



## **Indice degli argomenti**

<b>Indice degli argomenti .....</b>	<b>2</b>
<b>Indice delle figure .....</b>	<b>4</b>
<b>Indice delle tabelle .....</b>	<b>6</b>
<b>Ringraziamenti .....</b>	<b>7</b>
<b>1. Introduzione .....</b>	<b>8</b>
1.1. Organizzazione dei capitoli .....	9
1.2. Problematiche relative all'integrazione di sistemi IT prima dell'avvento delle SOA .....	10
1.3. Le SOA e la loro applicazione nel contesto dell'integrazione di sistemi informativi aziendali .....	14
<b>2. L'attività di brokering in una SOA .....</b>	<b>28</b>
2.1. Scopi e funzionalità di un sistema di brokering, in particolare relazione allo standard UDDI .....	28
2.2. I concetti di "broker centralizzato", "broker distribuito" e quello di "brokering di messaggi" .....	32
<b>3. Ambiente software e requisiti di esecuzione del sistema ADES .....</b>	<b>35</b>
3.1. Requisiti hardware .....	35
3.2. Sistemi operativi e runtime .NET supportati .....	36
3.3. Requisiti di rete .....	40
<b>4. Analisi, progettazione e sviluppo del sistema ADES .....</b>	<b>42</b>
4.1. Introduzione alla metodologia di sviluppo del sistema ADES .....	43
4.2. Dominio di utilizzo del sistema ADES .....	48
4.3. Il sistema ADES nella fase di Analisi funzionale .....	51

4.4.	La “dualità” del framework/environment ADES.....	62
4.5.	Il sistema ADES nella fase di Progettazione.....	63
<b>5.</b>	<b>Aspetti avanzati di progettazione e sviluppo del sistema ADES.....</b>	<b>92</b>
5.1.	Distribuzione “fisica” delle funzionalità del sistema ADES.....	92
5.2.	Publishing dei servizi tramite interfacce.....	94
5.3.	Metodi, pattern di utilizzo e schemi ricorrenti per l’accesso alle funzionalità del sistema ADES .....	97
5.4.	Gestione della sicurezza nel sistema ADES.....	105
<b>6.</b>	<b>Tools grafici e interfacce utente a corredo del sistema ADES .....</b>	<b>109</b>
6.1.	Una vista ad alto livello di astrazione del sistema ADES e delle applicazioni “Acheron” e “Limbo” .....	109
6.2.	La windows application Acheron .....	111
6.3.	La web application Limbo .....	116
<b>7.</b>	<b>Panoramica delle tecnologie di brokering presenti sul mercato.....</b>	<b>120</b>
7.1.	IBM WebSphere Business Integration Message Broker.....	120
7.2.	Microsoft Connected Services Framework.....	126
7.3.	Soluzioni BEA AquaLogic e SAP NetWeaver. ....	132
7.4.	Tavole riepilogative di confronto.....	135
<b>8.</b>	<b>Conclusioni .....</b>	<b>137</b>
	<b>Bibliografia .....</b>	<b>139</b>

## Indice delle figure

Figura 4-1 - Processo di sviluppo adottato per il sistema ADES .....	45
Figura 4-2 - Scenari di utilizzo del sistema ADES .....	49
Figura 4-3 - Scenario Intranet di impiego del sistema ADES.....	50
Figura 4-4 - Funzionalità principali del sistema ADES .....	53
Figura 4-5 - Funzionalità di service hosting del sistema ADES .....	54
Figura 4-6 - Supporto alla creazione dei servizi da parte del sistema ADES .....	57
Figura 4-7 - Funzionalità di service brokering del sistema ADES .....	58
Figura 4-8 - Funzionalità di gestione della cache dei servizi.....	61
Figura 4-9 - Relazioni tra servizio e il sistema ADES .....	64
Figura 4-10 - Tipo "ADES::Service" .....	66
Figura 4-11 - Sottosistemi del sistema ADES .....	68
Figura 4-12 - Gerarchia di ereditarietà delle classi ServiceInfo .....	70
Figura 4-13 - Tipo "LBServiceInfo" .....	71
Figura 4-14 - Gerarchia di eccezioni del sottosistema "ADES::ServicesCacheManagement" .....	72
Figura 4-15 - Classe "ServicesPool" .....	73
Figura 4-16 - Classe "PoolManager" .....	74
Figura 4-17 - Schema di funzionamento del namespace "ADES::ServicesCacheManagement" .....	76
Figura 4-18 - Architettura del sistema ADES .....	77
Figura 4-19 - Oggetto "Host" .....	78
Figura 4-20 - Oggetto "Broker" .....	79
Figura 4-21 - Gerarchia degli oggetti di gestione delle pools.....	80
Figura 4-22 - Oggetto "NetManager" ed enumerazione "NodeStatus" .....	81
Figura 4-23 - Servizio "RemotableBroker".....	83
Figura 4-24 - Gerarchia delle classi di pubblicazione dei servizi .....	86

