizzati, ma solo se i vari servizi esecutori siano progettati appositamente per l'applicazione descritta.

2.4 Tentative Hold Protocol (THP)

THP è una Specifica[120] del gruppo W3C realizzata da personale Intel, tanto per aggiungere un altro nome di prestigio alla lista di aziende che hanno interesse sull'argomento.

Nasce per facilitare il coordinamento automatizzato delle transazioni in ambiente multi-dominio. Fornisce un protocollo aperto per lo scambio dei messaggi tra sistemi businesses lascamente accoppiati, utilizzabile prima di eseguire una transazione vera e propria. Definisce un'architettura per permettere tentativi, non bloccanti, di assegnamento o prenotazione delle risorse. Può essere utilizzato per aumentare le probabilità di successo della transazione, negoziando, ad esempio, prezzo, quantità, tempi di consegna etc., prima di avviare la transazione in questione, il tutto consentendo al possessore delle risorse di avere pieno controllo delle stesse, e al richiedente di avere sempre dati aggiornati su cui poter decidere.

Per capire in maniera semplice l'idea su cui si basa il suo funzionamento, consideriamo una qualsiasi applicazione di compra-vendita online. THP permette al cliente di richiedere una prenotazione prima dell'acquisto. Tale prenotazione è non bloccante, nel senso che non implica necessariamente una riserva, permettendo così all'offerente di accettare più prenotazioni sulla stessa risorsa (Overbooking). Quando poi un cliente termini l'acquisto, gli altri clienti ricevono notificazione dell'avvenimento, diventando così consapevoli che la loro prenotazione non risulta ulteriormente valida. In questo modo il proprietario delle risorse ha il loro completo controllo, e può aumentare la disponibilità del proprio servizio per raggiungere un maggior numero di potenziali clienti. Dall'altro lato, i clienti, possono prendere decisioni su dati sempre aggiornati, e possono richiedere prenotazioni senza dover successivamente avere a che fare con operazioni di annullamento.

[120] La versione qui presa in esame è stata rilasciata il 28 novembre 2001.

La Specifica classifica gli “approcci” principali attualmente utilizzati per la composizione transazionale dei servizi Web in tre tipi: personalizzato e/o manuale, compensativo, e 2PC.

Per “approccio personalizzato” si intende quello in cui è la stessa applicazione a definire la logica di coordinamento, spesso richiedendo l’intervento umano per eseguire il Rollback di una transazione precedentemente completata. Questo tipo di organizzazione, nonostante preveda alti costi, sia operativi che di implementazione, è ancora la norma.

I benefici derivanti dall’aggiunta di una fase di “negoziazione” mediante THP sono:

- Minimizzazione degli annullamenti: è un beneficio sia per il fornitore, che non gradisce effettuarli, sia per il cliente, che non dovrà richiederli.
- Fornitura al cliente di dati sempre aggiornati su cui poter basare le proprie decisioni: il client viene infatti sempre aggiornato quando una sua prenotazione, precedentemente accordata, non sia più valida. Senza THP, invece, il client potrebbe prendere decisioni su dati obsoleti e non più validi.

L’ “approccio compensativo”, invece, prevede che ogni richiesta in una transazione sia esaudita immediatamente, e che, in caso di fallimento, un’azione di compensazione riporti il sistema ad uno stato consistente. Così è possibile minimizzare la durata dei locks sulle risorse, e dunque aumentare la Disponibilità, ma, a differenza del modello 2PC in cui la consistenza è una proprietà automaticamente soddisfatta, nell’approccio compensativo non vi è alcuna garanzia. Infatti la compensazione stessa potrebbe fallire, o potrebbe non essere possibile compensare completamente, ad esempio, come potrebbe accadere nella vendita di un prodotto con un prezzo caratterizzato da un alto tasso di variabilità.

Utilizzando il protocollo TH si minimizzerebbe la necessità di compensazioni, si ridurrebbe il periodo di tempo tra la richiesta di riserva e l’eventuale compensazione (essendo la richiesta di riserva esaudita in seguito alle richieste di tentativo), e il cliente avrebbe dati sempre aggiornati su cui poter prendere decisioni. Si noti che il secondo beneficio elencato è importante proprio quando la possibilità di compensare diminuisca con il trascorrere del tempo, come nell’esempio di vendita citato.

Infine, per quanto riguarda l' "approccio 2PC", anche se questo fosse utilizzato in maniera tradizionale con il locking delle risorse, l'aggiunta di una fase THP consentirebbe:

- una durata minima della *finestra di incertezza*, e dunque della durata del tempo di lock, dato che i tentativi sarebbero effettuati prima di avviare il protocollo 2PC;
- una minore probabilità di Rollback, essendo i cambiamenti sui dati subito notificati alla controparte;
- che il cliente possa prendere decisioni su dati sempre aggiornati;

Generalizzando, l'adozione di una fase di THP prima dell'avvio di una transazione, comporta benefici sia per il cliente che per il fornitore. Infatti il fornitore potrà aumentare la Disponibilità del proprio servizio ed avere pieno controllo sulle proprie risorse, mentre il cliente, nel caso in cui una prenotazione venga invalidata, potrà continuare la composizione senza perdere le altre prenotazioni già effettuate (mentre, utilizzando il solo modello transazionale classico, l'intera transazione dovrebbe essere ripetuta).

2.4.1 Scenari di casi d'uso

Alcuni scenari di esempio in cui THP può essere utilizzato convenientemente, sono forniti dalla Specifica stessa, e qui di seguito da noi riproposti.

Agenzia di viaggi

L'utente utilizza un servizio "coordinatore", collocato sul sito dell'Agenzia, mediante il quale può accedere ad un vasto numero di servizi sottostanti, ad esempio, per ottenere la prenotazione dell'aereo, dell'hotel, e per affittare un'auto sul luogo di villeggiatura. Tuttavia, in assenza di adeguati sistemi di coordinamento, è costretto ad effettuare molteplici interazioni per assemblare il proprio "pacchetto". Infatti, se dopo aver visionato gli hotels disponibili sia passato molto tempo dalla valutazione dei voli, dovrebbe tornare a verificare che il volo scelto precedentemente sia ancora disponibile

alle condizioni volute. Analogo discorso in seguito alla valutazione dell'auto. Solo quando sia certo che le tre valutazioni fatte siano contemporaneamente valide alle condizioni richieste, vorrà confermare l'acquisto avviando la transazione.

Alternativamente, potrebbe richiedere la prenotazione delle risorse dopo ogni singola valutazione, ma, ad esempio, se dopo la prenotazione del volo non dovessero esserci hotels disponibili, dovrebbe disdire detta prenotazione, e la compagnia aerea subirebbe un danno, non avendo reso disponibile il posto ad altri potenziali clienti.

Utilizzando THP, invece, il cliente richiederebbe delle prenotazioni non bloccanti, e sarebbe notificato dall'agenzia quando una prenotazione non fosse più valida. Non avrebbe bisogno di ricontrollare, dunque, le valutazioni precedentemente fatte, e non dovrebbe disdire prenotazioni già accordate. I singoli servizi delle compagnie, d'altro canto, incrementerebbero la propria Disponibilità, riservando risorse a clienti solo quando strettamente necessario. Inoltre, con questo sistema, le compagnie potrebbero monitorare[121] più facilmente il livello di interesse verso le proprie offerte, e adeguarsi di conseguenza.

Ricerca iterativa automatizzata

Un'azienda produce e vende biciclette. Per la costruzione di queste utilizza componenti, ad esempio sella, catena, ruote, etc., forniti da terze parti. Per abbassare i costi di produzione, l'azienda in questione effettua molteplici ordinativi, al fine di ottenere ogni singolo tipo di componente dal fornitore che gli assicuri i migliori prezzi, tempi di consegna, e così via. Tali ordinativi sono effettuati in una singola "transazione", attività, questa, complessa e limitata dal tempo che un operatore umano può contrattare a telefono o al terminale. Le limitazioni aumentano anche per via dell'instabilità delle condizioni: ad esempio l'azienda acquirente, subito dopo l'invio dell'ordinativo, potrebbe scoprire che il fornitore, in realtà, non ha la quantità richiesta del componente ordinato poiché, nel frattempo, questo è stato venduto ad un altro acquirente; oppure l'azienda potrebbe aver bisogno di 15000 selle in due settimane, e queste potrebbero non essere disponibili tutte dallo stesso fornitore, almeno nei tempi richiesti.

[121] Poiché i clienti prenoterebbero solo quando realmente interessati, i fornitori dei servizi gestirebbero dati reali, e non dati falsati dall'Overbooking.

Inoltre, per ottenere i prezzi migliori, l'azienda deve operare le sue scelte velocemente, per evitare che queste siano limitate dalla variabilità delle condizioni di mercato. Ad esempio, se il componente "sella" è prodotto da 20 fornitori, ognuno di questi offrirà i propri prezzi, tempi di consegna, quantità, e l'azienda dovrà contattarli e valutare tutte le offerte molto rapidamente, cioè prima che le condizioni delle offerte stesse possano cambiare, ad esempio, come conseguenza degli acquisiti della concorrenza.

Mediante THP è possibile automatizzare gran parte delle precedenti interazioni, con un sistema di ricerca iterativa dei fornitori che controlli prezzo, quantità, etc., e che combini le migliori offerte mediante dei "tentativi di riserva". Nel caso in cui un componente non dovesse essere più disponibile, il sistema del fornitore notificherebbe automaticamente il sistema dell'acquirente, che così potrebbe cercare rapidamente soluzioni alternative, e senza necessità di intervento umano o di "cancellazioni".

Spedizioni di merci

I clienti interagiscono con un sistema a cui inviano richieste per ottenere trasporto di merce. I fornitori dei servizi di trasporto interagiscono con lo stesso sistema per esaudire le richieste dei clienti. Ogni fornitore cerca di organizzare le proprie risorse in maniera da minimizzare i costi, e dunque i viaggi dei propri autocarri: l'introduzione di una nuova linea deve infatti essere giustificata da un numero di consegne che consentano un guadagno.

Evidentemente, se il fornitore volesse affidare il compito ad un operatore umano, data la complessità del problema (noto in letteratura come "Salesman Problem"), non riuscirebbe ad ottenere un'organizzazione efficiente. Adottando invece un sistema automatico, potrebbe decidere quali richieste esaudire e quali linee utilizzare per minimizzare i costi di trasporto.

L'automatizzazione può essere ottenuta utilizzando il protocollo TH. I fornitori raggruppano le richieste di trasporto e inviano, per ognuna, delle prenotazioni non bloccanti. Contemporaneamente valutano l'organizzazione migliore, ed eventualmente avviano l'esecuzione della transazione. Se una richiesta venisse esaudita, il servizio del cliente notificherebbe tutti i fornitori che abbiano una prenotazione attiva su tale richiesta, consentendogli così di tentare un nuovo raggruppamento in tempo reale.

2.4.2 Modello Concettuale

Un possibile modello concettuale, estrapolato dalla Specifica, è mostrato in figura seguente:

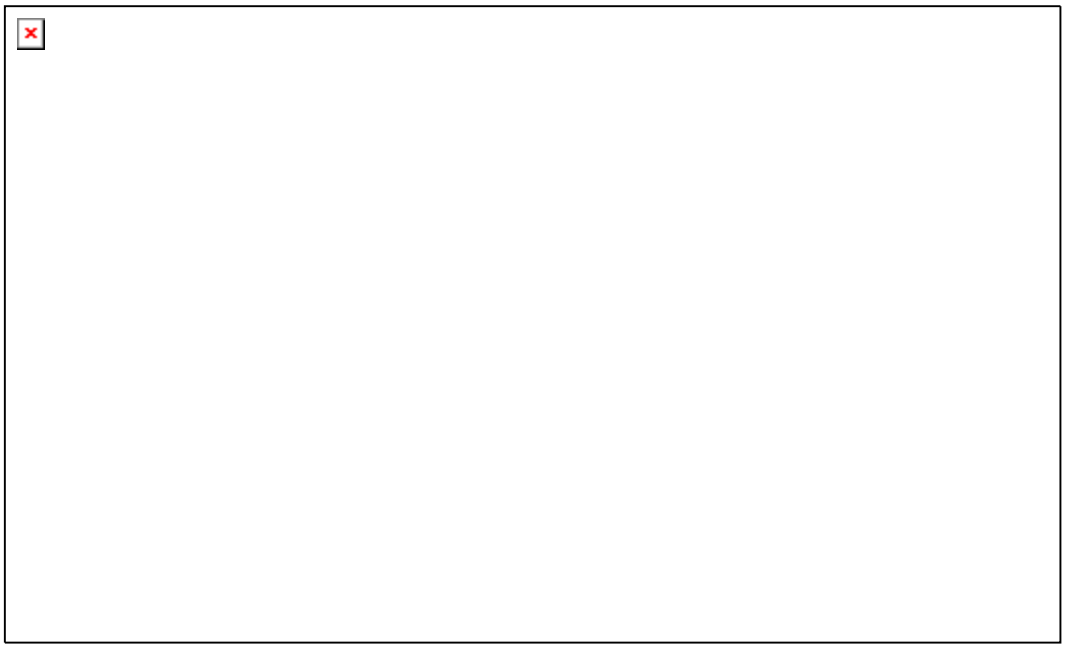

Figura 80 Modello concettuale di THP

Sia dalla parte client che dalla parte server, sono previste tre entità: un servizio applicativo, un coordinatore THP, e un DataStore. I servizi applicativi, rispettivamente ClientApplication e ServerService, implementano la logica business vera e propria. I coordinatori, rispettivamente THPClientCoordinator e THPResourceCoordinator, sono invece responsabili per lo scambio dei messaggi previsti dal protocollo TH, e per la memorizzazione delle informazioni richieste dalla Specifica. Il THPResourceCoordinator è anche responsabile della gestione delle prenotazioni accordate: le annulla quando scade il timeout associato, e invia un messaggio di notificazione al corrispondente coordinatore client. Le due entità stereotipate con "DataStore" rappresentano il luogo in cui dette informazioni vengono memorizzate.

Dalla parte server è mostrata anche una ulteriore entità, denominata RulesIntegrationModule (RIM). Essa rappresenta un modulo software "integrativo" che fornisce le regole con cui devono essere esaudite le richieste client: verifica che siano

richieste valide, stabilisce per quanto tempo continueranno ad esserlo, decide quando e se dovranno essere esaudite, con quale priorità, etc. In realtà, può essere considerata a tutti gli effetti una parte del servizio server (come indicato in Figura 80 utilizzando il medesimo colore), poiché la logica che implementa è parte integrante della logica applicativa, e poiché deve essere fornito dallo stesso proprietario delle risorse (del sistema server). Le funzionalità assegnate al RIM implicano pure che questo conosca la disponibilità delle risorse richieste: quando una nuova richiesta di prenotazione viene considerata valida, il RIM controlla la disponibilità della corrispondente risorsa, e in caso affermativo inserisce un trigger nel sistema che gestisce le risorse (ad esempio un DBMS). Questo trigger notificherà il coordinatore di risorse circa qualsiasi cambiamento nello stato delle risorse prenotate. In tal modo il coordinatore di risorse potrà notificare ai corrispondenti coordinatori clients quando una loro prenotazione non sia più valida.

La Specifica precisa i requisiti di ogni componente precedentemente definito come di seguito esposto.

Richiede che il **coordinatore client**:

1. determini, al suo avvio, lo stato di ogni precedente prenotazione effettuata e accordata;
2. sia implementato in maniera “leggera”, affinché sia capace di girare su svariate piattaforme;
3. fornisca un’interfaccia per permettere all’applicazione client di:
    - richiedere la prenotazione di una specifica risorsa da uno specifico proprietario;
    - effettuare interrogazioni sullo stato delle prenotazioni ottenute;
    - annullare una prenotazione ottenuta;
    - effettuare interrogazioni sulle attività “loggate” dal coordinatore;
    - modificare una prenotazione ottenuta.

I requisiti per il **coordinatore di risorse** sono, invece:

1. all’avvio, deve determinare lo stato di ogni prenotazione accordata e deve verificare che l’ “expirationTime” ad essa associato non sia scaduto;

2. deve poter rispondere alle richieste di prenotazione sia sincronamente (la risposta viene inviata immediatamente), che asincronamente (prima viene inviato solo un "ack", e successivamente il messaggio di risposta *holdRequestResponse*);
3. deve poter accedere alle risorse, gestite con delle *regole di integrazione*, per soddisfare nuove richieste di prenotazione;
4. deve notificare asincronamente i clients coinvolti quando una risorsa non sia più disponibile, ad esempio perché assegnata definitivamente;
5. deve definire un'interfaccia che permetta all'applicazione del proprietario delle risorse di:
    - interrogarlo sulle prenotazioni accordate;
    - annullare prenotazioni accordate;
    - interrogarlo sulle attività "loggate".

Requisiti del RulesIntegrationModule:

1. deve implementare le regole business associate con la concessione di una prenotazione e il relativo tempo di validità;
2. deve inserire triggers sul sistema che gestisce le risorse concesse, i quali, quando "stimolati" da un cambiamento di stato delle risorse associate, dovranno notificare il coordinatore di risorse su tale cambiamento.

I requisiti richiesti, in generale, ad una qualsiasi Implementazione del modello sono:

1. non deve richiedere uno specifico modello o approccio transazionale;
2. deve consentire la comunicazione dei messaggi anche attraverso "firewalls".

2.4.3 Protocollo

Un diagramma a stati finiti che rappresenta il protocollo, tra un singolo ClientCoordinator e un singolo ResourceCoordinator, è mostrato di seguito:

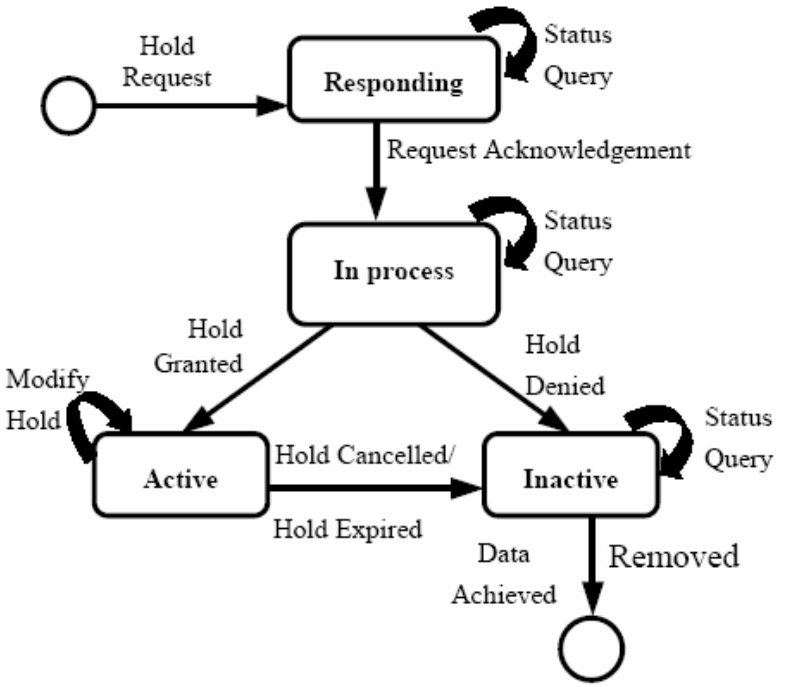

Figura 81 State Diagram del protocollo Tentative Hold
- tratta da Riferimenti[35 p174] -

Il significato degli stati è il seguente:

- Responding: è lo stato iniziale, raggiunto dopo che un cliente abbia inviato una richiesta di prenotazione.
- In process: è uno stato intermedio, indicante che una richiesta è pervenuta, e che un "ack" è stato inviato al cliente.
- Active: è raggiunto quando la prenotazione sia stata accordata. In questo stato il cliente può modificare le "specifiche" della prenotazione con un messaggio di *modifyHold*.
- Inactive: viene raggiunto quando scada la prenotazione, o quando il possessore o il richiedente invii un messaggio di *annullamento* al proprio interlocutore.

Inoltre, in qualsiasi momento, il cliente può richiedere lo stato corrente della prenotazione inviano un messaggio di *statusQueryRequest* (anche nello stato di Active, per il quale il messaggio non è mostrato in Figura 81). Si noti che THP definisce solo i messaggi tra coordinatore client e server, come mostrato in Figura 82.

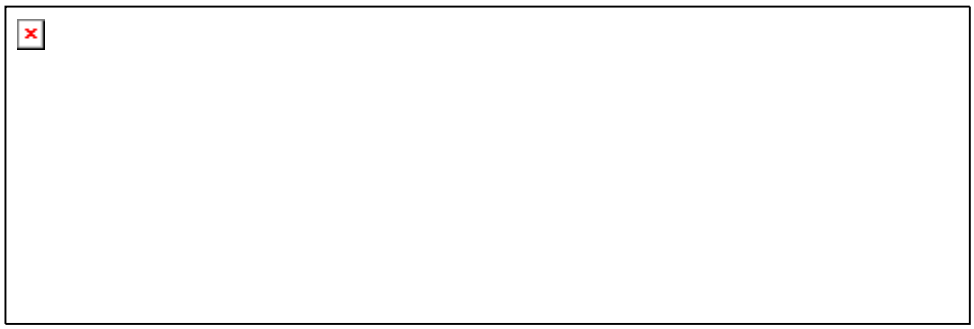

Figura 82 Messaggi previsti dal protocollo TH

Di seguito viene mostrato un Sequence Diagram di esempio, in cui si suppone che due clienti richiedano una prenotazione su una stessa risorsa:

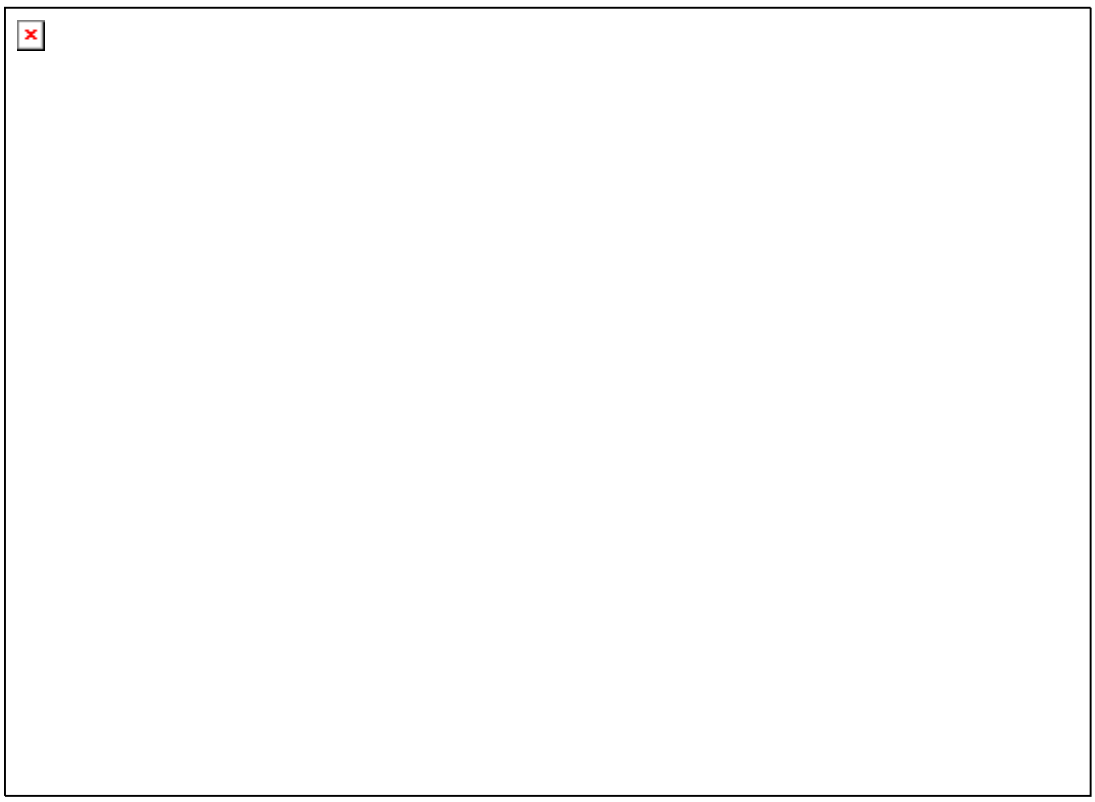

Figura 83 Esempio di interazioni secondo THP

Nell'esempio, dopo la prenotazione, il secondo client decide di acquisire definitivamente la risorsa. A questo punto il "dataManager", ad esempio un DBMS, attiva il trigger, associato a tale risorsa, che informerà il THPResourceCoordinator dell'evento. Quest'ultimo notificherà a sua volta il "client1", per informarlo che la prenotazione precedentemente effettuata non è più valida.

2.4.4 Recovery Handling

La Specifica richiede che il DataStore client memorizzi, in maniera persistente, come minimo la lista delle prenotazioni con le associate informazioni (ad esempio, risorsa prenotata, proprietario della risorsa, timeout, etc.), e un "log" storico sulle attività intraprese mediante THP: prenotazioni richieste, prenotazioni accordate, etc.

Analogamente, per quanto riguarda il DataStore del coordinatore di risorse, questo dovrà memorizzare almeno la lista delle prenotazioni con le associate informazioni (ad esempio, risorsa prenotata, il client della prenotazione, timeout, etc.), e un "log" storico sulle attività intraprese mediante THP.

2.4.5 Binding

La Specifica, pur essendo stata pensata per l'ambito dei WebServices, per il quale fornisce la completa descrizione WSDL, si rivolge a qualsiasi ambiente con simili caratteristiche di accoppiamento lasco. Tutti i messaggi vengono definiti mediante XML Schema, come blocchi headers, e possono essere utilizzati come messaggi indipendenti, o associati a messaggi applicativi, inserendoli nell'header o nel body SOAP.

3 ANALISI E CONFRONTO DELLE SOLUZIONI

Come abbiamo visto nella precedente sezione, le soluzioni proposte presentano molte analogie, ma anche importanti differenze.

BTP propone un unico modello, come estensione del 2PC, con due varianti: Atom, che non supporta la Composizione Dinamica, e Cohesion, che la supporta. Entrambe possono essere utilizzate con l'approccio compensativo, ed entrambe possono essere utilizzate con una semantica ACID.

WS-Transactions propone invece due modelli distinti: da una parte AT, per transazioni dalla semantica ACID, che, sintassi a parte, corrisponde praticamente ad X-Open DTP; e dall'altra BA, che segue l'approccio compensativo e supportata la Composizione Dinamica.

WS-CAF, come WS-Transactions, propone modelli distinti: ACID-Transaction dalla semantica ACID; LRA e BP come modelli compensativi.

Senza voler fare torto a nessuno, in questa terza parte della tesi cercheremo di valutare criticamente le varie Specifiche in questione, evidenziando eventuali problemi o lacune presenti in ognuna di esse. La valutazione sarà condotta confrontando le soluzioni proposte rispetto ai requisiti elencati nel paragrafo "1.4 Analisi e specifica dei Requisiti", che costituiranno il nostro punto di riferimento per sottolineare differenze ed analogie.

Per poter inquadrare meglio le soluzioni, nei prossimi tre capitoli analizzeremo preliminarmente alcuni argomenti in maniera più approfondita.

3.1 Modelli tradizionali: analisi rispetto ad Atomicità e Isolamento

Riconsideriamo il protocollo 2PC per transazioni distribuite tradizionali. Notiamo, prima di tutto, che esso si svolge in due fasi, ma che, inserito in un protocollo di coordinamento, si ottengono per quest'ultimo complessivamente tre fasi. Nella prima vengono indicate le operazioni da svolgere, mentre le altre due equivalgono complessivamente al singolo comando di Commit di una transazione locale o remota. Nel seguito chiameremo la prima fase "fase di Composizione".

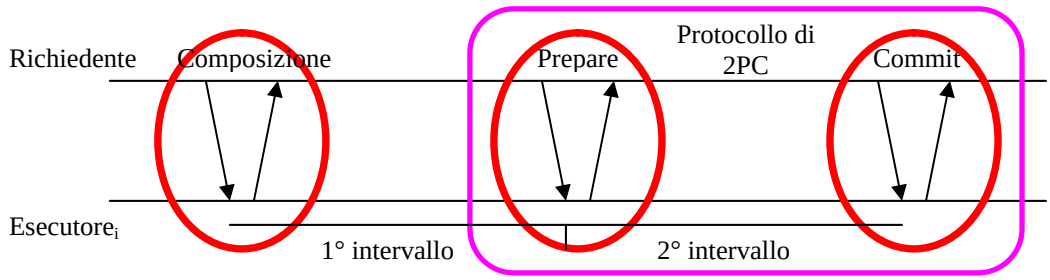

Figura 84 Protocollo di Coordinamento mediante 2PC

Come descritto nel paragrafo "1.2.1 Two Phase Commit e transazioni distribuite", affinché sia garantita l'atomicità, è necessario che dalla *promessa* del partecipante, alla conferma del coordinatore, il primo rimanga vincolato dalla promessa fatta. Tale vincolo si traduce automaticamente nella perdita, da parte del partecipante, della propria autonomia decisionale. Pertanto, in seguito all'invio del messaggio di *prepared* (più precisamente in seguito alla scrittura del record di Prepared da parte del partecipante), non saranno possibili "decisioni anticipate", né, tanto meno, limitazioni circa la durata della promessa (equivalenti ad un timeout).

Per quanto riguarda l'isolamento, abbiamo già notato come il 2PC venga utilizzato comunemente in associazione allo *Strict 2PL*, che per le operazioni di scrittura costituisce una forma di locking obbligata, per poter garantire isolamento massimo nelle operazioni di lettura. Utilizzando tale protocollo, un esecutore deve *lockare* le variabili

interessate da modifica subito dopo la richiesta applicativa, e fino a quando le modifiche non possano più essere revocate, cioè fino alla decisione di conferma comunicata dal coordinatore.

Così, per le operazioni di lettura, i DBMS commerciali possono garantire quattro diversi livelli di isolamento:

L1. **Uncommitted Read**: la transazione accetta letture sporche, ovvero letture di dati modificati da transazioni che ancora non hanno effettuato il Commit. Nella pratica questo comportamento può essere ottenuto ignorando tutti i lock in scrittura, e non richiedendo alcun lock per le letture.

L2. **Committed Read**: la transazione non accetta letture sporche. Ciò è ottenuto imponendo che le letture attendano la liberazione delle risorse, ma il lock in lettura viene rilasciato subito dopo che questa sia stata eseguita, sicchè due letture consecutive possono produrre risultati differenti.

L3. **Repetible Read**: la transazione è garantita anche da letture doppie, ma solo sulle tuple presenti correntemente. In tal caso il lock in lettura deve essere mantenuto fino alla fine della transazione, e nella pratica ciò corrisponde all'utilizzo del 2PL per le richieste di lock in lettura.

L4. **Serializable Read**: la transazione è garantita sulle letture doppie anche rispetto ad eventuali nuove tuple. Questo livello viene ottenuto aggiungendo, alla tecnica utilizzata per il livello 3, particolari meccanismi per il controllo degli inserimenti.

Nella tabella a pagina seguente vengono mostrate le anomalie che possono presentarsi in funzione delle diverse tecniche di lock adottate per le letture. La tabella mostra che è sufficiente utilizzare il 2PL per non avere Perdita di Update. In realtà, questo è vero solo se i lock in lettura siano "promossi" a lock in scrittura non appena sia incontrata un'operazione di scrittura che si riferisca allo stesso dato di quella in lettura (identificheremo questo tipo di operazioni come operazioni di read-write).

Tabella 2 Anomalie e tecniche di Isolamento (X = presenza dell'anomalia)

Tecnica per Scritture	*Tecnica per Letture*	*Letture Sporche*	*Perdita di Update*	*Letture Inconsistenti*	*Aggiornamenti Fantasma*	*Livello di isolamento*
Strict 2PL	-	√	√	√	√	L1
"	*Locking*	-	√	√	√	L2
"	*2PL*	-	-	-	-	L3

Come già osservato, fornire i quattro livelli di isolamento, richiede per le scritture l'utilizzo dello Strict 2PL tra la prima e la terza fase. Nei sistemi tradizionali, per minimizzare la durata della *finestra di incertezza*, le operazioni vengono eseguite (e lo stato antecedente memorizzato) nella fase di Composizione, mentre nella fase di Prepare viene *loggato* solo l'omonimo record.

In un modello per WebServices, prendendo spunto da quanto suggerisce BTP riguardo alle modalità di implementazione degli "Effect" (cfr. p102), può essere invece conveniente anche un'altra strategia. Essendo la latenza della rete molto maggiore del tempo di elaborazione, l'esecuzione può essere portata nella stessa fase di Prepare. In questo modo, anche il lock potrà essere portato solo tra la seconda e la terza fase. Si noti che la strategia detta consiste semplicemente nel posticipare l'inizio dell'esecuzione della transazione, e dunque non comporta conseguenze necessarie sull'isolamento: equivale semplicemente a supporre che la latenza introdotta dalla rete comporti un notevole ritardo nella consegna della richiesta applicativa. Per lo Strict 2PL infatti, le variabili vanno *lockate* subito prima che i dati siano modificati o letti, e fino a quando la modifica non possa più essere revocata, cioè fino al Commit, ma letture e scritture vanno intese rispetto al momento dell'esecuzione, che dunque può essere ritardato a piacimento (purchè sia conservato lo stesso ordine e sia rispettata la condizione necessaria data a p32 per l'isolamento). Da un punto di vista semantico, invece, si ha che la transazione ora inizia dopo l'invio del comando di *prepare* da parte del Coordinatore, ovvero dopo il *confirm* dell'utente. Pertanto, in questo caso, non è più possibile interpretare una risposta applicativa come il risultato di un'operazione di lettura interna alla transazione, essendo questo restituito prima del *confirm*. In altri

termini, per chi ha familiarità con i DBMS, una richiesta applicativa potrebbe essere vista come la chiamata ad una Store Procedure, e solo le operazioni interne a questa sarebbero parte della transazione. Tale tecnica non è, dunque, facilmente conciliabile con l'interattività, in fase di Composizione, imposta dal requisito di Composizione Dinamica. Tuttavia, per evitare i lock nel primo intervallo, essa potrebbe essere adottata da tutti quei servizi che offrano esternamente solo operazioni di lettura ad L1, ed in cui le operazioni di scrittura, ed eventualmente di read-write, siano eseguite sempre come l'ultimo passo di esecuzione del "contratto applicativo".[122]

Molti sono i servizi che rispettano questa logica. Ad esempio, considerando un qualsiasi servizio di vendita on-line, prima della conferma dell'acquisto l'utente effettuerà certamente una serie di letture per visionare il prezzo, la disponibilità, le condizione di vendita e così via. Contemporaneamente, o successivamente, effettuerà una serie di operazioni assimilabili a scritture semplici, poiché, ad esempio, selezionerà i prodotti di interesse, la modalità di pagamento, comunicherà le proprie generalità, il numero della carta di credito, e selezionerà la modalità di consegna della merce. A questo punto, l'utente potrebbe effettuare operazioni simili presso altri servizi, e solo alla fine confermerebbe definitivamente tutti gli acquisti. Assumendo che le operazioni di lettura "visibili" all'utente siano indipendenti da quelle di scrittura, che, eventualmente, potrebbero precederle, il servizio esecutore potrebbe applicare la tecnica detta, concedendo tutte le letture ad L1, e memorizzando temporaneamente tutte le operazioni di scrittura, che, eventualemente trasformate in operazioni di read-write, eseguirebbe realmente solo dopo la conferma finale dell'utente.

Nel seguito faremo riferimento a questa tecnica con il nome di *Strict 2PL ritardato*.

[122] Si noti che, se il DBMS effettuasse solo la compilazione statica delle transazioni, per evitare le Perdite di Update, eventuali operazioni di read-write dovrebbero sempre appartenere ad una stessa transazione locale, altrimenti il DBMS sottostante non potrebbe eseguirle correttamente. Se invece il DBMS compilasse dinamicamente le transazioni, e consentisse che le operazioni transazionali siano comunicate singolarmente, non ci sarebbero problemi, proprio grazie al fatto che, per le operazioni di read-write, i lock in lettura sarebbero "promossi" a lock in scrittura.

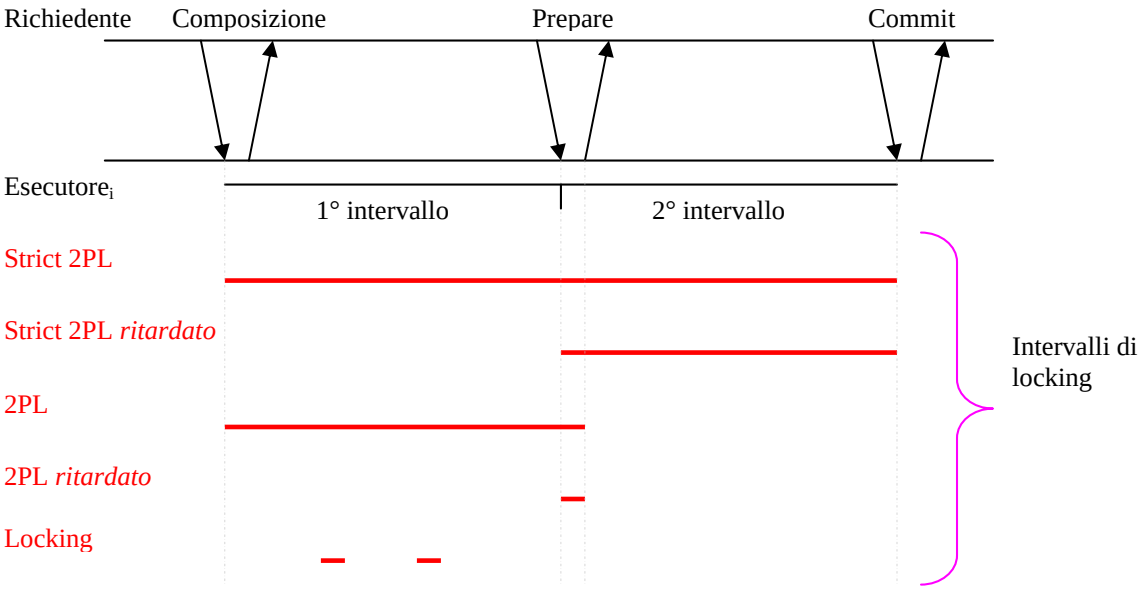

Figura 85 Strategie per l'Isolamento

Sempre per i WebServices, si potrebbe pensare anche ad altre strategie di locking per le operazioni di scrittura. Ad esempio, supponendo che i dati siano letti e memorizzati su un data-base di back-end, e supponendo che al messaggio di *prepare* il servizio reagisca richiedendo al DBMS locale direttamente il commit della transazione, e che in caso di *abort* avvii una opportuna procedura, contenente una ulteriore transazione locale di annullamento, si otterrebbe l'equivalente dell'applicazione del 2PL semplice per le scritture. In questo caso, evidentemente, non potrebbe più essere garantito il livello L2 per le letture, seppure, adottando il 2PL semplice anche per esse, potrebbero essere evitate ancora Letture Inconsistenti e Aggiornamenti Fantasma (poiché la transazione locale di annullamento non potrà comunque violare i lock sulle letture, ma dovrà attenderne il rilascio).

Passando dal 2PL al 2PL *ritardato* per le scritture, valgono considerazioni analoghe a quelle fatte per la variante Strict.

Infine, se per le scritture fosse utilizzato il protocollo di Locking (locks subito rilasciati), otterremmo lo stesso risultato commentato per il 2PL. In effetti, per evitare letture sporche, è sufficiente lo Strict 2PL per le scritture e il Locking per le letture, mentre, per evitare Letture Inconsistenti e Aggiornamenti Fantasma, è sufficiente il 2PL

per le letture e il Locking per le scritture. Se, poi, il 2PL per letture e scritture venga applicato promuovendo i lock in lettura a lock in scrittura, quando le due operazioni si riferiscano allo stesso dato (ovvero per operazioni di read-write), si eviterebbero pure le Perdite di Update.