Figura 4-25 - Accesso alle funzionalità di pubblicazione WSDL da browser.....	87
Figura 4-26 - Particolare del documento WSDL associato al servizio "RemotableBroker" e della stringa di connessione utilizzata .....	88
Figura 4-27 - Formato delle stringhe di connessione ai servizi .....	89
Figura 4-28 - Tipo "ADES::Configuration" .....	90
Figura 4-29 - Tipo "ADES::Utils" .....	90
Figura 4-30 - Vari oggetti di utilità del namespace "ADES" .....	91
Figura 5-1 - Distribuzione fisica del sistema ADES – Cartelle .....	93
Figura 5-2 - Distribuzione fisica del sistema ADES - Applicazioni e librerie.....	94
Figura 5-3 - Documentazione a corredo del framework ADES.....	98
Figura 5-4 - Classi per la crittografia delle informazioni di pubblicazione .....	105
Figura 5-5 - Meccanismi di security-policing nel sistema ADES.....	106
Figura 5-6 - Dettaglio dei meccanismi di security-policing del sistema ADES .	107
Figura 6-1 - Il sistema ADES e gli applicativi a corredo .....	110
Figura 6-2 - La system tray icon dell'applicativo Acheron.....	111
Figura 6-3 - Il menu disponibile per l'icona dell'applicativo Acheron.....	112
Figura 6-4 - Finestra principale dell'applicazione Acheron .....	113
Figura 6-5 - Finestra di configurazione del sistema ADES fornita dall'applicativo Acheron .....	114
Figura 6-6 - Finestre di personalizzazione dei servizi "hosted" e delle relative pools .....	115
Figura 6-7 - Console di debug e di report degli eventi del sistema.....	115
Figura 6-8 - Vista d'insieme della web application Limbo .....	117
Figura 6-9 - Pagina di controllo delle pools dell'oggetto "Host" .....	117
Figura 6-10 - Flusso di lavoro di modifica delle proprietà del sistema ADES attraverso Limbo .....	118
Figura 6-11 - Alert dialog visualizzato al tentativo di esecuzione di più istanze di Acheron .....	119

## **Indice delle tabelle**

Tabella 7-1 - Aspetti tecnici di confronto fra i due prodotti .....	135
Tabella 7-2 - Aspetti e scelte architettureali .....	136
Tabella 7-3 - Confronto logico-concettuale e aspetti relativi all'astrazione fornita all'utente.....	136

## **Ringraziamenti**

L'autore desidera inaugurare questo spazio dedicato ai ringraziamenti con uno particolarmente sentito all'indirizzo del Prof. Ing. Fenu, che è stato di grande aiuto in questo lavoro, concedendomi ampia libertà, preziosi consigli, numerose lezioni di buon senso e una passione per la professionalità che mi auguro di avere, almeno in parte, acquistato.

Inoltre non posso che essere riconoscente e debitore verso tutti gli amici, i parenti e le persone che ogni giorno (e specialmente in questo periodo) mi sopportano con infinita pazienza (e in questo caso nessun altro aggettivo potrebbe essere più calzante di “infinita”). A loro va un ringraziamento veramente speciale (e le scuse per non averli potuti citare tutti).

Infine, non certo ultime in ordine di importanza (semmai il contrario), vengono le persone alle quali devo di più.

Ai miei genitori va il ringraziamento più grande che potrei mai immaginare ma che non è comunque abbastanza per esprimere quanto sia loro riconoscente.

## 1. Introduzione

**Scopo del presente lavoro è la presentazione di un software distribuito atto a favorire integrazione e sviluppo di sistemi IT<sup>i</sup> aziendali in contesti eterogenei e orientati alla comunicazione in rete, che offra elevati livelli di astrazione e trasparenza all'utente finale, sia esso uno sviluppatore o l'amministratore di una rete potenziata per mezzo di tale software.**

ADES (Advanced Distributed Extensible Services), questo il suo nome, è stato realizzato e viene qui descritto con duplice intento: non solo si tratta di un'applicazione dalle ricadute pratiche non indifferenti ma dalla sua analisi si ottiene anche una valenza didattica dovuta alle tematiche delicate e di grande attualità affrontate.