Tabella 3 Anomalie e tecniche di Isolamento (X = presenza dell'anomalia)

Tecnica per Scritture	*Tecnica per Letture*	*Letture Sporche*	*Perdita di Update*	*Letture Inconsistenti*	*Aggiornamenti Fantasma*	*Livello di isolamento*
2PL/Locking	-	√	√	√	√	L1
"	*Locking*	√	√	√	√	L1
2PL	*2PL*	√	-	-	-	Non Definito
Locking	*2PL*	√	√	-	-	Non Definito

Per quanto detto, e come mostrato in Tabella 3, utilizzare il 2PL o il Locking per le scritture equivale a rinunciare definitivamente all'isolamento, in quanto, dei livelli precedentemente descritti, potrebbe essere garantito solo il livello L1 (assenza di isolamento).

Quanto sopra esposto in relazione ad un protocollo di coordinamento che utilizzi il 2PC, può essere facilmente generalizzato. Ad esempio, supponendo che la fase di Prepare sia associata alla fase di Composizione, si otterrebbero complessivamente due sole fasi. Esse sarebbero, in ogni caso, equivalenti alle fasi di Prepare e di Commit, e di conseguenza dovrebbero essere rispettati tutti i vincoli suddetti per garantire atomicità e isolamento.

3.2 Interpretazioni dei Modelli Compensativi

L'approccio compensativo, da un punto di vista concettuale, è molto semplice. Infatti esso richiede che le azioni siano subito eseguite, i lock subito rilasciati, e in caso di fallimenti, che siano invocate opportune azioni compensatrici.

Questo concetto può essere applicato in modi differenti, rendendo possibili altrettante interpretazioni dei Modelli Compensativi:

Come "Protocollo 2PC equivalente"

Ricordando i protocolli di coordinamento che utilizzano l'approccio compensativo, notiamo che essi sono tutti a due o a tre fasi. Tali protocolli, seppure avrebbero potuto utilizzare la stessa sintassi del 2PC tradizionale, come fa BTP, utilizzano per le fasi una sintassi differente. Ora, a prescindere dalla particolare sintassi, è possibile assumere, per analogia, che tali fasi corrispondano a quelle del protocollo di coordinamento che utilizza il 2PC visto nel capitolo precedente. A tal proposito, per uniformare la terminologia e per sottolineare l'analogia, d'ora in avanti le chiameremo fase di Prepare equivalente e fase di Commit equivalente, e lo stesso faremo per i messaggi di protocollo (ad esempio parleremo di messaggio di *prepared equivalente*).

Sulla base di quest'analogia, richiedere che le azioni siano subito eseguite, e che i lock siano subito rilasciati, equivale ad applicare il protocollo di Locking per le scritture, o al limite il 2PL. Pertanto, il modello compensativo può essere interpretato come un protocollo di coordinamento transazionale *2PC equivalente*, dall'isolamento rilassato anche per le operazioni di scrittura.

Tutti i modelli compensativi proposti sono interpretabili in questo modo, e con le seguenti particolarità:

Cohesion sovrappone fase di Composizione e fase di Prepare equivalente, per ottenere la Composizione Dinamica, pur lasciando le due fasi distinte. Con l'ottimizzazione One-Shot, le due fasi vanno a coincidere completamente.

BA utilizza tre fasi distinte, e nella variante BA-wPC, il messaggio di *prepare equivalente* può essere considerato implicito.

LRA utilizza due sole fasi, e la prima può essere considerata come associazione della fase di Composizione e della fase di Prepare equivalente.

BP utilizza ancora tre fasi, ma la fase di Prepare equivalente è opzionale. Tale opzionalità, se non utilizzata, riporta al caso di LRA.

In accordo a questa interpretazione, l'approccio compensativo viene pubblicizzato dai produttori come la tecnica per risolvere i problemi di performance di concorrenza per transazioni di lunga durata, quali sono le Transazioni Business. Per capirne la reale convenienza, riconsideriamo uno degli scenari precedentemente trattati, in cui gli autori propongono l'utilizzo del modello compensativo. Consideriamo, ad esempio, lo scenario "1.3.1 Arranging a Night-Out". Se in seguito alla prenotazione del taxi, del ristorante, e di un posto a teatro, la prenotazione dell'albergo dovesse fallire, l'utente dovrebbe invocare l'operazione di compensazione sui tre servizi già prenotati. Ora, dalla prenotazione alla richiesta di compensazione, potrebbe passare teoricamente anche molto tempo, e tutte le risorse prenotate, in tale tempo, rimarrebbero indisponibili per altri utenti (proprio ciò che il modello della compensazione voleva evitare), semplicemente perché momentaneamente assegnate. Così, oltre la perdita di isolamento, non ci sarebbero miglioramenti nella disponibilità del servizio, essendo la risorsa implicitamente bloccata dall'assegnazione anticipata, e potendo i servizi fornitori perdere clienti allo stesso modo di quanto accadrebbe utilizzando lo Strict 2PL.

Tuttavia, ipotizzando che l'operazione di assegnamento di una risorsa utilizzi in scrittura delle "variabili condivise" (come la variabile di *disponibilità* descritta nello scenario "1.3.4 Manifacturer-Supplier-Shipper"), ovvero delle variabili che siano modificate anche per l'assegnamento di altre risorse, allora l'approccio compensativo potrebbe effettivamente risultare conveniente, in quanto consentirebbe di assegnare più

risorse contemporaneamente. Nella stessa ipotesi, utilizzando invece lo Strict 2PL dell'approccio tradizionale, si potrebbe assegnare una ulteriore risorsa solo al completamento della transazione che blocca l'eventuale "variabile condivisa", anche quando si abbiano a disposizione più risorse da assegnare.

Come un protocollo per standardizzare la Procedura di Annullamento

Abbiamo già osservato che, in molti casi, è utile disporre di una procedura per annullare in extremis delle transazioni precedentemente confermate. L'applicazione di questo concetto prevede due fasi: nella prima vengono avviate e completate delle transazioni, mentre nella seconda sono eventualmente invocate le corrispondenti procedure di annullamento (dal richiedente), o di completamento (dal servizio esecutore). Del resto, anche l'approccio compensativo prevede due fasi, una fase di richiesta e una fase di compensazione, e dunque è possibile interpretarlo proprio allo stesso modo. Con questa interpretazione, dunque, una transazione business[123], nella prima fase, diventa "contenitore" di transazioni ACID.

Potrebbe essere interpretato in questo modo il modello LRA, che utilizza due sole fasi, ma anche tutti gli altri modelli compensativi: basta assumere che la fase di Prepare equivalente sia una fase di interrogazione, per conoscere lo stato di completamento delle sotto-transazioni richieste nella fase di Composizione. Ovviamente, tale interpretazione può sembrare una forzatura per il modello BTP, ma si addice perfettamente al modello BA, in cui il messaggio di *prepare equivalente* corrisponde al messaggio di *completed*, e al modello BP, in cui la fase di Prepare equivalente, oltre ad essere facoltativa, corrisponde ad una vera e propria interrogazione sullo stato di avanzamento.

Abbiamo volutamente utilizzato il condizionale - potrebbe essere interpretato - poiché, affinché tale interpretazione sia verosimile, l'esecutore dovrebbe interpretare correttamente una richiesta applicativa contenente due Contesti, quello della transazione principale di tipo Business, e quello della sotto-transazione ACID, tra loro correlati in qualche maniera. L'invio di Contesti relazionati è possibile solo per i modelli di WS-CAF, che adottano il modello delle Transazioni Innestate, e prevedono che la relazione "padre-figlio" possa essere comunicata nel Context, ma riguardo all'interpretazione,

[123] Il termine "transazione business" viene qui utilizzato impropriamente, in quanto non corrisponde al concetto di "Transazione Business" precedentemente definito nel paragrafo 1.4.2.4.

non vi è nulla di esplicito in tale Specifica. Pertanto, i modelli compensativi possono essere utilizzati con questa interpretazione, tacitamente per relazioni intra-dominio, mentre, per relazioni multi dominio, solo ipotizzando che i servizi si accordino anticipatamente sul significato del Context.

Come un modello per il controllo del flusso di programma

Una ulteriore interpretazione ci viene suggerita da WS-CAF. In proposito, si riveda la Figura 65 a pag 239. Chiunque abbia un minimo di esperienza con un qualsiasi linguaggio di programmazione strutturato, non si meraviglierebbe se gli si dicesse che in tale figura è rappresentato il flusso di controllo di un'applicazione. Infatti, ogni "biforcazione" presente nella struttura ivi rappresentata potrebbe essere tradotta in una istruzione del tipo "if *opN=Success* then *exec(opM)* else *compensate(opN)*", e il fatto che *opN* e *opM*, in realtà, siano, o possano essere, delle invocazioni remote, non comporta alcuna differenza.

Con questa interpretazione, una transazione business[124] rappresenta semplicemente un modello per strutturare a run-time il flusso di controllo di un'applicazione distribuita: gli elementi della struttura rappresentano operazioni qualsiasi, locali o remote, eventualmente anche transazioni ACID, mentre un'operazione di compensazione può consistere in un'operazione di annullamento (analogamente all'interpretazione data al punto precedente), oppure in una generica operazione dipendente dalla particolare applicazione. Ad esempio, un'operazione di compensazione potrebbe servire per riportare i dati ad uno stato consistente, eventualmente annullando gli effetti di eventuali transazioni che abbiano effettuato letture sporche o inconsistenti. Chiaramente, se questi "effetti" fossero stati utilizzati nel frattempo da altre transazioni, si potrebbero eseguire altre operazioni di compensazione. In questo modo, si potrebbe ottenere una compensazione a valanga che difficilmente potrebbe riportare allo stato di partenza; per di più, la stessa operazione di compensazione potrebbe fallire. Così, una ulteriore operazione di compensazione potrebbe servire per informare l'utente, o l'amministratore di sistema, e la decisione di continuare in una compensazione fallita potrebbe dipendere dalla specifica applicazione: ad esempio, se la compensazione

[124] Vedi nota 123.

consistesse nel ritentare l'acquisto di un oggetto, potrebbe essere inappropriato eseguirla se il prezzo dell'oggetto, nel frattempo, fosse salito rapidamente.

La relazione di "innesto" tra transazioni business, in questo contesto, potrebbe avere un significato generale e sempre valido, oppure potrebbe avere un significato specifico per ogni particolare applicazione. Ad esempio, potrebbe significare che il risultato delle sotto-transazioni sia soggetto a quello finale della transazione "padre", e potrebbe richiedere che i compensatori delle sotto-transazioni si arruolino con il coordinatore della transazione "padre" per conoscerne l'esito definitivo, come è possibile per il modello LRA di WS-CAF; oppure potrebbe significare che le riserve (i locks) debbano essere mantenute tra le sotto-transazioni, come descritto nel paragrafo 2.3.3.4 relativamente allo scenario "Home entertainement system". Oppure, ancora, potrebbe avere un significato che consenta al richiedente (compositore) di comunicare, in parte o in toto, la struttura applicativa voluta ai servizi esecutori: ad esempio, potrebbe servire per indicare che l'operazione di compensazione, per una certa richiesta applicativa, debba corrispondere ad un preciso insieme di altre operazioni esposte dall'esecutore.

In definitiva, un'agevolazione per costruire applicazioni distribuite, che, ovviamente, funzionerebbe bene solo per applicazioni progettate da "una sola mente", ovvero per applicazioni intra-dominio. Tutti i modelli compensativi, in maniera più o meno simile, possono essere interpretati e utilizzati in questo modo, in particolare quelli definiti da WS-CAF, che adotta pure il modello delle Transazioni Innestate, seppure con qualche limitazione: come imposto dalla Specifica, una ACID-Transaction non può essere innestata in una LRA, ma solo in un Business Process.

3.3 Tecnica dell'Overbooking

Analizziamo ora un'altra questione, quella dell'Overbooking. Questa tecnica consiste nell'assegnare temporaneamente una stessa risorsa a più richiedenti contemporaneamente, prima dell'assegnamento definitivo ad uno solo di essi; oppure, equivalentemente, nel dichiarare una disponibilità maggiore di quella reale per una data risorsa. Questo significa, in caso di assegnamento definitivo ad un utente (o al massimo numero di utenti nel secondo caso), che la risorsa, implicitamente o esplicitamente, viene "revocata" agli altri utenti. Per conservare l'atomicità, tali "revoche" devono essere effettuate prima di comunicare un *prepared equivalente*. Pertanto, la tecnica non è conciliabile con:

> *protocolli che fondono fase di Composizione e fase di Prepare equivalente in una fase unica, quando si voglia garantire anche l'atomicità, poiché la "promessa" deve essere mantenuta e non è revocabile.*

Ne segue che l'Overbooking è possibile, insieme all'atomicità, solo in tutti quei protocolli in cui esista la fase di Composizione separatamente dalla fase di Prepare equivalente.

Per quanto riguarda l'isolamento, almeno concettualmente, esso sarà rilassato, poiché il risultato di una transazione dipenderà da quello di altre transazioni. Nella pratica, invece, non vi è nessuna conseguenza necessaria, poiché l'unica conseguenza obbligatoria è che un assegnamento definitivo produca sempre l'Abort delle altre transazioni interessate alla stessa risorsa. Tuttavia, se, ad esempio, l'utente avesse *visibilità* della disponibilità reale delle risorse, è chiaro che tale "visibilità" si tradurrebbe in operazioni di lettura inconsistenti.

Relativamente a questa tecnica, abbiamo anche visto che il modello proposto da THP, che applica la strategia dell'Overbooking con Notifica, è molto efficiente, poiché prevede una minimizzazione delle transazioni che vanno a cattivo fine, e ciò, semplicemente permettendo che un partecipante notifichi tutti i richiedenti interessati ad una risorsa, concessa temporaneamente in condivisione, quando questa sia assegnata

definitivamente ad uno di loro. Dalla descrizione degli scenari proposti da THP, potrebbe sembrare che i vantaggi derivanti dall'adozione di questo modello possano essere utili solo per particolari tipi di servizi, come servizi di compra-vendita, di prenotazione, etc. In realtà, è possibile utilizzare la stessa tecnica in maniera molto più generale: basta interpretare come risorse anche i dati. Nel capitolo 3.1 abbiamo visto che, per un servizio esecutore, in molti casi può essere conveniente fornire solo letture non isolate. Tuttavia, abbiamo anche visto, e più volte sottolineato, che la mancanza di isolamento potrebbe non essere desiderabile per molte applicazioni. Adottando il modello dell'Overbooking con Notifica, e interpretando i dati come risorse, si potrebbero conciliare le due cose, poiché sarebbe possibile avvisare il richiedente, in tempo reale, su eventuali letture inconsistenti. Ad esempio, non appena un dato letto dal richiedente sia modificato da altri, questo sarebbe avvisato, e potrebbe decidere eventualmente di rileggere il dato, o di abortire la transazione. Tale discorso può essere applicato particolarmente bene quando per le scritture sia utilizzato lo *Strict 2PL ritardato*. In tal caso, infatti, le operazioni di scrittura o di read-write, avendo assunto che non producano risultati che debbano essere letti dall'utente, sarebbero naturalmente isolate, e il problema si presenterebbe solo per le letture concesse esternamente al richiedente.

Per quanto appena detto, la strategia dell'Overbooking con Notifica rappresenta un notevole vantaggio, e il protocollo THP, supportandola, sembrerebbe quasi indispensabile. In realtà, almeno in quei protocolli che prevedono già un messaggio con cui un partecipante possa revocare la propria partecipazione, e che supportano la Composizione Dinamica, per la quale il richiedente può richiedere l'annullamento ad un partecipante, il semplice protocollo TH risulta già parte del protocollo in questione (a meno della possibilità di modifica della prenotazione). Alcuni dei protocolli precedentemente descritti permettono che il partecipante "revochi" la partecipazione con un messaggio di *aborted equivalente*, prima che inizi la fase di Prepare equivalente. Tale messaggio, dunque, potrebbe essere utilizzato allo scopo, ma solo se, in qualche modo, sia inviato al richiedente (o se questo coincida con il coordinatore), e se non richieda l'abort della transazione, poiché, diversamente, si perderebbero automaticamente tutti i benefici dovuti all'adozione del protocollo TH stesso.

Negli altri casi, è possibile aggiungere una fase THP prima dell'esecuzione del protocollo di coordinamento?

A detta degli autori di THP, ciò dovrebbe essere sempre possibile. Tuttavia, si nota subito che esistono delle questioni da definite. Infatti, come minimo, dovrebbe essere definita la relazione tra messaggio di richiesta applicativa, comprensivo di "richiesta di trattenuta" secondo TH, e arruolamento "temporaneo" del partecipante. Ad esempio, se l'arruolamento avvenisse subito dopo la richiesta applicativa, e se non fosse supportata la composizione dinamica, non sarebbe più possibile eliminare tale partecipante dalla transazione, e questa dovrebbe terminare necessariamente con un Abort. In questo caso, per non perdere i benefici di THP, se l'arruolamento fosse di tipo "Push", questo potrebbe essere ritardato il più possibile, al limite, fino a poco prima della conferma; ma con un arruolamento di tipo "Pull", la soluzione al problema non è immediata. In ogni caso, più soluzioni sarebbero altrettanto lecite, e non essendo definite dalla Specifica, sarebbero "soluzioni proprietarie". Di conseguenza, THP non è integrabile in maniera standard con i protocolli di coordinamento che non supportino la composizione dinamica, e questo costituisce un grande limite, poiché, evidentemente, compromette totalmente l'Interoperabilità.

3.4 Come le Specifiche soddisfano i requisiti

Siamo giunti, finalmente, alla valutazione delle Specifiche. Nel seguito, tutti i modelli trattati, ovviamente THP escluso, saranno valutati rispetto ai requisiti fissati in precedenza, e ciò, per generalità, indipendentemente dal fatto che seguano una semantica ACID, o un approccio compensativo.

Composizione dinamica

La composizione dinamica è supportata, seppure in maniera differente, da tutte le famiglie di Specifiche.

BTP la supporta mediante il modello Cohesion, all'interno delle fasi di Composizione e di Prepare. In quest'ultima il *richiedente* può richiedere al coordinatore la preparazione e/o la cancellazione selettiva dei partecipanti, cosicché, da un punto di vista pratico, si può dire che la dinamicità, in BTP, è ottenuta concedendo al richiedente gli stessi poteri che il classico coordinatore TM ha nella fase di Prepare. Pur tuttavia, il coordinatore continua ad avere funzionalità proprie, poiché la "concessione" dei poteri viene revocata subito dopo la richiesta di confirm.

WS-Tx la supporta mediante il modello BA. Essa viene ottenuta in maniera simile[125] a quanto fa BTP, solo che in BA il richiedente corrisponde realmente al coordinatore.

WS-CAF la supporta sia con il modello LRA che con il modello BP. Tuttavia, a differenza di Cohesion e BA, nei modelli di WS-CAF è ottenuta in maniera molto particolare: eseguendo ogni operazione (invocazione) come una transazione separata, e dunque controllabile separatamente. Evidentemente, utilizzando questa stessa strategia,

[125] In Cohesion la Composizione Dinamica permette la ripetizione, un numero arbitrario di volte, della fase di Prepare. In BA, invece, la fase di Prepare equivalente non può essere ripetuta. Inoltre, mentre in BTP il Participant rappresenta un'entità separata dal servizio applicativo, in BA, quello del partecipante è un semplice ruolo (un'interfaccia). Di conseguenza in BTP, per ogni esecutore utilizzato nella composizione, il richiedente dovrà conoscere due indirizzi, quello del servizio applicativo, che è certamente noto, e quello del Participant associato.

tutti i modelli supporterebbero il requisito in questione (ACID-Tx compreso). Essa equivale, infatti, a concedere al richiedente tutti i poteri del coordinatore TM, e la presenza del "coordinatore" assegnato ad ogni singola operazione, rappresenta evidentemente una semplice (ed inutile) intermediazione.

Revoca di Partecipazione

Un partecipante può "revocare" la propria partecipazione ad un transazione business, per la quale risulta arruolato, come segue:

- Atom e Cohesion: inviando un messaggio di *cancelled* in qualsiasi momento prima della fase di Commit. La revoca automatica è possibile utilizzando il qualificatore standard "InferiorTimeout".
- AT: inviando un messaggio di *aborted* prima di essersi dichiarato *prepared*.
- BA: inviando un messaggio di *exit* prima della fase di Prepare equivalente.
- ACID-Tx: inviando una decisione autonoma di *voteRollBack* in qualsiasi momento prima della fase di Commit.
- LRA: non è possibile utilizzando l'arruolamento di tipo "Push", mentre, per l'arruolamento di tipo "Pull", è possibile in qualsiasi momento, inviando un messaggio di *removeParticipant* al coordinatore.
- BP: per l'arruolamento di tipo "Push", inviando, in qualsiasi momento, un messaggio di *workCancelled* al coordinatore; per l'arruolamento di tipo "Pull", oltre al suddetto messaggio, può essere utilizzato il messaggio di *removeParticipant*.

(Si noti che l'utilizzo del messaggio di *removeParticipant* per i protocolli LRA e BP, associato all'arruolamento di tipo Pull, comporta il problema descritto in "2.3.2.2.1 Protocollo di Arruolamento", e perciò, di fatto, non è utilizzabile).

Atomicità

Come preciseremo meglio analizzando il requisito "Rilassamento dell'Atomicità**"**, tutti i modelli proposti assumono che la "promessa" fatta nella fase di Prepare, o di Prepare-equivalente, possa essere revocata in qualche modo. Tuttavia alcuni di essi definiscono le informazioni che dovrebbero essere mantenute persistentemente per garantire l'atomicità, anche in caso di *failures*.

BTP utilizza il modello dell'Abort Presunto, e precisa tutte le informazioni che dovrebbero essere mantenute persistentemente per il recovery. Per condurre il recovery non definisce esplicitamente messaggi particolari, poichè richiede semplicemente la rispedizione di quelli ordinari. Ciò nonostante definisce il ruolo di StatusRequestor che, in generale, può essere utilizzato per richiedere ad un coordinatore o ad un partecipante il suo stato corrente. Definisce inoltre il meccanismo della Redirection, particolarmente utile in caso di *failures*, con cui, qualsiasi servizio applicativo o BTP, può comunicare al proprio interlocutore un nuovo set di indirizzi validi.

WS-Tx, per il modello AT, si comporta in maniera analoga a BTP, seppure non definisca con la stessa precisione le informazioni da mantenere persistentemente. Inoltre non definisce nessun messaggio per conoscere lo stato corrente di un peer, ma definisce il messaggio di *replay*, con cui un partecipante può richiedere al coordinatore la rispedizione dell'ultimo messaggio di protocollo già spedito. Per BA, invece, non fa alcun riferimento esplicito all'atomicità, ma richiede genericamente che qualsiasi variazione di stato sia preceduta da una memorizzazione permanente dello stato antecedente al nuovo. Anche in tal caso non sono definiti esplicitamente messaggi particolari per il recovery, poichè è semplicemente consentita la rispedizione di quelli ordinari, ma sono definiti i messaggi di *getStatus* e *status*, per permettere di interrogare l'interlocutore e conoscerne lo stato corrente.

WS-CAF, per il modello ACID-Transacions, afferma che viene utilizzato il modello dell'Abort Presunto ma, a riguardo, in nessuna occasione esplicita le informazioni da mantenere persistentemente. Definisce, inoltre, due messaggi aggiuntivi, rispettivamente per richiedere al coordinatore l'avvio del recovery e lo stato corrente. Per il modello LRA, fa esplicito riferimento all'atomicità ma, di fatto, non definisce

nessun meccanismo per garantirla. Per BP, la caratteristica dell'atomicità non viene proprio presa in considerazione.

Da quanto detto segue che, esclusi i modelli compensativi (BA, LRA e BP), l'atomicità è certamente supportata per interazioni intra dominio, per le quali la possibilità di non rispettare questa caratteristica rappresenta semplicemente una scelta interna, ovvero un'opportunità in più per gestire la composizione dei servizi.

Rilassamento dell'Atomicità

Tutti i modelli presi in considerazione rilassano in qualche modo la proprietà di Atomicità.

BTP, per entrambi i modelli Atom e Cohesion, ammette infatti che i partecipanti mantengano il proprio potere decisionale anche in seguito alla fase di Prepare, permettendo loro di intraprendere decisioni autonome, eventualmente anche in maniera pianificata, e non solo conseguenti a *failures*.

WS-Tx adotta strategie differenti per i modelli AT e BA. Nel primo caso assume che un partecipante possa inviare un messaggio di *inconsistentInternalState* al coordinatore, anche in seguito alla fase di Prepare, con cui comunicare la propria "inabilità ad esaudire le proprie obbligazioni".

Nel modello BA, invece, è possibile che nella fase di Commit equivalente un servizio esecutore risponda con un messaggio di *fault*, in seguito ad una richiesta di *compensate*.[126]

WS-CAF, per il suo modello ACID-Tx, si comporta in maniera analoga a BTP, poiché, anche dopo la fase di Prepare, permette che un partecipante prenda una decisione finale indipendente da quella del coordinatore. Tuttavia, a differenza di BTP, non specifica se tali decisioni possano essere prese, o meno, anche in maniera pianificata (e dunque è lecito farlo). Per il modello LRA, un servizio esecutore può sempre rispondere con un *failedToCompleted* o un *failedToCompensated*, in seguito alla fase di Commit-equivalente, mentre per BP, può rispondere con un messaggio di *failure* o *failureHazard* in qualsiasi momento.

[126]E' opportuno osservare che il modello BA, in accordo all'approccio compensativo, non consente l'invio di un *fault* in risposta ad un messaggio di *close* (*commit equivalente*).

Segnalazione risultato non atomico

BTP, sia per Atom che per Cohesion, indica chiaramente quali informazioni dovrebbero essere mantenute persistentemente da ogni partecipante, e quale dovrebbe essere il comportamento di questi affinché un risultato non atomico sia segnalato al coordinatore anche in caso di *failures*. Tuttavia assume, in definitiva, che queste siano scelte applicativa, e dunque non pone obblighi in generale.

WS-Tx, si affida a quanto specificato da WS-ReliableMessaging affinché un messaggio di *inconsistentInternalState* o di *fault* sia correttamente recapitato, ma tali messaggi potrebbero non essere inviati.

WS-CAF, almeno per il modello ACID-Tx, seppure in maniera imprecisa, definisce meccanismi analoghi a quelli specificati in BTP, e similmente, assume che le informazioni da mantenere persistentemente siano un problema applicativo.

Nel modello LRA, si richiede che un partecipante memorizzi solo la condizione per cui non possa compensare, ma non anche quella per cui non riesca a completare (in accordo al modello compensativo, in cui il completamento dell'operazione avviene subito, nella fase di Prepare-equivalente, che in LRA coincide con la fase di Composizione). Tuttavia, i messaggi di *failedToCompleted* o di *failedToCompensated* potrebbero non essere inviati. Nel modello BP, invece, la persistenza è completamente a carico dell'applicazione, e si assume che, in taluni casi, il recovery non sia possibile.

Da quanto detto segue che, escluso BP, tutti i modelli, potenzialmente, potrebbero essere utilizzati per garantire la segnalazione di risultato non atomico, ma non necessariamente. Pertanto, come per l'atomicità, il requisito in questione può essere considerato garantito solo per relazioni intra-dominio.

Procedura di Annullamento

Solo i modelli LRA e BP, come spiegato in "3.2 Interpretazioni dei Modelli Compensativi", possono essere interpretati come un "protocollo per standardizzare la Procedura di Annullamento". Tuttavia, come osservato nello stesso paragrafo, tale interpretazione non è esplicitata nella Specifica, e così va bene soltanto per relazioni intra-dominio.

Contratti Vincolanti relativi all'Atomicità

Solo **WS-Tx**, relativamente al modello AT, assume che un messaggio di *inconsistentInternalState*, comunicato da un partecipante al coordinatore, eventualmente anche in seguito ad un messaggio di *prepared*, indichi "inabilità ad esaudire le proprie obbligazioni". Inoltre, precisa che un partecipante che abbia inviato un *abort* può ritenersi svincolato dalla transazione solo dopo aver ricevuto un *aborted*.

Per quanto riguarda **BTP**, si noti che, anche per una Atom, è il coordinatore ad essere obbligato, dalla Specifica, a trattare tutti i suoi partecipanti in maniera atomica, ma. per questi ultimi, non viene specificata nessuna obbligazione.

Per i modelli BA, LRA e BP, come precedentemente detto, l'Atomicità non è supportata, e pertanto tale requisito non è applicabile.

Isolamento

BTP, sia per il modello Atom che per il modello Cohesion, consente il rilassamento dell'Isolamento anche per le scritture, poiché assume che eventuali scelte in proposito siano un problema applicativo.

WS-Tx non specifica mai esplicitamente come possa essere ottenuto l'isolamento, tuttavia, propone il modello BA come modello compensativo, e, come osservato a pag.154, in AT richede implicitamente lo Strict 2PL per le scritture.

WS-CAF propone invece i modelli LRA e BT come compensativi, e il modello ACID-Tx come modello dalla semantica ACID, ma, a differenza di WS-Tx, il livello di isolamento non è indicato implicitamente nemmeno per il modello ACID-Tx.

Contratti Vincolanti relativi all'Isolamento

Come osservato al punto precedente, solo WS-Tx, per il modello AT, impone implicitamente lo Strict 2PL per le scritture.

Per i modelli compensativi (BA, LRA e BP) il requisito in questione non è applicabile, poiché l'Isolamento non è supportato.

Disponibilità

Per quanto riguarda l'isolamento, abbiamo già osservato che l'unico modello in cui questo non può essere rilassato rispetto alle operazioni di scrittura è AT di WS-Tx.

Passiamo all'Overbooking. Per quanto riguarda l'Overbooking semplice (senza notifica), potrebbe essere applicato con tutti i modelli, in quanto gli unici modelli in cui vengono fuse fase di Composizione e fase di Prepare equivalente sono modelli dall'Atomicità rilassata. Tuttavia, almeno con i modelli che non prevedono la composizione dinamica, ovvero Atom, AT e ACID-Tx, questo tipo di strategia produrrebbe notevoli Abort per le transazioni, poiché il Coordinatore per questi modelli è vincolato a trattare tutti i partecipanti in maniera atomica.

Per quanto riguarda, invece, l'Overbooking con Notifica, ne parleremo al punto successivo.

Infine, relativamente all'ultimo fattore che abbiamo indicato in "1.4.2.1 Requisiti degli esecutori" come strettamente legato al requisito della Disponibilità, solo WS-Tx definisce un modello per la Sicurezza, mentre BTP e WS-CAF, almeno per ora, delegano la questione a future versioni delle rispettive Specifiche.

Overbooking con Notifica

La strategia dell'Overbooking con Notifica (stile THP), potrebbe essere applicata in Cohesion, che permette una decisione autonoma di *cancelled* anche prima della fase di Prepare, e in cui tale decisione, che può essere intesa come una revoca, non richiede necessariamente l'abort della transazione. Tuttavia il messaggio di *cancelled* viene consegnato al coordinatore, e solo interrogando quest'ultimo, ne potrà essere al corrente anche il richiedente. Pertanto, affinché il richiedente possa ragionare sempre su dati aggiornati, sono necessarie continue interrogazioni, a tutto discapito dell'efficienza (a maggior ragione nel caso in cui il coordinatore sia implementato su un nodo distinto da quello del richiedente).

Nel modello Atom, invece, non è applicabile, poiché il messaggio *aborted* di un partecipante obbliga il coordinatore a prendere la stessa decisione per l'intera transazione, e come già osservato, si perderebbero tutti i benefici della "notifica".

Per lo stesso motivo, non è applicabile nemmeno nel modello AT e nel modello ACID-Tx. E' invece possibile in BA, dove il coordinatore coincide con il richiedente, e dove è prevista esplicitamente la possibilità di revoca di un partecipante prima dell'inizio della fase di Prepare-equivalente.

Per i modelli LRA e BP, utilizzando l'arruolamento di tipo "Pull", un partecipante potrà dearruolarsi, in qualsiasi momento, inviando un messaggio di *removeParticipant* al coordinatore. Tuttavia non c'è modo di portare questa informazione dal coordinatore al richiedente. Scegliendo un arruolamento di tipo "Push", relativamente al modello LRA, esso è un protocollo a due sole fasi totali, e tra le due fasi non prevede nessun messaggio di revoca dal partecipante; per BP, invece, che è un protocollo a tre fasi con la fase di Prepare equivalente facoltativa, un partecipante può autonomamente inviare un *workCancelled* secondo il protocollo WorkStatus, e poiché tale cancellazione non implica l'annullamento dell'intero Processo Business, può essere utilizzata in luogo della "revoca" di partecipazione. Tuttavia, come per l'arruolamento di tipo "Pull", non vi è nessun meccanismo affinché questa informazione sia passata al richiedente.

In ogni caso è opportuno sottolineare che anche per Cohesion e BA, le rispettive Specifiche non fanno alcun riferimento esplicito alla tecnica in questione. Pertanto, assumeremo che il requisito sia soddisfatto per le relazioni multi-dominio, ma con la consapevolezza che ciò rappresenta una nostra concessione.

Interposizione

Tutti i modelli supportano il concetto di Interposizione. Alcuni consentono anche l'Interposizione Mista, ovvero tra coordinatori di diverso tipo, mentre altri non la consentono.

In particolare BTP permette l'Interposizione Mista tra i due modelli Atom e Cohesion, WS-Tx[127] non la permette, e WS-CAF la permette solo attraverso il modello BP:

[127] In un documento di fonte Microsoft-IBM, riportato in [28], gli autori criticano la possibilità di interposizione mista offerta da BTP con la seguente giustificazione: in BTP i due tipi di partecipanti per Atom e Cohesion sono identici, e così potrebbero essere arruolati in transazioni dell'uno o dell'altro tipo indifferentemente; in questo modo risulterebbe difficile capire la correttezza dell'applicazione risultante.
In realtà, la critica non è giustificata, poiché abbiamo visto che in BTP entrambi i modelli seguono la stessa semantica (nessuno è ACID): essi differiscono solo per la Composizione Dinamica, che non è supportata in Cohesion. Il problema nascerebbe invece se fossero intercambiabili i partecipanti AT e BA in WS-Tx, avendo i rispettivi modelli una semantica differente (ACID il primo e Business il secondo).

ricordiamo che un coordinatore BP può "pilotare" solo altri coordinatori BP, ma questi possono avere partecipanti sia di tipo LRA che di tipo ACID-Tx (eventualmente anche di altri tipi, definiti da altri modelli di coordinamento).

Nessuna delle Specifiche contempla, invece, il problema dei *deadlock strutturali* di tipo *permanente*. Assumendo che agevolazioni in proposito non siano essenziali, essendo l'isolamento rilassato, possiamo comunque considerare il requisito sull'Interposizione soddisfatto anche per relazioni multi-dominio.

Conoscenza dello "stato" di elaborazione dell'esecutore

BTP mette a disposizione il ruolo di StatusRequestor, con cui il richiedente può interrogare l'esecutore per conoscerne lo stato corrente. Mette anche a disposizione il messaggio *BTPContext_Reply*, per consentire ad un esecutore di indicare autonomamente che ha finito i propri "sotto-arruolamenti". Tuttavia, la fine degli arruolamenti, potrebbe non coincidere con la fine delle elaborazioni.

Anche BA, con i messaggi di *getStatus* e *status*, permette al coordinatore (che coincide con il richiedente) di interrogare un partecipante per conoscerne lo stato corrente. Contemporaneamente, con la variante BAwPC, un partecipante può indicare autonomamente quando ha completato il proprio lavoro con un messaggio di *completed*, e lo stesso accade con la variante BAwCC, ma solo dopo che il coordinatore abbia inviato un messaggio di *complete*. In ogni caso, il messaggio di *completed* citato viene scambiato in quella che abbiamo chiamato, seppure impropriamente, fase di Prepare equivalente. Pertanto, non può essere considerare ai fini del soddisfacimento del requisito in esame.

Infine WS-CAF, tramite quanto definito in WS-CF, per tutti i suoi modelli, e con la stessa sintassi di BA, permette al richiedente di interrogare un partecipante per conoscerne lo stato corrente, ma, a differenza di BA, prevede l'intermediazione del coordinatore. Inoltre BP, mediante il protocollo WorkStatus, fornisce un ulteriore meccanismo affinché il richiedente possa interrogare un partecipante, e quest'ultimo possa rispondere che la propria elaborazione è ancora in corso. Tale messaggio potrebbe eventualmente trasportare un qualificatore che indichi una stima dell'ulteriore durata, ma, purtroppo, il qualificatore in questione non è stato standardizzato. Il protocollo

WorkStatus può anche essere avviato direttamente da un partecipante, che così può comunicare spontaneamente quando ha terminato i lavori richiesti, ma tale comunicazione è riportata solo al coordinatore, e il protocollo non impone che quest'ultimo riporti la notizia al richiedente. Si noti che anche l'esecuzione del protocollo WorkStatus equivale ad eseguire la fase di Prepare equivalente. Pertanto, come detto per BA, in ogni caso non può essere considerato ai fini del soddisfacimento del requisito in esame.

Da quanto detto si deduce che l'unico modello, che non mette a disposizione nessun meccanismo per la fase di *composizione*, è AT.

Conoscenza dello "stato" di composizione del richiedente

BTP mette a disposizione ancora il ruolo di StatusRequestor, con cui un partecipante può interrogare il richiedente per conoscerne lo stato corrente. Mette a disposizione anche il Qualificatore standard TransactionTimelimit, con cui il richiedente può indicare opzionalmente al coordinatore per quanto tempo dovrebbe durare la fase di Active (comprende le fasi di Composizione e di Prepare), che ha sua volta lo indicherà agli Inferiors.

In WS-Tx, il richiedente può indicare opzionalmente la durata della transazione, nel messaggio di creazione del Context, mediante l'attributo Expires. Inoltre BA, sempre con i messaggi di *getStatus* e *status*, permette ad un partecipante di interrogare il coordinatore (che coincide con il richiedente) per conoscerne lo stato corrente.

In WS-CAF, analogamente a WS-Tx, è previsto l'attributo Expired nel Context, ma questa volta non è opzionale. Inoltre, tramite quanto definito in WS-CF, per tutti i suoi modelli e con la stessa sintassi di BA, permette ad un esecutore di interrogare il richiedente per conoscerne lo stato corrente, ma, a differenza di BA, prevede che il coordinatore sia utilizzato come intermediario.

Recovery anche nella fase di Composizione

Non è supportato in BTP, in AT, in ACID-Tx, e in LRA. In BA è supportato naturalmente, poiché la Specifica supporta la rispedizione affidabile dei messaggi, e richiede che ogni variazione di stato sia preceduta da una memorizzazione permanente

dello stato antecedente al nuovo. Adottando questa strategia, non è stato necessario definire nessun protocollo specifico. Al contrario, BP definisce una serie di protocolli, peraltro concettualmente molto buoni, proprio con quest'obiettivo.

Protocolli Efficienti

Come precedentemente osservato, tutti i modelli supportano la tecnica dell'Interposizione. Inoltre, alcuni di essi prevedono delle ottimizzazioni.

BTP prevede l'ottimizzazione One-shot[128], che riduce il numero di messaggi scambiati fondendo le fasi di Composizione e di Prepare equivalente (tale ottimizzazione, ovviamente, non è compatibile con la tecnica dell'Overbooking con notifica). Prevede anche l'ottimizzazione Resignation, corrispondente alla Read-only del 2PC tradizionale, la "conferma in una fase", quando esista un solo partecipante, e la Preparazione Spontanea.

Anche AT e ACID-Tx prevedono l'ottimizzazione Read-only, e ACID-Tx prevede pure la "conferma in una fase". Come protocollo *2PC equivalente*, anche BA supporta la Preparazione Spontanea, con la variante BAwPC.

Infine, per il recovery, BTP, AT e ACID-Tx utilizzano il modello dell'Abort Presunto.

Comunicazioni Sicure

Come già precedentemente notato relativamente al requisito sulla Disponibilità, solo WS-Tx definisce un modello per la Sicurezza, mentre BTP e WS-CAF, almeno per ora, delegano la questione a future versioni delle corrispondenti Specifiche.

Molteplici modelli per la Sicurezza

Il modello per la Sicurezza definito da WS-Tx si basa sulla famiglia di Specifiche Security, e questa supporta molteplici modelli per la Sicurezza.

[128] L'ottimizzazione One-shot presenta tuttavia l'inconveniente che il servizio esecutore non riceve notifica dell'avvenuto arruolamento. In questo modo dovrà attendere per tutto il *tempo di vita* della transazione anche quando non sia stato arruolato, con gravi conseguenze sull'efficienza.

Comunicazioni Affidabili

Tutte le famiglie di Specifiche definiscono meccanismi di Binding per ottenere un trasporto affidabile.