ADES nasce infatti da considerazioni sulle ultime evoluzioni delle architetture SOA (Service Oriented Architectures), sulle attività di brokering di servizi, su quelle relative al brokering di messaggi e sull'analisi degli attuali standards nel panorama dei WebServices.

È quindi anzitutto necessario, ai fini della completezza della componente didattica di questo lavoro, fornire una panoramica esaustiva delle attuali tecnologie di integrazione di applicazioni, prendendo le mosse da quelle esigenze che hanno portato all'adozione di un nuovo standard architetturale orientato ai servizi.

L'analisi dei problemi connaturati a questo nuovo standard e di tools e applicazioni nati per supportarlo è da considerarsi assolutamente propedeutica alla presentazione del software stesso, del quale, sul versante sperimentale, si può affermare come l'obiettivo principale della progettazione e dello sviluppo sia stato quello di risolvere per l'appunto i problemi di integrazione dei quali soprattutto in questa introduzione (ma non esclusivamente) ci si accinge a dissertare.



## **1.1. Organizzazione dei capitoli**

Nel Capitolo 1 (questa Introduzione) si tratta dell'adozione del nuovo standard architetturale orientato ai servizi. Lo si fa in due distinti passaggi: nel primo si discute della situazione precedente all'avvento delle SOA ("Service Oriented Architectures"); nel secondo si analizzano invece i dettagli teorici dell'architettura orientata ai servizi e della loro applicazione a quello che è uno dei problemi più pressanti dell'attuale mondo dell'IT, l'integrazione di applicazioni.

Nel Capitolo 2, "L'attività di brokering in una SOA", si chiarisce il ruolo di un sistema di brokering all'interno di una SOA, in particolare relazione allo standard UDDI e al software ADES. Dopo un'introduzione generale si passerà all'analisi di concetti più avanzati quali brokering centralizzato, distribuito o di messaggi.

Nel Capitolo 3 si discute dell'ambiente runtime, del software e della stratigrafia delle macchine virtuali che permettono l'esecuzione del sistema ADES.

Lo si fa in due punti: il primo si occupa dei requisiti hardware, il secondo degli OS (Operating System) supportati.

Il Capitolo 4 ("Analisi, progettazione e sviluppo del sistema ADES") analizza nel dettaglio il software ADES e lo descrive da diversi punti di vista: si parte da un'introduzione alla metodologia di sviluppo con la quale è stato elaborato ADES, per arrivare al disegno dettagliato del modello che lo descrive.

Il Capitolo 5 ("Aspetti avanzati di progettazione e sviluppo del sistema ADES") tocca poi alcuni dettagli relativi ai flussi di lavoro che ADES offre, al meccanismo di pubblicazione dei servizi ADES e descrive un possibile modello di security-policing<sup>ii</sup> per il sistema.

Il Capitolo 6 (“Tools grafici e interfacce utente a corredo del sistema ADES”) descrive poi brevemente le applicazioni a corredo del sistema che hanno il compito di offrire un’interfaccia semplice e consistente all’input dell’utente.

Il Capitolo 7 (“Panoramica delle tecnologie di brokering presenti sul mercato”) si occupa invece del confronto fra alcuni tools commerciali già presenti sul mercato che hanno funzionalità assimilabili o paragonabili a quelle del software presentato in questo lavoro.

### **1.2. Problematiche relative all’integrazione di sistemi IT prima dell’avvento delle SOA**

Nell’era immediatamente successiva a quella dell’esplosione del mercato dell’ICT (Information and Communication Technology) i problemi di integrazione dei diversi sistemi informativi sono cresciuti silenziosamente ma inesorabilmente, ponendo oggi un serio ostacolo allo sviluppo delle strategie di molte compagnie pesantemente “IT-based”.

I diversi problemi relativi all’integrazione di applicazioni afferiscono a due categorie principali:

- EAI (Enterprise Application Integration)
- B2B (Business-to-Business integration)

In questa introduzione si presenterà una vista d’insieme di questi problemi conducendo un’analisi di relazioni e punti di contatto più o meno evidenti tra queste due sfaccettature della stessa medaglia e rimandando all’interessante verifica dei concetti teorici esposti condotta sia sul sistema ADES, sia su applicazioni simili già presenti sul mercato (e presentate nel Capitolo 7, a pagina 120).

Tale analisi focalizzerà su realtà aziendali di livello “medio-enterprise”, ma può essere agevolmente generalizzata a contesti medio-piccoli, conservando inalterati gran parte dei concetti esposti.