BTP richiede semplicemente un protocollo di trasporto in cui i messaggi arrivino correttamente o non arrivino del tutto. WS-CAF prevede anche che i messaggi arrivino in maniera scorretta. WS-Tx, utilizza allo scopo la Specifica WS-ReliableMessaging.

Compatibilità tra Protocollo Business e tecnologia dei data-base

I modelli dalla semantica ACID (AT e ACID-Tx) sono tutti compatibili, per loro natura, con la tecnologia dei DBMS, in quanto rappresentano una estensione all'ambiente dei WebServices del protocollo X-Open DTP. Per gli altri modelli, nessuna delle Specifiche prende in seria considerazione il problema. Solo BTP, quando specifica il significato degli "Effects", accenna alla relazione tra il protocollo BT e la tecnologia dei DBMS. BTP può comunque essere ritenuto compatibile con detta tecnologia, almeno quando, per supportare gli "Effects", sia utilizzato l' "approccio tipico dei DBMS" descritto in Tabella 1 a pag 102. Tuttavia, essendo possibili anche altri "approcci", il requisito in esame può essere considerato soddisfatto solo in modo parziale.

Modello per *Transazioni ACID per Web*

BTP non fornisce esplicitamente nessun modello per transazioni dalla semantica ACID, ma entrambi i modelli Atom e Cohesion possono essere utilizzati per interfacciarsi con l'interfaccia XA dello standard X-Open, in quanto il protocollo BP rappresenta esso stesso una generalizzazione di tale Standard.

WS-Tx e WS-CAF forniscono rispettivamente i modelli AT e ACID-Tx.

QoS a Runtime (A-I-S-Overbooking-etc.)

Nessuna Specifica fissa differenti livelli di QoS, e tranne WS-CAF, nemmeno si pone il problema. Di conseguenza, nessuna specifica fornisce meccanismi per negoziare differenti livelli di QoS a Runtime.

Tuttavia, in WS-Tx, si potrebbe utilizzare WS-Policy per pubblicizzare i livelli di QoS supportati da un certo servizio, mentre in BTP e WS-CAF sono forniti dei meccanismi di estensione che potrebbero essere utilizzati allo scopo. Inoltre, anche per la negoziazione, è possibile ottenerla facilmente grazie alla possibilità, offerta da tutte le Specifiche, di estensione del Context e dei messaggi di protocollo.

Estendibilità

Non era tra i nostri requisiti, ma, visto che tutte le soluzioni proposte forniscono meccanismi di estensione, e poiché questi potrebbero essere utilizzati per migliorare i protocolli offerti, è opportuno considerare la questione.

Tutte e tre le famiglie di Specifiche permettono l'estensione dei messaggi: BTP e WS-CAF mediante Qualificatori[129], WS-Tx mediante campi predefiniti e appositamente previsti in ogni messaggio. A prima vista potrebbe sembrare una caratteristica di pregio, poiché, grazie ad essa, si potrebbero recuperare eventuali mancanze nelle Specifiche. Questo è certamente vero quando esse siano utilizzate per relazioni intra dominio, ma non è necessariamente vero per quelle multi dominio. In queste altre, infatti, la caratteristica di estendibilità potrebbe contrapporsi a quella di interoperabilità, principio cardine dell'ambiente WebServices. Tuttavia, assumendo che i messaggi "estesi" siano sempre opportunamente standardizzati, non ci sarebbe nessun problema. Purtroppo le Specifiche non pongono limiti in proposito, e così chiunque potrebbe ritenere appropriato estendere i protocolli a proprio piacimento. Nelle relazioni multi dominio, invece, l'utente ha bisogno di protocolli **certi**, e pertanto avrà valore solo quanto correntemente definito. Nella valutazione condotta finora ci siamo posti proprio dal punto di vista dell'utente, e abbiamo considerato le Specifiche per quello che propongono, non per quello che "potrebbero" proporre.

[129] BTP definisce anche dei Qualificatori Standard molto utili.

WS-CAF e WS-Tx prevedono anche la possibilità che siano definiti nuovi protocolli. Per questo secondo tipo di "estendibilità" vale un discorso analogo al precedente, ma con qualche considerazione aggiuntiva. La "filosofia" adottata da WS-CAF e WS-Tx si basa sull'adozione di "molteplici" protocolli, ognuno rivolto ad uno specifico dominio applicativo e a specifiche esigenze. Pertanto, assumono che in futuro possano sorgere nuove esigenze, e dunque che nuovi protocolli transazionali diventino necessari. Per l'utente questo può essere un vantaggio, ma anche uno svantaggio. Infatti, l'utente vorrebbe evitare che eventuali cambiamenti apportati internamente ad un proprio servizio, risultino in una variazione dell'interfaccia da questo esposta. Adottando un modello unico, non si pone alcun problema. Con la filosofia "dei molteplici modelli", invece, è possibile che eventuali cambiamenti interni al servizio richiedano un modello differente da quello precedentemente adottato, e di conseguenza un cambiamento nell'interfaccia esposta, con tutto quello che consegue negli altri servizi che utilizzavano tale interfaccia. La stessa critica viene fatta anche dagli autori del gruppo Choreology[130].

La Tabella 4 riassume tutte le considerazioni precedentemente fatte. Essa mostra i requisiti soddisfatti in pieno o in parte da ogni modello trattato. Mostra anche i requisiti che sono soddisfatti solo per relazioni intra-dominio. Un requisito soddisfatto in questo modo, potenzialmente potrebbe essere soddisfatto anche per relazioni multi dominio, ma solo se richiedente e partecipanti siano "obbligati" a rispettare particolari vincoli.

Tale distinzione vale, in maniera particolare, per i requisiti relazionati ad Atomicità e Isolamento, essendo queste proprietà globali della transazione.

[130] La critica si può trovare sul Web all'indirizzo:
http://www.choreology.com/standards/standards_btm_WS-CA.htm

Tabella 4 Come i modelli soddisfano i requisiti

Modello / Requisito	BTP		WS-Tx		WS-CAF		
	Atom	Cohesion	AT	BA	ACID	LRA	BP
Composizione dinamica	-	√	-	√	-	√	√
Revoca di Partecipazione	√	√	√	√	√	-	√[131]
Atomicità	I	I	I	-	I	-	-
Rilassamento dell'Atomicità	√	√	√	√	√	√	√
Segnalazione risultato non atomico	I	I	I	I	I	I	-
Procedura di Annullamento	-	-	-	-	-	I	I
Contratti Vincolanti relativi all'Atomicità	-	-	P[132]	NA	-	NA	NA
Isolamento	I	I	√	-	I	-	-
Rilassamento Isolamento	√	√	-	√	√	√	√
Contratti Vincolanti relativi all'Isolamento	-	-	P[133]	NA	-	NA	NA
Disponibilità[134]	-	-	-	√	-	-	-
Overbooking con Notifica	-	P[135]	-	√	-	-	-
Interposizione[136]	√	√	√	√	√	√	√
mista	√	√	-	-	-	-	√
Conoscenza dello "stato" di elaborazione dell'esecutore[137]	P	P	-	P	P	P	P
Conoscenza dello "stato" di composizione del richiedente	√	√	P[138]	√	√	√	√
Recovery anche nella fase di Composizione	-	-	-	√	-	-	√
Protocolli Efficienti[139]	√	√	√	√	√	√	√
Comunicazioni Sicure	-	-	√	√	-	-	-
Molteplici modelli per la Sicurezza	-	-	√	√	-	-	-
Comunicazioni Affidabili	√	√	√	√	√	√	√
Compatibilità tra Protocollo Business e tecnologia dei data-base[140]	P	P	√	-	√	-	-
Modello per Transazioni ACID per Web	√	-	√	-	√	-	-
QoS a Runtime (A-I-S-Overbooking-etc.)	-	-	-	-	-	-	-

Legenda:
I = requisito soddisfatto solo per relazioni intra dominio
√ = requisito soddisfatto anche per relazioni multi dominio
P = come √, ma requisito soddisfatto solo parzialmente
NA = requisito Non Applicabile

[131] Solo con arruolamento di tipo "Push".
[132] Solo per l'Atomicità.
[133] L'isolamento è vincolato solo per le operazioni di scrittura, ma non anche per quelle di lettura.
[134] Si considera supportata se sono supportati contemporaneamente i requisiti: Comunicazioni Sicure, Rilassamento Isolamento, Overbooking delle risorse Semplice e con Notifica.
[135] Supportato con una soluzione non del tutto efficiente.
[136] Nessuna delle Specifiche fornisce delle agevolazioni per supportare *deadlock strutturali* di tipo *permanente*.
[137] Tutti prevedono la sola interrogazione del richiedente, ma non anche la notifica autonoma dell'esecutore.
[138] Non è prevista l'interrogazione da parte del partecipante al richiedente.
[139] Le comunicazioni sono considerate Efficienti se è supportata almeno l'Interposizione.
[140] In BTP il requisito può essere ritenuto soddisfatto solo quando sia utilizzato, per l'implementazione degli "Effects", l' "approccio tipico dei data base".

3.5 Problemi irrisolti

In Tabella 5 è possibile osservare i requisiti che non sono supportati, o sono supportati solo in parte, dai modelli precedentemente analizzati rispetto al nostro modello di riferimento, che abbiamo chiamato “Modello Ottimo”.

Per il Modello Ottimo, abbiamo ipotizzato che esso sia composto da due modelli distinti, per transazioni ACID e Business rispettivamente, in maniera simile a quanto hanno fatto WS-Tx e WS-CAF. Questa ipotesi potrebbe certamente essere rimossa riunendo i due modelli in un modello unico, ma, a nostro parere, sarebbe un’inutile forzatura, dato il differente utilizzo per il quale si propongono, e i diversi requisiti cui devono rispondere. Con due modelli separati, inoltre, si eviterebbe la “confusione” nell’utilizzatore, citata in “1.2.1 Two Phase Commit e transazioni distribuite”, circa il protocollo 2PC e l’isolamento.

Inoltre, per il modello Ottimo Business, abbiamo assunto che sia supportata l’Atomicità, la Segnalazione di risultato non atomico, la Procedura di Annullamento, e l’Isolamento, solo per relazioni intra dominio, e che questi stessi requisiti siano estesi a garanzie per relazioni multi dominio proprio grazie alla definizione di opportuni Contratti vincolanti. Per il modello Ottimo ACID, invece, è sufficiente che i requisiti di Atomicità e Isolamento siano supportati solo per relazioni intra dominio, poiché qualsiasi violazione sarebbe una scelta “interna” all’Organizzazione. Nel paragrafo “1.4.2.4 Supporto per transazioni tradizionali” abbiamo infatti assunto che questo tipo di transazioni siano utilizzate sempre in un contesto di fiducia “precostruito”.

Come si può vedere, nessuno dei modelli riesce a soddisfare in pieno i requisiti che volevamo, seppure i modelli proposti per transazioni ACID, e Atom di BTP, si avvicinino molto al nostro modello ACID di riferimento. Le differenze più importanti si trovano invece rispetto al modello Business, in particolar modo per i modelli che seguono l’approccio compensativo. Questi ultimi, infatti, non si limitano a rilassare in maniera irreversibile l’isolamento, ma scelgono di rilassare totalmente anche

l’atomicità, ovvero proprio quelle due caratteristiche che normalmente si richiede siano supportate da un modello per transazioni distribuite.

Tabella 5 Soddisfacimento dei requisiti rispetto al "Modello Ottimo".

Modello / Requisito	BTP		WS-Tx		WS-CAF			Ottimo	
	Atom	Cohesion	AT	BA	ACID-Tx	LRA	BP	ACID	Business
Composizione dinamica	-	√	-	√	-	√	√	-	√
Revoca di Partecipazione	√	√	√	√	√	-	√	√	√
Atomicità	I	I	I	-	I	-	-	I	I
Rilassamento dell'Atomicità	√	√	√	√	√	√	√	I	√
Segnalazione risultato non atomico	I	I	I	I	I	I	-	I	I
Procedura di Annullamento	-	-	-	-	-	I	I	-	I
Contratti Vincolanti relativi all'Atomicità	-	-	P	NA	-	NA	NA	-	√
Isolamento	I	I	√	-	I	-	-	I	I
Rilassamento Isolamento	√	√	-	√	√	√	√	I	√
Contratti Vincolanti relativi all'Isolamento	-	-	P	NA	-	NA	NA	-	√
Disponibilità	-	-	-	√	-	-	-	-	√
Overbooking con Notifica	-	P	-	√	-	-	-	-	√
Interposizione	√	√	√	√	√	√	√	√	√
mista	√	√	-	-	-	-	√	-	√
Conoscenza dello "stato" di elaborazione dell'esecutore	P	P	-	P	P	P	P	√	√
Conoscenza dello "stato" di composizione del richiedente	√	√	P	√	√	√	√	√	√
Recovery anche nella fase di Composizione	-	-	-	√	-	-	√	-	√
Protocolli Efficienti	√	√	√	√	√	√	√	√	√
Comunicazioni Sicure	-	-	√	√	-	-	-	√	√
Molteplici modelli per la Sicurezza	-	-	√	√	-	-	-	√	√
Comunicazioni Affidabili	√	√	√	√	√	√	√	√	√
Compatibilità tra Protocollo Business e tecnologia dei data-base	P	P	√	-	√	-	-	√	√
Modello per Transazioni ACID per Web	√	-	√	-	√	-	-	√	-
QoS a Runtime (A-I-S-Overbooking-etc.)	-	-	-	-	-	-	-	-	√

Riprendiamo ora una questione lasciata in sospeso durante la definizione e specifica dei requisiti. Non abbiamo infatti specificato nessun requisito sull' *indipendenza* del servizio di coordinazione. Abbiamo detto che questo potrebbe essere incluso nel servizio del richiedente, o che potrebbe essere implementato come un servizio autonomo, indifferentemente.

Potrebbe sembrare, dunque, che ciò sia una pura preferenza implementativa: in realtà, come vedremo tra breve, in molti casi la possibilità di implementare il coordinatore come servizio autonomo rappresenta una estrema convenienza, o addirittura una necessità.

In proposito, riconsideriamo lo scenario "1.3.1 Arranging a Night-Out", e supponiamo che il requisito di Atomicità sia vincolato da *contratto*. Se dunque il coordinatore coincidesse con il richiedente, quest'ultimo, essendo il TM, sarebbe vincolato dall'atomicità al pari dei servizi esecutori. Se ora l'applicazione utente risiedesse sul PDA di questo, ciò comporterebbe notevoli inconvenienti, poiché, solitamente, le connessioni instaurate con questo tipo di dispositivi sono poco stabili. Senza contare che tale connessione potrebbe anche essere molto costosa, e così, in generale, l'utente non desidererà mantenerla fino al completamento della transazione. La possibilità di riconnettersi in un secondo momento per verificarne l'esito, sarebbe certamente preferita.

La soluzione al problema non è ovviamente unica. Basterebbe infatti che richiedente e coordinatore, pur coincidendo, risiedano su un server intermediario, al quale l'utente potrebbe accedere mediante il proprio PDA e un sito Web intermediario. Ma, chiaramente, si perderebbe la potenzialità di comporre servizi Web direttamente attraverso un'applicazione residente sulla macchina dell'utilizzatore ultimo (come mostra il Deployment Diagram in Figura 86), piuttosto che tramite un ulteriore servizio Web accessibile via Internet (come mostrato in Figura 87).

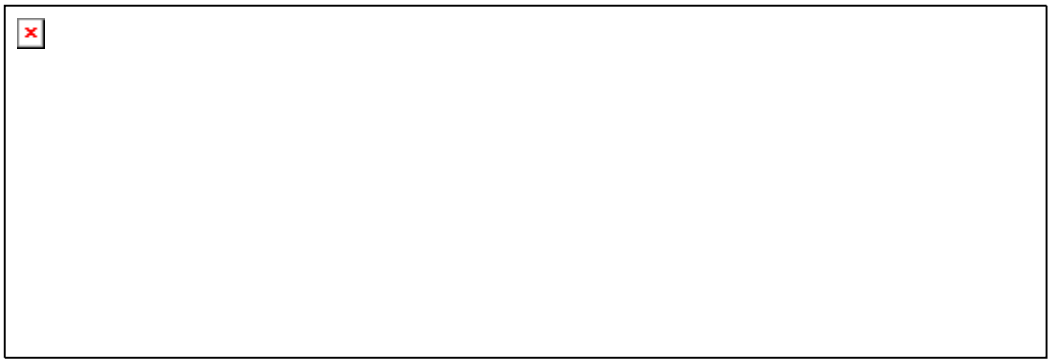

Figura 86 Esempio di Composizione direttamente sul Client

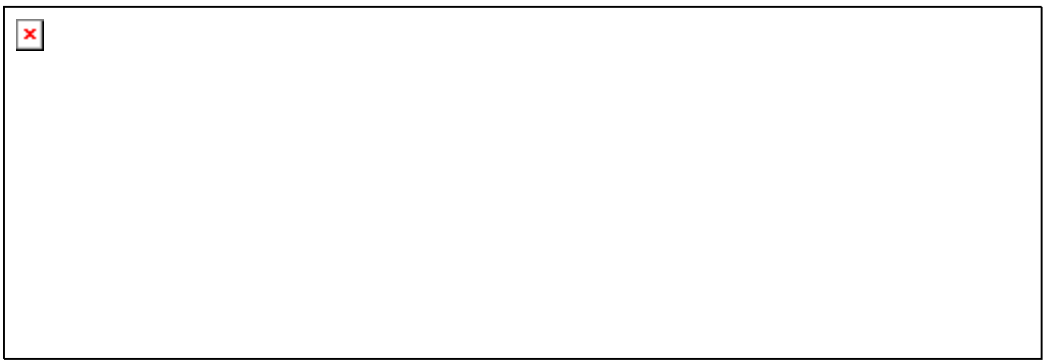

Figura 87 Esempio di Composizione tramite servizio *intermediario*

Un altro motivo per cui può essere necessario un coordinatore indipendente, è strettamente legato alle conseguenze che discendono dal vincolare, contemporaneamente, atomicità e isolamento. Ricordiamo che, in queste ipotesi, una eventuale caduta del TM per una certa durata comporta il blocco delle risorse degli esecutori per la stessa durata. Pertanto, difficilmente un partecipante accetterebbe un contratto con vincoli siffatti. Quanto detto risulta vero, a maggior ragione, nelle ipotesi che il partecipante sia costretto ad affidarsi ad un servizio coordinatore implementato sulla macchina del generico richiedente.

Un servizio esecutore che sottoscriva un contratto di questo tipo vorrà garantirsi, infatti, quantomeno richiedendo l'utilizzo di un coordinatore di propria fiducia. E' dunque possibile immaginare scenari in cui esistano delle società fornitrici esclusivamente di servizi di coordinazione, opportunamente certificati, che possano dare le dovute garanzie. Ad esempio, il Deployment Diagram in Figura 88 mostra un servizio coordinatore che, per fornire le garanzie volute, utilizza un nodo di backup, che

potrebbe essere gestito, ad esempio, implementando il 2PC a quattro fasi. In caso di problemi al nodo primario, inoltre, sarebbe indispensabile il meccanismo di Redirection fornito da BTP, o un meccanismo equivalente, per istruire i vari partecipanti del "cambio" di coordinatore.

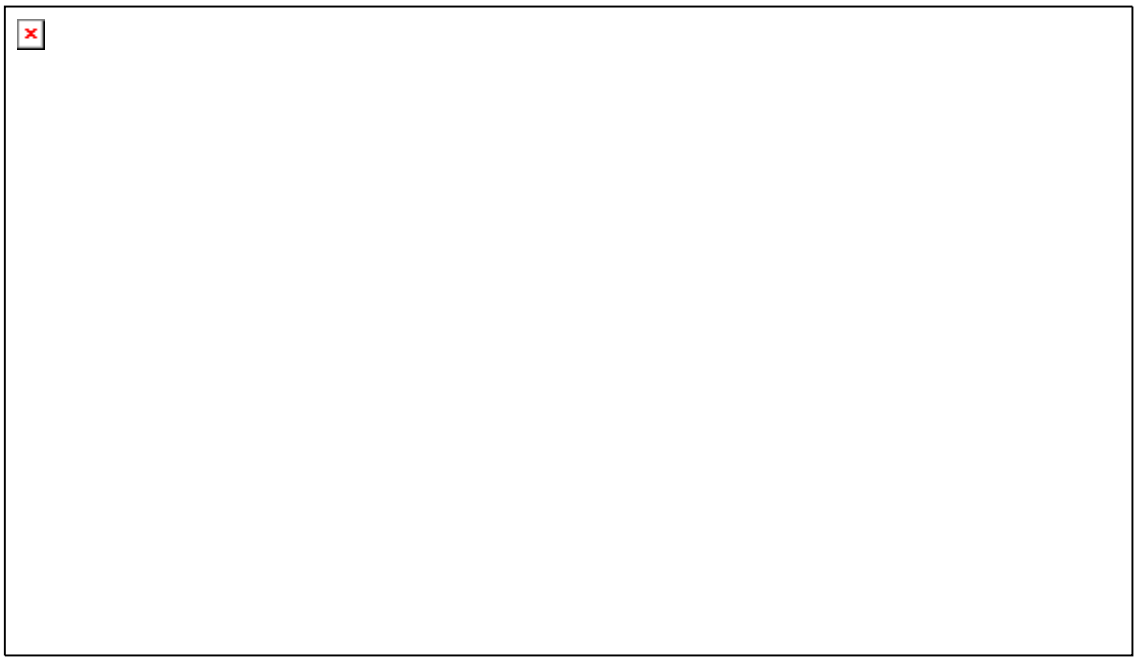

Figura 88 Esempio di servizio di coordinazione autonomo.

In definitiva, sarebbe desiderabile che il modello di coordinazione soddisfasse l'ulteriore requisito:

RC7. Coordinatore indipendente

Molto spesso l'utente finale dei servizi Web interagirà direttamente con servizi intermediari (ad esempio tramite siti Web), che realizzeranno la composizione transazionale di altri servizi. In altri casi utilizzerà, invece, un'applicazione locale alla propria macchina, e nelle ipotesi di atomicità vincolata da *contratto*, è desiderabile che il richiedente non sia vincolato dal dover mantenere la connessione fino alla risoluzione di eventuali eventi di *failures*. Analogamente, sempre in caso di atomicità vincolata da contratto, per un servizio esecutore sarebbe desiderabile potersi affidare a servizi di coordinazione di fiducia, eventualmente gestiti da società terze e autonome, in grado di fornire le dovute garanzie.

Si richiede, pertanto, che il servizio di coordinazione sia implementabile come servizio autonomo, e dunque che il modello di coordinazione per transazioni business definisca appropriati *protocolli di completamento*. Inoltre, per consentire la realizzazioni di servizi affidabili attraverso la ridondanza (come mostrato, ad esempio, in Figura 88), si richiede pure un meccanismo analogo a quello di Redirection definito da BTP.

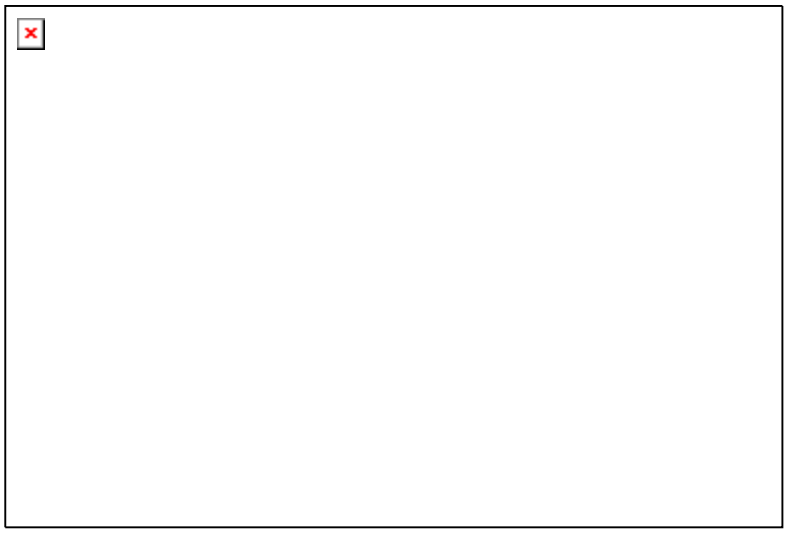

Figura 89

Abbiamo già osservato come l'unico modello che non prevede un protocollo di completamento è il modello BA. Tutti gli altri, invece, permettono l'implementazione del coordinatore come servizio autonomo, ma presentano alcuni inconvenienti comuni:

P1. Perdita *inconsapevole* di Atomicità

Con l'arruolamento di tipo "Pull", se, in seguito ad una richiesta applicativ,a il corrispondente servizio non dovesse arruolarsi, il richiedente dovrebbe avere consapevolezza dell'accaduto, per non rischiare di avviare una transazione con la mancanza di alcuni importanti partecipanti. Ovviamente, se il problema si presentasse, l'atomicità sarebbe certamente violata, e in maniera trasparente al richiedente. Tutti i modelli utilizzano una strategia di arruolamento di questo tipo, ma nessuno definisce un meccanismo che assicuri dal possibile inconveniente; il problema non si presenta solo nel modello BA, semplicemente perché il richiedente corrisponde con il coordinatore.

Coerentemente, tutti i modelli propongono anche l'atomicità rilassata, rendendo questo inconveniente accettabile. Esso non sarebbe accettabile, invece, se l'atomicità fosse vincolata da *contratto*.

P2. Irrecuperabilità della relazione richiedente-coordinatore

Tutti i modelli definiscono meccanismi per il recovery della relazione coordinatore-partecipante, ma nessun modello definisce analoghi meccanismi per la relazione richiedente-coordinatore. Questo, in caso di *failures*, si traduce nell'impossibilità, da parte del richiedente, di conoscere l'esito di una transazione precedentemente confermata. Quanto detto non è applicabile al modello BA, poiché esso prevede che richiedente e coordinatore coincidano.

P3. Terminazione delle attività di un servizio coordinatore autonomo

Se il coordinatore fosse implementato come un servizio autonomo (ad esempio perché gestito da una società terza rispetto a quelle di richiedente ed esecutori), e se questo servizio fosse utilizzato contemporaneamente da molti richiedenti e per molte transazioni, servirebbe un meccanismo per poterlo fermare, bloccando ulteriori richieste, ma garantendo che le transazioni avviate possano completare regolarmente.

Nessun modello fornisce un meccanismo di questo tipo. L'unica agevolazione fornita in tal senso, è costituita dal meccanismo di Redirection definito da BTP, con cui è possibile indirizzare i partecipanti coinvolti nelle transazioni già avviate verso una nuova istanza del servizio coordinatore, ma, per quest'ultimo, il problema si ripresenterebbe. Queste considerazioni non sono applicabili al modello BA, in cui non è prevista un'implementazione come servizio autonomo del coordinatore.

I tre problemi descritti si presentano, dunque, solo quando il coordinatore sia implementato come un servizio distinto da quello del richiedente, ipotesi, questa, che potrebbe essere particolarmente utile quando l'Atomicità sia vincolata da Contratto.

Tabella 6 Problemi in caso di coordinatore implementato come servizio autonomo

Modello / Requisito/Problema	BTP		WS-Tx		WS-CAF		
	Atom	Cohesion	AT	BA	ACID	LRA	BP
Coordinatore indipendente	√	√	√	-	√	√	√
Meccanismo di "Redirection"	√	√	-	-	-	-	-
Perdita *inconsapevole* di Atomicità	√	√	√	-	√	√	√
Irrecuperabilità della relazione richiedente-coordinatore	√	√	√	NA	√	√	√
Terminazione delle attività di un servizio coordinatore autonomo	√	√	√	NA	√	√	√

Legenda:
√ = problema presente
√ = requisito soddisfatto
NA = Non Applicabile

3.6 Possibili soluzioni e miglioramenti

I modelli analizzati possono essere modificati e/o estesi, spesso in maniera anche molto semplice, per soddisfare altri requisiti, oppure per soddisfare meglio dei requisiti soddisfatti solo parzialmente. Nel seguito di questo paragrafo proporremo le tecniche con cui ciò potrebbe essere ottenuto.

QoS a Runtime (A-I-S-Overbooking-etc.)

Per consentire al richiedente di conoscere i livelli di QoS supportati da un certo servizio esecutore, può essere utilizzata la famiglia di Specifiche WS-Policy. Un certo livello di QoS potrebbe essere richiesto mediante Qualificatori da aggiungere nei messaggi di richiesta applicativa. Eventualmente, si potrebbe prevedere che alcuni livelli di QoS siano richiesti come essenziali, mentre altri come desiderabili. In questo caso il messaggio di risposta potrebbe trasportare i Qualificatori corrispondenti ai livelli di QoS, tra quelli desiderabili, accettati dal partecipante.

Queste considerazioni sono applicabili a tutti i modelli, poiché tutti prevedono l'estendibilità dei messaggi, e tutti possono beneficiare di quanto già definito in WS-Policy.

Comunicazioni Sicure e Molteplici modelli per la Sicurezza

Per garantire la sicurezza delle transazioni, è possibile utilizzare le stesse Specifiche utilizzate dalla soluzione WS-Transactions, cioè le Specifiche della famiglia WS-Security. Queste, infatti, supportano molteplici modelli per la sicurezza, e sono facilmente integrabili in qualsiasi protocollo transazionale.

Procedura di Annullamento

Si potrebbe definire un modello tipo WS-CAF, ma è molto più semplice aggiungere una ulteriore coppia di messaggi alla fine del normale protocollo di coordinamento,

cancel e *cancelled*, validi solo se supportati dal particolare esecutore, così come indicato nella sua policy.

Contratti Vincolanti relativi all'Atomicità e Contratti Vincolanti relativi all'Isolamento

Per quanto riguarda l'atomicità, alcuni contratti che potrebbero essere definiti sono:

A1. nessuna garanzia di Atomicità (contratto di default);

A2. Garanzia di segnalazione per risultato non atomico;

A3. Garanzia di Atomicità;

A4. garanzia rilasciata solo con Coordinatore Certificato; un servizio che supporti questo *contratto* dovrà indicare, nella sua policy, le "certificazioni" accettate.

A5. disponibilità della Procedura di Annullamento: un servizio che supporti questa procedura dovrà indicare, nella sua policy, entro quanto tempo questa potrà essere utilizzata dall'eventuale conferma della transazione.

Sono valide anche le accoppiate: A2+A4, A3+A4. Inoltre, sia le accoppiate dette, sia A1, A2 e A3, potrebbero essere concesse con l'aggiunta di A5.

Per l'Isolamento, indipendentemente dalla politica utilizzata internamente al servizio per isolare le scritture, si potrebbero prevedere Contratti a partire dai comuni livelli supportati dai DBMS:

I1. Uncommitted read;

I2. Committed read;

I3. Repetible read;

I4. livello supportato con "Notifica di Inconsistenza".

Il Contratto I4 prevede che una eventuale inconsistenza sia segnalata al richiedente, mediante un messaggio opportunamente definito. Può essere supportato insieme ai Contratti I2 e I3.

Ogni contratto dovrebbe avere una validità almeno pari al *tempo di vita*[141] della transazione, come comunicato nel Context inviato insieme alla richiesta applicativa, e di conseguenza questo dovrebbe essere anche il minimo tempo di attesa in caso di mancanza di comunicazioni.

Ovviamente, ogni contratto dovrebbe contemporaneamente indicare anche tutti i vincoli di protocollo da rispettare.

Per sottolineare ancora una volta la stretta dipendenza di questo tipo di vincoli dal protocollo, consideriamo i modelli che i produttori rivolgono ai servizi Business, ovvero BA, LRA e BP, e per BTP scegliamo Cohesion, visto che Atom non supporta la Composizione Dinamica.

Valutiamo prima il problema dell'Atomicità, ricordando preliminarmente che comunque sia vincolata, non sarà più possibile la revoca da parte di un partecipante tra le fasi di Prepare e Commit equivalenti, almeno per il tempo di vita della transazione. Per questo motivo potrebbe essere utile, in caso di problemi, una comunicazione della situazione dal partecipante al coordinatore/richiedente, con una stima del tempo atteso per la *ripresa*. In proposito può essere molto conveniente la strategia adottata da BTP, che permette ad ogni partecipante di avere un set di indirizzi a cui rispondere, piuttosto che un singolo indirizzo come accade negli altri modelli.

In Cohesion, per via della Composizione Dinamica, la fase di Prepare può essere eseguita un numero arbitrario di volte durante la composizione. Ammettere che ciò sia possibile in caso di Atomicità vincolata, significa ammettere che i partecipanti perdano il loro potere decisionale per tutta la durata della composizione. Poiché questo sarebbe inammissibile, vincolare l'Atomicità in Cohesion, significherà certamente rivedere il significato della fase di Prepare, poiché si dovrà imporre che questa sia eseguita una sola volta, e contemporaneamente per tutti i partecipanti. Chiaramente il rispetto di tale imposizione non sarebbe facilmente verificabile, nemmeno se il coordinatore fosse implementato come servizio indipendente. Ecco perchè, almeno in quest'ultimo caso, si potrebbe richiedere esplicitamente che il servizio coordinatore sia gestito da società autonome e sia opportunamente *certificato* da società di fiducia.

[141] Individuato dal Qualificatore standard TransactionTimelimit in BTP, dal campo Expires in BA, dal campoc ExpiredTimeout In LRA e BP.

Vincolare l'Atomicità in BA significa prima di tutto stabilire le informazioni da mantenere persistentemente affinché ciò sia possibile. A tal proposito, essendo il modello "compensativo", potrebbe essere utilizzato il modello speculare a quello dell' "Abort Presunto", ovvero il modello del "Commit Presunto". Ora, se è vero che la fase di Prepare equivalente non potrà essere ripetuta con lo stesso partecipante come in Cohesion, è anche vero che, essendo il richiedente coincidente con il coordinatore, questa fase potrebbe essere avviata indipendentemente per ogni partecipante, e in momenti differenti. Pertanto, per non ricadere nello stesso difetto di Cohesion, si dovrà imporre che la fase di Prepare equivalente sia avviata contemporaneamente per tutti i partecipanti. Purtroppo, per BA, non è stato definito un protocollo di completamento, e così il coordinatore non può essere implementato come servizio autonomo.

Anche per i modelli LRA e BP si dovrebbero definire, prima di tutto, le informazioni da mantenere persistentemente. Inoltre, data la particolare strategia che utilizzano per supportare la Composizione Dinamica, che richiede un numero di coordinatori pari al numero di operazioni componenti la transazione, si presenta lo stesso problema di BA, aggravato dal fatto che, evidentemente, non si potranno vincolare più coordinatori indipendenti ad eseguire contemporaneamente la fase di Prepare equivalente.

Quando si voglia vincolare la Segnalazione di Risultato non atomico, si presenta un ulteriore e importante problema, questa volta indipendentemente dal particolare protocollo di coordinamento utilizzato. Supponiamo infatti che il servizio esecutore, per garantire la Segnalazione di Risultato non atomico, sia vincolato ad inviare un messaggio di *fault* al coordinatore. Essendo possibili fallimenti di comunicazione e/o di nodo, affinché tale servizio abbia certezza che il messaggio spedito arrivi a destinazione, si potrebbe prevedere un semplice messaggio di riscontro da parte del coordinatore. Ma come potrebbe, il coordinatore, essere certo che tale riscontro venga consegnato all'esecutore? Un altro messaggio di riscontro?

Questo problema è del tutto analogo al problema[142], noto come "Problema dei Due Eserciti", che si presenta per determinare la chiusura di una connessione. Esso si presenta sempre quando esista un "ultimo messaggio", di importanza determinate, che debba attraversare un mezzo insicuro, e purtroppo, non ammette soluzione. L'unica

[142] Il "problema dei due eserciti" è spiegato in Riferimenti[13 p482-483].

soluzione, nel nostro caso, sarebbe richiedere che almeno uno dei due interlocutori rimanga attivo per sempre, in maniera tale che l'altro possa itterrogarlo in qualsiasi momento per ottenere il messaggio di riscontro, nel caso non sia riuscito ad ottenerlo in precedenza. Una soluzione di compromesso, ma tecnicamente realizzabile, consiste invece nell'ipotizzare che uno dei due interlocutori rimanga attivo per un tempo sufficientemente lungo[143] (ad esempio un giorno per transazioni che hanno durata dell'ordine delle ore). Tale ipotesi è praticamente sempre verificata per i servizi che operano tramite server Web, poiché questi, generalmente, saranno servizi sempre attivi, e pertanto sarà certamente verificata nel caso in cui sia utilizzato un coordinatore indipendente, fornito da società terze. Inoltre, per i casi estremi, si potrebbe prevedere l'utilizzo di sistemi ridondanti, e di mezzi di comunicazione alternativi, anche banali e semplici come il telefono, affinché sia sempre possibile comunicare eventuali imprevisti.

In ogni caso, sia il tempo di attesa del coordinatore, che le modalità di utilizzo di mezzi di comunicazione alternativi, dovranno essere previsti dalla Specifica, e negoziabili tra le parti mediante opportuni Qualificatori.

Passiamo all'Isolamento. L'unico modello in cui questo può essere vincolato è Cohesion, in quanto gli altri tre modelli presi in considerazione utilizzano l'approccio compensativo, e imporre un vincolo di questo tipo significherebbe stravolgere completamente lo loro la logica di funzionamento. Per Cohesion non ci sarebbero particolari problemi, poiché vincolare l'isolamento significherebbe semplicemente imporre l'utilizzo "dell'approccio tipico dei Data Base" per l'implementazione degli "Effects". In proposito si noti che, se il coordinatore fosse gestito da società terze come servizio autonomo, questo potrebbe sempre comportarsi in maniera tale da non rispettare la condizione necessaria data a pag 32, poiché potrebbe completare prima una sotto-transazione, e poi tutte le altre. Anche in questo caso il richiedente potrebbe perdere l'isolamento, e così sarebbe opportuno che il servizio coordinatore sia obbligato a rispettare detta condizione da contratto.

[143] Il tempo di attesa potrebbe essere stimato staticamente, oppure potrebbe essere calcolato dinamicamente. La stima dovrebbe essere proporzionata alla durata di un'interruzione di rete, di un congestionamento, del tempo di caduta di un nodo, e dovrà essere limitata inferiormente dal *tempo di vita* della transazione.

Interposizione mista

In WS-Tx non è permessa l'Interposizione mista. Di conseguenza non si può richiedere al Servizio di Attivazione una transazione AT come sotto-transazione di una transazione BA. Seppure nulla vieti che un partecipante funga da "ponte" per i due modelli, comportandosi da coordinatore *interposto*, questo rappresenterebbe un coordinatore non standard secondo la Specifica, e pertanto sarebbe opportuno rimuovere tale vincolo.

Problema dei deadlock strutturali di tipo permanente

Quando sia adottata la tecnica dell'Interposizione in ambiente multi-dominio, e quando l'isolamento comporti vincoli per il servizio esecutore, sarebbe desiderabile che eventuali deadlock strutturali di tipo permanente siano individuati e segnalati.

Consideriamo, in particolare, Cohesion, essendo l'unico modello dall'isolamento vincolabile tra quelli per Transazioni Business. Per l'individuazione del problema, potrebbe essere utilizzata la tecnica esposta a pag 85, ma si dovrebbe prima definire un identificatore univoco per ogni transazione. Per la segnalazione, invece, si potrebbe prevedere un apposito messaggio, o si potrebbe definire un opportuno qualificatore, eventualmente inseribile nel messaggio di *notPrepared equivalente* (*cancelled* in Cohesion), per comunicare il motivo del "non pronto". Lo stesso qualificatore potrebbe essere aggiunto nel messaggio, sull'esito della transazione, ritornato dal coordinatore al richiedente.

Conoscenza dello "stato" di elaborazione dell'esecutore

Tutti i modelli già prevedono la possibilità di interrogare un partecipante per conoscerne lo stato di elaborazione corrente. Per ottenere una maggiore efficienza (minimizzazione della *finestra di incertezza*), è sufficiente aggiungere un ulteriore messaggio, dall'esecutore al richiedente, eventualmente associabile alla risposta applicativa, con cui l'esecutore dichiara che ha terminato proprie le elaborazioni.

Recovery anche nella fase di Composizione

Potrebbe essere utilizzata la stessa strategia prevista da BA: “ogni cambiamento di stato deve essere preceduto da una memorizzazione permanente dello stato antecedente al nuovo”. Ma è sufficiente che lo “stato antecedente al nuovo” sia memorizzato permanentemente solo prima dell’invio di un messaggio alla controparte.

Utilizzando un servizio di questo tipo, il richiedente, se anche effettuasse più richieste applicative rivolte allo stesso partecipante, potrebbe semplicemente ripetere l’ultima richiesta per la quale non abbia ancora ricevuto una risposta. Tuttavia, poiché è sempre possibile che la memorizzazione abbia avuto qualche problema, è necessario prevedere almeno un ulteriore messaggio, con cui un servizio esecutore possa comunicare che l’intera sequenza deve essere ripetuta, o eventualmente il numero di sequenza da cui sia necessario ripartire.

Un servizio che supporti questa strategia per il recovery nella fase di Composizione, potrà semplicemente indicarlo nella propria policy.

Compatibilità tra Protocollo Business e tecnologia dei data-base

Per quanto riguarda l’approccio compensativo, nel capitolo 3.2 abbiamo visto che esso è interpretabile come un protocollo 2PC equivalente che applica il 2PL o il Locking per le scritture. Nel capitolo 3.1, con riferimento a queste tecniche, abbiamo implicitamente descritto come i modelli compensativi siano integrabili con le tecnologie dei DBMS: utilizzando due transazioni locali distinte, rispettivamente per effettuare le azioni richieste, e per annullarle/compensarle. Questo tipo di informazioni dovrebbero essere precisate nelle Specifiche dei protocolli.

Perdita *inconsapevole* di Atomicità

Il problema può essere evitato prevedendo un ulteriore messaggio, eventualmente associabile alla risposta applicativa (in maniera analoga a come viene già associato il Context alla richiesta), con cui il partecipante comunica al richiedente che ha effettuato

l'arruolamento con successo. Almeno i contratti che garantiscono Atomicità o Segnalazione di risultato non atomico dovrebbero obbligare il suo utilizzo.

Overbooking con Notifica

La tecnica dell'Overbooking con Notifica prevede la revoca di partecipazione nella fase di Composizione. Quando la revoca viene comunicata solo al coordinatore, potrebbe presentarsi ancora il problema della "Perdita *inconsapevole* di Atomicità". Tale problema non è presente in BA, essendo coordinatore e richiedente un'unica entità, mentre in BTP viene risolto con una tecnica poco efficiente: il richiedente deve comunicare al coordinatore gli identificativi dei partecipanti nel messaggio di conferma. In questo modo, se uno dei partecipanti appartenenti al Confirm-set avesse nel frattempo già revocato la propria partecipazione, la transazione sarebbe automaticamente abortita. Per minimizzare il numero di Abort, BTP prevede che il richiedente possa richiedere periodicamente al coordinatore lo stato dei partecipanti, ma, evidentemente, questa tecnica è poco efficiente, soprattutto per quelle applicazioni in cui lo stato delle risorse vari frequentemente (con conseguente alto numero di revoche).

Una tecnica molto più effiente consiste, invece, nel prevedere un opportuno messaggio con cui il coordinatore comunichi subito, e spontaneamente, eventuali revoche di partecipazione al richiedente. Questa informazione, così come accade in THP, potrebbe anche essere scambiata direttamente tra partecipante ed esecutore, senza l'intermediazione del coordinatore.

Disponibilità

Per incrementare la Disponibilità dei servizi, senza perdere completamente l'Isolamento, potrebbe essere applicata la tecnica dell'Overboking con Notifica anche rispetto ai dati, come descritto in "3.3 Tecnica dell'Overbooking", ovvero ritardando l'esecuzione delle operazioni di scrittura, e concedendo letture esterne (quelle visibili all'utente) con Contratti I2+I4, I2+I3+I4 (letture eseguite internamente a livello L1), o I3+I4 (letture eseguite internamente a livello L2).

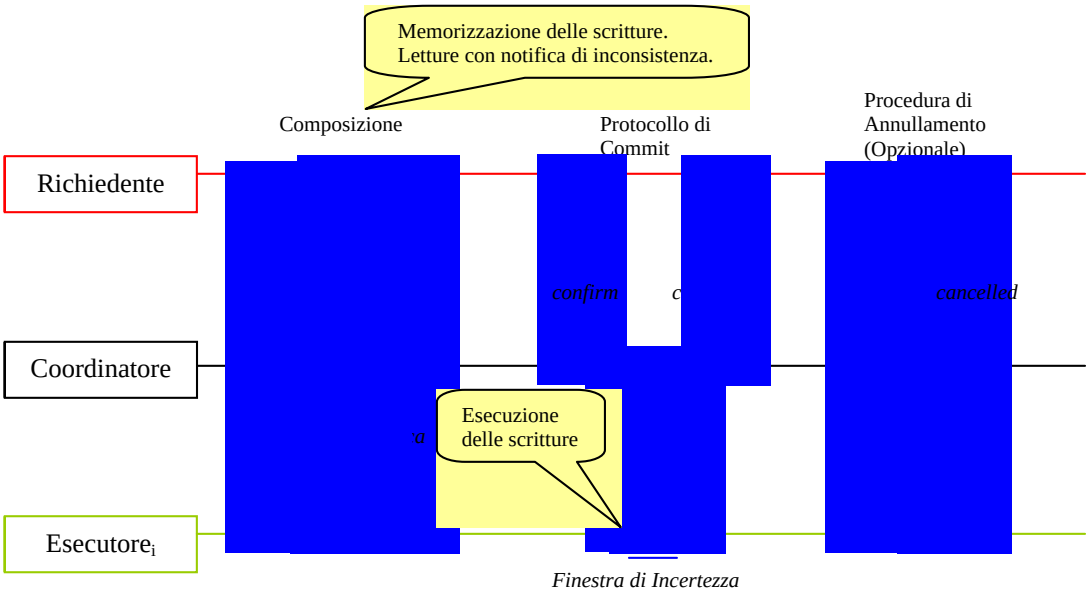

Figura 90 Linee essenziali del protocollo Ottimo Business

Irrecuperabilità della relazione richiedente-coordinatore

Il problema si presenta perché, in tutti i modelli proposti, il coordinatore è tenuto a mantenere persistentemente solo informazioni che riguardano il protocollo di Commit, e non anche il protocollo di Completamento.

La soluzione consiste dunque nel semplice utilizzo di un file di log, associato ad una coppia di messaggi di richiesta-risposta, con cui sia possibile interrogare il coordinatore circa una transazione di cui non si conosca il risultato. In proposito a questa questione, va menzionato ancora una volta il meccanismo di Redirection definito da BTP, mediante il quale si potrebbero indirizzare tutte le richieste per transazioni già concluse verso un'entità separata, utilizzata con quest'unico scopo.

Nelle ipotesi in cui siamo, ovvero di coordinatore implementato come servizio autonomo, è anche necessario prevedere un'opportuna *assertion*, da inserire nella policy del coordinatore, che indichi entro quanto tempo, oltre il termine della transazione, sarà possibile richiedere informazioni a riguardo. Per avere una maggiore efficienza, si può prevedere un messaggio di riscontro dal richiedente al coordinatore, ricevuto il quale le informazioni possono essere cancellate anche prima del tempo stabilito.

Inoltre, prevedere la presenza di un file di log quando il coordinatore sia gestito da società certificate, potrebbe costituire un elemento ulteriore cui fare riferimento in caso di contenziosi.[144]

Terminazione delle attività di un servizio coordinatore autonomo

Per prevenire il problema, si può definire un opportuno Qualificatore standard, che indichi per quanto tempo il coordinatore sia disposto a gestire una nuova transazione. Oltre questo tempo, il coordinatore potrà decidere arbitrariamente se terminare la transazione in Abort, o continuare nella normale esecuzione. Il Qualificatore in questione potrebbe essere parte del Context ritornato all'atto della creazione, in maniera tale che sia conosciuto sia dal richiedente che da tutti i partecipanti. In associazione dovrebbe essere previsto un ulteriore Qualificatore (il *tempo di vita* della transazione, che abbiamo già utilizzato precedentemente), da utilizzare come segue: il richiedente, nel messaggio di creazione di una nuova Transazione, utilizzerà questo secondo Qualificatore per indicare il tempo per il quale si aspetta la disponibilità del coordinatore. Se il coordinatore non è disponibile per il tempo richiesto, dovrà rifiutare la creazione, e dovrà rispondere con un opportuno messaggio inserendovi il Qualificatore che indica la durata massima della propria disponibilità. Poiché la durata di tali Qualificatori è espressa in secondi, in generale si presenta il problema del ritardo introdotto dalla comunicazione, che in caso di congestioni può produrre un notevole sfasamento. Per ovviare all'inconveniente, i servizi potrebbero comunicare un tempo assoluto, piuttosto che una durata, ma, in tal caso, si presenterebbe l'ulteriore problema[145] della *sincronizzazione degli orologi*, che tuttavia potrebbe essere risolto in maniera semplice quando sia sufficiente una sincronizzazione approssimata.

Una soluzione alternativa consiste nel vincolare i servizi ad interpretare le durate ricevute mediante Qualificatori, rispetto al momento esatto in cui è stato inviato l'ultimo messaggio in senso inverso. Così si otterrebbero delle approssimazioni per eccesso, ma

[144] Lo stesso suggerimento è stato dato da WS-CAF, come descritto in "2.3.2.1.1 Possibili estensioni future di WS-CF" a pag211.

[145] Si noti che le Specifiche utilizzano già "Qualificatori di tipo durata", poiché tutte definiscono un *tempo di vita* per una transazione, che viene comunicato ai partecipanti tramite la condivisione del Context. Tuttavia, il problema della *sincronizzazione degli orologi* non si presenta, poiché le durate sono solo indicative, e atomicità e isolamento sono rilassati.

si eviterebbero tutti i problemi di sincronismo. Tuttavia, quanto detto, non è applicabile per i primi messaggi scambiati in ogni direzione.

CONCLUSIONI

Vogliamo iniziare questo capitolo conclusivo ponendoci una precisa domanda: le Specifiche valutate, secondo quanto stabilito a pag 90, sono delle *buone Specifiche*?

BTP, nel complesso, risulta una Specifica ben fatta e precisa, e si distingue dalle altre due concorrenti per l'approccio seguito nella definizione della soluzione, che è rivolto al "contratto" tra le parti, così come discende dalle definizioni di "Relazione Business" e "Transazione Business" che propone. Tuttavia, tale approccio rimane un'intenzione, poiché abbiamo visto che il "contratto" definito da BTP lascia massima libertà alle parti, e così, di fatto, è come se non ci fosse. Gli unici vincoli dati da BTP sono rivolti essenzialmente all'implementazione del coordinatore, che in Atom è obbligato a trattare tutti i partecipanti in maniera atomica.

Valutandolo dal punto di vista della soluzione, un'importante pregio è rappresentato dall'ottimo **modello per il recovery**, con cui è possibile garantire transazioni consistenti, o segnalazione di inconsistenze dovute a decisioni autonome. Tale capacità ha un'enorme importanza, soprattutto in relazione alla elevata inaffidabilità del mezzo di "trasporto" più comune utilizzato dai servizi Web: Internet. Quello delle "decisioni autonome", poi, in accordo al requisito di **autonomia dei partecipanti**, costituisce, evidentemente, un notevole pregio del protocollo, almeno quando non sia vincolata l'atomicità. Altrettanto importante è il meccanismo della Redirection, utilissimo in moltissime circostanze. Un considerevole difetto, invece, è la mancanza di un qualche meccanismo che supporti realmente il requisito della **Disponibilità** degli *esecutori*. Infatti, l'unica agevolazione, che potrebbe essere considerata in questa direzione, è rappresentata dalla possibilità, offerta alle Implementazioni, di poter scegliere come implementare gli Effects. A questo si associa la mancanza di supporto per la Sicurezza, che molto probabilmente sarà trattata nelle prossime versioni della Specifica.

Anche **WS-Tx** è una Specifica ben fatta, almeno dal punto di vista del documento, seppure, qualche punto, avrebbe potuto essere meglio esplicitato. E' molto concisa, forse fin troppo, poiché la presenza di qualche esempio di utilizzo dei protocolli che

definisce, ne avrebbe certamente agevolato la comprensione. Alcuni esempi possono essere trovati in altri documenti[146] ancora di origine Microsoft-IBM, ma, almeno quelli da noi valutati, sono stati di scarsa utilità, non essendo del tutto coerenti con quanto definito nella Specifica.

Per quanto riguarda la soluzione proposta, una lode è certamente meritata per la completezza, soprattutto in riferimento all'intero stack di protocolli definiti per WebServices, con cui riesce a coprire molte delle problematiche presenti nel particolare ambiente. Abbiamo più volte osservato circa la **mancanza di un *Protocollo di Completamento*** per WS-BusinessActivity. In realtà è possibile interpretare tale mancanza come una precisa scelta degli autori, giustificata dal fatto che, essendo per una Attività Business possibile la selezione dinamica dei partecipanti, e non essendo supportata l'Atomicità, l'utilizzo di un coordinatore autonomo rappresenta, in tal caso, un' "inutile" intermediazione.

WS-CAF, nata in seguito, si presenta come un "super-set" delle due precedenti Specifiche: BTP e WS-Tx. Come si può dedurre dai diversi modelli UML che abbiamo fornito nel presente testo, propone una soluzione particolarmente complicata rispetto alle concorrenti. La soluzione è stata definita suddividendola su tre livelli distinti (uno per Specifica), in maniera davvero molto discutibile. Il livello di WS-CTX, per condividere informazioni e per definire un Contesto, non giustifica la stesura di un intero documento di tale complessità (ben 60 pagine contro le 20 di WS-Coordination). Per di più, non specifica, ad esempio, come il ContextService debba comportarsi rispetto alla collezione di ALSes in caso di *failures*.

Il livello di WS-CF, con le sue 63 pagine, definisce essenzialmente un modello di inviluppo per protocolli che saranno definiti altrove, con la speranza, forse, che questo contribuisca ad aumentare l'interoperabilità, come può dedursi dal fatto che gli autori cercano di supportare anche Modelli di coordinazione forniti da terze parti. Eppure, sarebbe opportuno chiedersi quale vantaggio possa comportare il meccanismo dell' "inviluppo", piuttosto che l'utilizzo di semplici entità che fungano da intermediarie tra Modelli distinti, quando tale "inviluppo" contribuisca essenzialmente ad aumentare la complessità della Specifica, e gli sforzi per implementare servizi conformi ad essa.

[146] Alcuni esempi possono essere trovati in Riferimenti[28, 42].

Senza considerare la perdita di efficienza dovuta al fatto che un servizio, prima di poter interpretare i messaggi del particolare protocollo utilizzato, dovrà interpretare quelli definiti da WS-CF.

Passando al livello di WS-TXM, in ben 111 pagine, questo definisce, oltre al modello per transazioni ACID, i due modelli *compensativi* LRA e BP, offrendo come unica caratteristica aggiuntiva rispetto a BA di WS-Tx, un buon supporto per il recory.

WS-CAF si propone come successore di WS-Tx e così, ad esempio, il protocollo Synchronization definito da WS-TXM corrisponde in pratica al protocollo Volatile2PC definito da WS-AT, ma si propone anche come successore di BTP, dal quale non riprende, invece, l'utilissimo meccanismo della Redirection. In totale, oltre 200 pagine, per una soluzione del tutto analoga, a meno del supporto per la Sicurezza, a quella offerta da WS-Tx in 60 pagine. Se, almeno, a questa maggiore lunghezza corrispondesse una maggiore precisione e chiarezza … e invece, WS-CAF, come documento di Specifica, lascia molto a desiderare. In diverse parti è poco chiaro, utilizza nomi di entità non definiti in nessun luogo, o utilizza come sinonimi nomi di entità definite come distinte. Inoltre, spesso sembra essere incompleto, e per quanto riguarda il modello BP, relativamente alla specificazione del ruolo di Terminator, cade addirittura in contraddizione[147]. In generale è imprecisa e soggetta a molteplici interpretazioni: WS-CTX, nonostante la notevole lunghezza del documento, definisce la possibilità di innestare Attività, ma cosa comporti l'applicazione di questo concetto non è mai spiegato esplicitamente: deve essere estrapolato dalla specificazione dei protocolli, così come abbiamo fatto noi alla fine del paragrafo "2.3.1.1 Modello Concettuale". L'utilizzo di LRA innestate, anzi, introduce una certa indeterminatezza: è possibile innestare Attività gestite da ContextServices indipendenti? Il ContextService del cliente è tenuto ad arruolare un'ALS del servizio provider? Tutto è possibile, poiché ben poco è specificato in proposito. WS-CF definisce il concetto di "subordinazione", ma non chiarisce la relazione con il concetto di "Attività innestate" dato in WS-CTX. WS-TXM, infine, definisce in pratica un modello 2PC e due modelli "compensativi". Per quanto riguarda questi ultimi due, l'unica differenza sostanziale consiste nel fatto che BP permette la composizione mista di LRA e Transazioni ACID, mentre il modello LRA consente la composizione solo di LRAs. Per il resto, il modello Business Process

[147] Vedi nota 118 a pag.253.

rappresenta un ulteriore protocollo basato ancora una volta sul modello della compensazione, ma con un supporto per il recovery potenziato.

Alcune idee sono certamente meritevoli, come i molteplici spunti che fornisce WS-TXM relativamente alle modalità per condurre il recovery, ma le modalità con cui sono state riportate in un documento di Specifica sono davvero pessime. Qualsiasi lettore avrebbe difficoltà a comprendere appieno, se mai sia possibile, i modelli in essa definiti, quando risulti conveniente utilizzarli, e quali vincoli realmente comportino. Lo stesso modello concettuale presentato in questa tesi, nonostante la sua semplicità (ovvero nonostante rappresenti solo aspetti essenziali), è stato ottenuto con notevoli difficoltà.

Infine, dal punto di vista "contrattuale", c'è ben poco da valutare. Forse gli autori volevano fornire un framework per il controllo di flusso di applicazioni distribuite, piuttosto che un modello per Transazioni Business: per sperare di ottenere una buona strutturazione dell'applicazione, infatti, i servizi dovranno essere progettati seguendo una logica unica, una sola mano e una sola mente. Caratteristica, questa, in completo contrasto con l'interoperabilità richiesta dai servizi Web nell'ambito delle Transazioni multi-dominio.

THP è una Specifica molto buona: concisa, precisa, esaustiva, e ottimamente corredata da esempi. Come soluzione propone in pratica l'utilizzo l'*Overbooking con Notifica*, grazie al quale molti servizi Business potrebbero incrementare la propria Disponibilità, anche in transazioni di lunga durata, ma senza penalizzare l'utilizzatore. Associato opportunamente ad un protocollo *2PC equivalente*, permetterebbe di minimizzare la *finestra di incertezza*, poiché sposta la fase di composizione dinamica a monte del Protocollo di Commit. Purtroppo non è utilizzabile in maniera standard con le altre Specifiche. Inoltre, con alcuni modelli per Transazioni Business, seppure implicitamente, l'*Overbooking con Notifica* può già essere applicato: basterebbe riconoscere i servizi che adottano tale politica, ad esempio mediante WS-Policy.

Secondo quanto stabilito a pag 90, una *buona Specifica* avrebbe dovuto rispettare i requisiti fissati in "1.4 Analisi e specifica dei Requisiti", proponendo una soluzione semplice, con una complessità proporzionata al problema. Avrebbe dovuto, cioè, stabilire esattamente le entità in gioco, che nel caso specifico sarebbero potute essere semplicemente tre (coordinatore, richiedente ed esecutore), l'interfaccia tra queste

(significato e sintassi dei messaggi scambiati), e il comportamento atteso in seguito allo scambio dei messaggi.

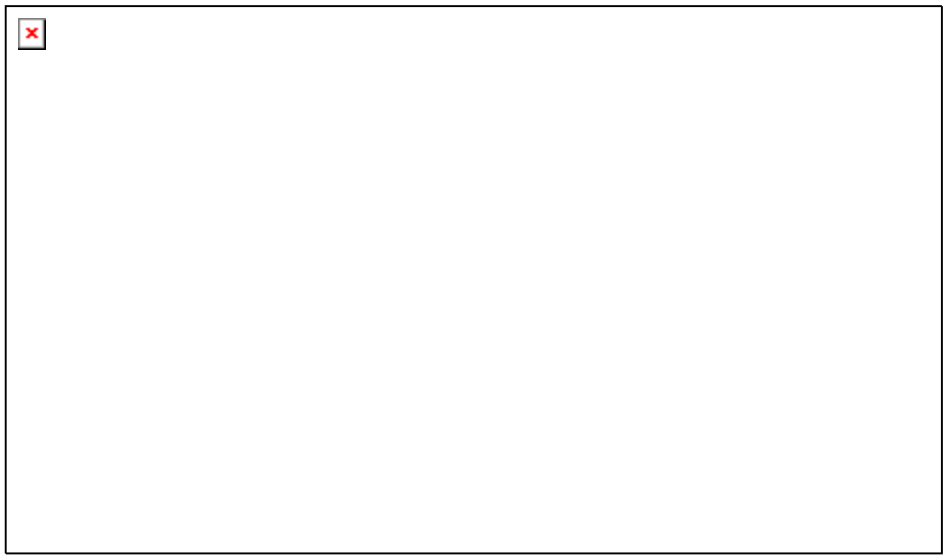

Figura 91 Esempio di “Semplice Modello Concettuale”

Una buona Specifica avrebbe dovuto essere, prima di tutto, un buon documento di specifica, dunque corretto, completo, non ambiguo.

Una buona Specifica avrebbe dovuto fornire una *base contrattuale*, per supportare generici contratti Business. Il comportamento interno delle entità sarebbe dovuto essere specificato sempre con un “may”, e mai con un “must”. Ad esempio, quello delle informazioni da mantenere persistentemente, è un problema del singolo partecipante, sia che queste riguardino l’applicazione, sia che riguardino il protocollo di coordinamento. Una Specifica si sarebbe dovuta limitare a consigliare cosa memorizzare affinché il recovery sia possibile. Questo, ad esempio, è quello che in parte fa BTP. In parte perché, nella sezione normativa del documento, obbliga, ad esempio, un Superior ad accertarsi che sia memorizzata la decisione di Confirm, prima di inviare l’omonimo messaggio agli inferiori.

Le Specifiche valutate, invece, non rispettano molti dei requisiti che abbiamo individuato in questa tesi, e per quanto riguarda WS-CAF, questa definisce un modello molto complesso rispetto a quello che realmente offre. Inoltre tutte presentano imprecisioni e incompletezze, specialmente WS-CAF, e non definiscono nessuna “base contrattuale”.

Evidentemente, la risposta alla nostra domanda iniziale non può che essere: non sono delle *buone Specifiche*!

Tuttavia, le Specifiche valutate propongono tutte un buon modello per *Transazioni ACID per Web*, Sicurezza a parte, che per il momento è trattata solo da WS-Transactions. Secondo noi, lo stesso non si può dire, invece, riguardo i modelli per Transazioni Business, seppure questo fosse il loro scopo principale. Del resto, le uniche vere novità che propongono per supportare Transazioni di questo tipo, sono quella dell'Approccio Compensativo e quella della Composizione Dinamica, rilassando tuttavia Atomicità e Isolamento. Certo, l'Approccio Compensativo ha il suo campo di applicazione, ma ignorando i requisiti di atomicità e isolamento, lo stesso risultato si sarebbe potuto ottenere in maniera molto più semplice: sarebbe bastato standardizzare il Context e utilizzare WS-Policy per individuare i servizi che supportano la *compensazione*. In tal caso, infatti, il coordinatore risulta essere un intermediario completamente superfluo, limitandosi a consegnare messaggi di *close* o di *compensate* ad ogni partecipante, secondo quanto indicato dal *richiedente*. Risulterebbe pertanto più efficiente che sia il *richiedente* stesso a comunicare direttamente tali messaggi (come accade in BA). Inoltre, non risulta giustificata nemmeno l'introduzione di una Specifica che vincoli la sintassi del metodo di *compensate*: così come si richiede all'utente di dover conoscere il metodo della richiesta applicativa, così si potrebbe richiedere che conosca il corrispondente metodo di compensazione. In questo modo, ogni servizio esecutore potrebbe avere un metodo di "compensate" con una propria sintassi e con parametri specifici (al pari della richiesta applicativa), e potrebbe trarre vantaggio da questa personalizzazione. Sarebbe stato sufficiente, dunque, definire un meccanismo per identificare il particolare metodo di compensazione associato ad una certa richiesta applicativa, ad esempio, ancora utilizzando WS-Policy.

Nelle ipotesi di assenza di Atomicità e Isolamento, invece, secondo noi sarebbe stato molto più utile considerare l'Approccio Compensativo come un modello per standardizzare la Procedura di Annullamento: due sotto-transazioni ACID distinte, una per richiedere lavoro applicativo, e una, eventualmente, per richiederne successivamente l'annullamento. Così sarebbe stata garantita almeno l'Atomicità delle due sotto-transazioni (richiesta e annullamento), e per quanto riguarda l'Isolamento tra le due

operazioni, abbiamo già notato, durante l'analisi dei casi d'uso nel capitolo 1.3, che esistono particolari applicazioni in cui è lecito e conveniente trascurare tale proprietà.

In definitiva, possiamo accettare i modelli per Transazioni ACID, la complessità aggiunta, le imprecisioni e incompletezze in versioni non Standard, ma, a nostro parere, non è accettabile la totale mancanza di supporto per Atomicità e Isolamento, proprio lì dove tali proprietà sarebbero maggiormente necessarie, ovvero nelle Transazioni Business. Anzi, parlare di supporto Transazionale in queste ipotesi, ci sembra quasi una forzatura.[148]

Secondo noi, dunque, i modelli esaminati non sono sufficienti in ambito Business; in particolar modo non è sufficiente l'Approccio Compensativo, e a maggior ragione quando sia applicato rilassando totalmente anche l'Atomicità, oltre che l'Isolamento, ovvero le uniche due proprietà che rappresenterebbero un problema per qualsiasi transazione distribuita. Nel nostro percorso, infatti, supportare queste due caratteristiche, in un ambiente multi-dominio, ci ha condotto verso molte scelte di compromesso, ad esempio: per garantire contemporaneamente la Disponibilità dei servizi; per la definizione di opportuni "contratti", così come prevede il concetto stesso di Transazione ("prevenzione di liti"). E ci ha portato a valutare importanti problemi, quali il problema della *Sincronizzazione degli Orologi* e il *Problema dei due Eserciti*.

Pertanto, sebbene molte applicazioni business intra-dominio, o multi-dominio (li dove atomicità e isolamento non rappresentano un problema), possano beneficiare del supporto "transazionale" offerto, le applicazioni che hanno realmente necessità di supporto transazionale, nel vero senso della parola, dovranno ancora attendere. Per le altre, l'unica cosa che ci sentiamo di dire è che, per coerenza e completezza, la Soluzione proposta da Microsoft-IBM è quella che, allo stato attuale, ci è sembrata migliore.

A questo punto, date le critiche che abbiamo portato alle attuali Proposte, sembrerebbe naturale continuare questo lavoro di tesi proponendo un nuovo modello, che racchiuda tutti i pregi di quelli analizzati, ne elimini i difetti e, sulla base di quanto

[148] Questo è il motivo principale per cui il lavoro della presente tesi parte dalla definizione di Transazione.

descritto in “3.6 Possibili soluzioni e miglioramenti”, si avvicini il più possibile a quello che abbiamo indicato come il “Modello Ottimo”.

Fortunatamente per il lettore annoiato, l’ambizioso obiettivo dell’autore della presente tesi di fornire “un’alternativa alle proposte attuali”, è già stato raggiunto, e la nostra “alternativa” è rappresentata proprio dall’applicazione di tutti i suggerimenti proposti nel corso di questo testo.

Tuttavia, sarebbe aspicabile che tali suggerimenti siano presi in considerazione nella stesura di una nuova Specifica, compito dal quale ci riteniamo esonerati. Dal lavoro presentato fino a questo momento, infatti, una cosa risulterà certamente chiara al lettore: non può esistere supporto per Transazioni Business, nel mondo dei servizi Web, che non sia uno standard. Lasciamo dunque a chi di dovere questo compito, e così, senza peccare di incompletezza, e nella speranza di essere stati utili a qualcuno, possiamo ritenere questa tesi conclusa, o quasi.

“Quasi” perché dobbiamo ancora fornire risposta ad una domanda: come mai, in tanti anni di sviluppo, non è stato prodotto ancora nessun risultato concreto?

Probabilmente ciò potrebbe essere dovuto ai lunghi tempi, quasi sempre necessari per qualsiasi standardizzazione; ma, francamente, cinque anni ci sembrano un poco troppi. Oppure potrebbe essere conseguenza delle molteplici divergenze di opinioni, come discende dal fatto che i vari produttori sono partiti insieme, per poi passare a sviluppare prodotti propri; ma le soluzioni, nella sostanza, ci sembrano molto simili.

La risposta, forse, è proprio quella più scontata: come noi, anche i potenziali clienti rimandano *feedback* negativi ai produttori, e così, questi ultimi stanno studiando soluzioni migliori!

Purtroppo, data la situazione attuale, è lecito supporre che dovremo attendere ancora a lungo prima che queste “soluzioni migliori” diventino qualcosa di concreto. Nel frattempo continueranno a proliferare ancora soluzioni proprietarie, e magari, come già altre volte è capitato nell’imprevedibile mondo dell’informatica, una di queste, dandoci ragione, diventerà il nuovo *Standard de Facto* per WebServices.

RIFERIMENTI

Bibliografia Essenziale (comprende le Specifiche principali):

1. Atzeni, Ceri, Paraboschi, Torlone, *Basi di Dati*, 2° edizione 1999
   McGraw-Hill

2. OASIS Business Transactions Technical Committee, *Use Cases for the Business Transaction Protocol*, 3-5-2001
   http://www.oasis-open.org/cover/BTP-UseCases00000.pdf

3. OASIS, *Business Transaction Protocol version1.0.9.5, BTP 1.1 Working Draft 05*, 9-11-2004
   http://xml.coverpages.org/BTPv11-200411.pdf

4. OASIS Standard 200401, *"WS-Security"*, marzo 2004
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

5. Microsoft, IBM, BEA Systems, *"WS-Coordination"*, novembre 2004
   http://www-128.ibm.com/developerworks/library/specification/ws-tx/

6. Microsoft, IBM, BEA Systems, *"WS-AtomicTransaction"*, novembre 2004
   http://www-128.ibm.com/developerworks/library/specification/ws-tx/

7. Microsoft, IBM, BEA Systems, *"WS-BusinessActivity"*, novembre 2004
   http://www-128.ibm.com/developerworks/library/specification/ws-tx/

8. Oracle, Sun, Iona, Arjuna, Fujitsu, *Web Services Context (WS-Context)*, 28-6-2003
http://www.oracle.com/technology/tech/webservices/htdocs/spec/WS-CTX.pdf

9. Oracle, Sun, Iona, Arjuna, Fujitsu, *Web Services Coordination Framework (WS-CF)*, 28-6-2003
http://www.oracle.com/technology/tech/webservices/htdocs/spec/WS-CF.pdf

10. Oracle, Sun, Iona, Arjuna, Fujitsu, *Web Services Transaction Management (WS-TXM)"*, 28-6-2003
http://www.oracle.com/technology/tech/webservices/htdocs/spec/WS-TXM.pdf

11. Jerry Roberts (Intel), Krishnamurthy Srinivasan (Intel), *Tentative Hold Protocol Part 1: White Paper*, 28 novembre 2001
http://www.w3.org/TR/tenthold-1

12. Jerry Roberts (Intel), Tim Collier (Intel), Pallavi Malu (Intel), Krishnamurthy Srinivasan (Intel), *Tentative Hold Protocol Part 2: Technical Specification*, 28 novembre 2001
http://www.w3.org/TR/tenthold-2

Approfondimenti:

13. Andrew S. Tanenbaum, *Reti di Computer*, 2° edizione (italiana), Jackson

14. Nicola Zingarelli, *Vocabolario della lingua italiana*, 12° edizione1996

15. Theo Haerder and Andreas Reuter, *Principles of Transaction-Oriented Database Recovery*, December 1983
ACM Computing Surveys, 287—317

16. OpenNetwork, Layer7, Netegrity, Microsoft, Reactivity, IBM, VeriSign, BEA Systems, Oblix, RSA Security, Ping Identity, Westbridge, Computer Associates, *"WS-SecureConversation"*, maggio 2004
http://msdn.microsoft.com/ws/2004/04/ws-secure-conversation/

17. OpenNetwork, Layer7, Netegrity, Microsoft, Reactivity, VeriSign, IBM, BEA Systems, Oblix, RSA Security, Ping Identity, Westbridge, Computer Associates, *"WS-Trust"*, maggio 2004
http://msdn.microsoft.com/ws/2004/04/ws-trust/

18. Microsoft, IBM, Sun Microsystems, BEA, SAP, *WS-Addressing*, 10-8-2004 (submission W3C)
http://msdn.microsoft.com/ws/2004/08/ws-addressing/

19. VeriSign, Microsoft, Sonic Software, IBM, BEA Systems, SAP, *"WS-Policy"*, settembre 2004
http://msdn.microsoft.com/ws/2004/09/policy/

20. Microsoft, IBM, BEA, SAP, *WS-PolicyAssertions*, 28-5-2003
http://msdn.microsoft.com/ws/2004/09/policyattachment/

21. VeriSign, Microsoft, Sonic Software, IBM, BEA Systems, SAP, *WS-PolicyAttachment*, settembre 2004
http://msdn.microsoft.com/ws/2004/09/policyattachment/

22. RSA Security, VeriSign, Microsoft, IBM, *WS-SecurityPolicy*, 18-12-2002
http://msdn.microsoft.com/ws/2005/07/ws-security-policy/

23. (Request for Comments: 2371) J. Lyon (Microsoft); K. Evans, J. Klein (Tandem Computers); *Transaction Internet Protocol Version 3.0*, giugno 1998
http://www.ietf.org/rfc/rfc2371.txt

24. DevelopMentor, IBM, Microsoft, Lotus Development Corp., UserLand Software, Inc., *Simple Object Access Protocol (SOAP) 1.1*, 8-5-2000 (W3C Note)
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

25. Microsoft, IBM, Sun Microsystems, Canon, *SOAP Version 1.2 Part 1: Messaging Framework*, 24-6-2003 (W3C Recommendation)
http://www.w3.org/TR/soap12-part1/

26. Microsoft, IBM, Sun Microsystems, Canon, *SOAP Version 1.2 Part 2: Adjuncts*, 24-6-2003 (W3C Recommendation)
http://www.w3.org/TR/2003/REC-soap12-part2-20030624/

27. The Internet Society, *Uniform Resource Identifiers (URI): Generic Syntax*, agosto 1998
http://www.ietf.org/rfc/rfc2396.txt

Letture:

28. Mark Little(HP), Thomas J Freund (IBM), *A comparison of Web services transaction protocols*, 7-10-2003
http://www-128.ibm.com/developerworks/webservices/library/ws-comproto/

29. Mark Little (HP), *A framework for implementing business transactions on the Web*, 2001
http://www.hpmiddleware.com/downloads/PDF/business_transactions_on_the_web.pdf

30. Doug Kaye, *Web-Services Transactions, From Loosely Coupled - The Missing Pieces of Web Services*, 2003
http://xml.sys-con.com/read/43755.htm

31. Mark Little, *Transactions and Web Services*, ottobre 2003

32. Luis Felipe Cabrera, George Copeland, Jim Johnson, David Langworthy; (Microsoft Corporation); "*Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity*", 28 gennaio 2004
http://msdn.microsoft.com/webservices/understanding/specs/default.aspx?pull=/library/en-us/dnwebsrv/html/wsacoord.asp

33. Microsoft, IBM, "*Secure, Reliable, Transacted Web Services: Architecture and Composition"*, settembre 2003
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

34. Scott Seely (Microsoft Corporation), "*Understanding WS-Security*", ottobre 2002
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

35. Benchaphon Limthanmaphon, Yanchun Zhang, ”*Web Service Composition Transaction Management*”, 2004
http://citeseer.ist.psu.edu/cache/papers/cs/30400/http:zSzzSzcrpit.comzSzconfpaperszSzCRPITV27Limthanmaphon.pdf/web-service-composition-transaction.pdf

36. John McDowall, *Long Lived Transactions - the future of e-commerce?*, 29-9-2002
http://www.mcdowall.com/webservices/2002_09_29_archive.htnl

37. Dr. Srinivas Padmanabhuni (Technical Architect, Infosys Technologies Ltd), “*Web services transactions standards: Core requirements*”, 10 Jul 2003
http://searchwebservices.techtarget.com/originalContent/0%2C289142%2Csid26_gci913977%2C00.html

38. Priya Dhawan, Tim Ewald, Microsoft Developer Network, “*ASP.NET Web Services or .NET Remoting: How to Choose Building Distributed Applications with Microsoft .NET*”, September 2002
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdadotnetarch16.asp

39. Simon Maple, *Bridging the JTA-Web Services gap using WS-AtomicTransaction*, 27-1-2004
http://www-128.ibm.com/developerworks/webservices/library/ws-transjta/

40. Mike Lehmann, “*Developer WEB SERVICES, Who Needs Web Services Transactions?*”, novembre 2003
http://www.oracle.com/technology/oramag/oracle/03-nov/o63dev_web.html

41. Tom Freund(IBM), Tony Storey (IBM), *Transactions in the world of Web services, Part 1*, agosto 2002
http://www.ibm.com/developerworks/webservices/library/ws-wstx1/

42. Tom Freund(IBM), Tony Storey (IBM), *Transactions in the world of Web services, Part 2*, agosto 2002
http://www.ibm.com/developerworks/webservices/library/ws-wstx2/

43. Henry Peyret, *Mission-Critical Web Services: Plan for Long-Running Transactions*, 18-3-2002
http://www.hpmiddleware.com/downloads/pdf/giga_report.pdf

44. Oracle, Sun, Iona, Arjuna, Fujitsu, *Web Services Composite Application Framework (WS-CAF) Ver1.0*, 28-6-2003
http://www.oracle.com/technology/tech/webservices/htdocs/spec/WS-CAF%20Primer.pdf

45. OASIS Business Transactions TC, *Business Transaction Protocol Primer*, 3-June-2002
http://www.oasis-open.org/committees/business-transactions/documents/primer/BTP_Primer_v1.0.20020605.pdf

46. Luis Felipe Cabrera, Christopher Kurt, Don Box, *An Introduction to the Web Services Architecture and Its Specifications*, ottobre 2004
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

47. IBM, Microsoft, *Federation of Identities in a Web Services World*, 8-6-2003
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

48. Microsoft Corporation, Microsoft's Vision for an Identity Metasystem, maggio 2005
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

49. Microsoft, IBM, *Reliable Message Delivery in a Web Services World: A Proposed Architecture and Roadmap*, 13-3-2003
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

50. Microsoft, IBM, *Security in a Web Services World: A Proposed Architecture and Roadmap*, 7-4-2002
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

51. Kim Cameron (Microsoft), *The Laws of Identity*, maggio 2005
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

52. Aaron Skonnard (Skonnard Consultino), *Understanding WS-Policy*, agosto 2003
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

53. David Chappell, *New Technologies Help You Make Your Web Services More Secure*, aprile 2003
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

54. Microsoft, BEA Systems, *XML, SOAP, and Binary Data*, 26-2-2003
http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx

Siti importanti:

Microsoft

http://msdn.microsoft.com/webservices/webservices/default.aspx

http://msdn.microsoft.com/webservices/understanding/default.aspx

Microsoft-Web Services Enhancements (WSE)

http://msdn.microsoft.com/webservices/webservices/building/wse/default.aspx

IBM

http://www-128.ibm.com/developerworks/library/specification/ws-tx/

IBM-WebSphere® Application Server Network Deployment

http://www-306.ibm.com/software/webservers/appserv/was/network/

Arjuna

http://www.arjuna.com/index.html

OASIS

http://www.oasis-open.org/cover/oasisBT.html

Choreology

http://www.choreology.com/downloads/documents_library.htm

Hewlett Packard, *HP web services transactions*

http://www.hpmiddleware.com/HPISAPI.dll/hpmiddleware/products/webservices_transactions/default.jsp

Oracle, *Web Services Composite Application Framework (WS-CAF)*

http://www.oracle.com/technology/tech/webservices/htdocs/spec/ws-caf.html

BEA Systems

http://www.bea.com/

IONA

http://www.iona.com/

W3C

http://www.w3.org/TR