Una siffatta generalizzazione è possibile poiché le problematiche architetturali relative all'integrazione e al dialogo tra sistemi eterogenei ormai non riguardano più solamente quelle aziende il cui business sfocia naturalmente in un contesto “IT-based”, ma investono anche tutte le altre realtà del mercato, sia che si parli di “enterprise companies” dotate di un'estesa infrastruttura IT, sia che si considerino invece “middle companies”, basate meno pesantemente su un simile sub-strato tecnologico.

È infatti ormai evidente come sia impossibile ignorare il “mare di informazioni” nel quale ogni realtà commerciale che si affacci sul mercato si trova ogni giorno a vivere, indipendentemente dal target al quale essa si rivolga.

Tutto ciò ingenera nel tessuto aziendale più delicato, quello dedito agli scambi con l'esterno, un processo che si potrebbe definire di “osmosi planetaria”, determinato dalle dinamiche della globalizzazione e veicolato principalmente da Internet. Un processo al quale, beninteso, sarà sempre più difficile sottrarsi.

Se si limitasse l'analisi delle dinamiche di integrazione ad aziende che, pur vivendo a stretto contatto con la Rete, conservano comunque un confine concettuale-organizzativo ben definito tra quello che è il mondo esterno e quello che invece sta all'interno di esse, si perderebbe in partenza la possibilità di analizzare gran parte delle diverse situazioni che già oggi si presentano quando è la realtà stessa dell'azienda a essere frammentata, anche geograficamente, a distanze considerevoli (si pensi a titolo di esempio a pratiche quali l' “offshore outsourcing”<sup>iii</sup>, che stanno rapidamente diventando prassi comune).

In questi casi il problema dell'integrazione dei vari sistemi attraverso la Rete diventa serio e delicato (cosa questa quantomeno ironica, considerato che la Rete stessa è pensata per facilitare le comunicazioni, non per renderle difficoltose).

Nel corso degli anni diverse sono state le tecnologie proposte per ovviare a queste difficoltà comunicative: J2EE, CORBA, DCOM, XML-RPC e, nel vicino passato, una grande varietà di dialetti XML con i quali si è cercato di stabilire una certa “zona franca” a livello linguistico.

Purtroppo questo proliferare, anziché risolvere il problema, lo ha reso ancora più complicato, ottenendo l’effetto diametralmente opposto a quello che si era sperato e saturando il mercato di “non-standards” (che “standard” invece volevano essere) incompatibili e spesso in conflitto tra loro.

Le soluzioni finora proposte (un vasto ombrello di opzioni delle quali le succitate tecnologie rappresentano una minima parte) hanno fallito spesso a causa del loro focalizzarsi su una piattaforma software particolare, ignorando le esigenze di compatibilità, portabilità e semplicità.

A questo proposito si potrebbe obiettare che Java e anche XML-RPC potrebbero sembrare sufficientemente “portabili”; ciò non è completamente vero, perlomeno dal punto di vista dell’interoperabilità con altri sistemi che a breve chiariremo.

Non è infatti da considerarsi completa un’analisi che faccia riferimento solo ed esclusivamente ad un’eventuale difficoltà linguistica del dialogo. **Parte del problema è infatti riconducibile a un vero e proprio “rifiuto” della comunicazione.**

Questa situazione è stata favorita dalla visione di Internet come mera “bacheca elettronica” e non come mezzo **globale** volto allo scambio **attivo** di informazioni. Infatti, ciò che oggi viene veicolato in rete rappresenta per il 90% una presentazione, una “istantanea” (snapshot) dei dati, non i dati stessi.

Come corollario a queste considerazioni è poi doveroso focalizzare l'attenzione, piuttosto che sulla mentalità con la quale si guarda allo sviluppo di sistemi, su un aspetto invece tecnico di tale sviluppo: i paradigmi architetturali del client-server, del mainframe-computing e di altre tecnologie (tutte caratterizzate da una gestione fortemente centralizzata del controllo sulle informazioni) hanno quasi “esaurito” la loro spinta, il loro compito; il cambio di direzione che il mondo dell'IT impone oggi è invece legato alle metodologie di sviluppo “distribuite” e alla gestione sempre più decentralizzata di un sistema informatico richiesta da mercati quali quello dell'**e-Work**, dell'**e-Health**, e dell'**e-Government**, per non parlare poi delle ricadute nel mercato del **mobile** e del **wireless**.

Si consideri la possibilità offerta a un professionista di poter consultare e **lavorare** su proiezioni di mercato, ordini, dati sperimentali e quant'altro mentre si trova in un'altra città o in un altro continente rispetto all'ubicazione della propria organizzazione. In situazioni come queste il “distribuirsi” di un sistema informativo è chiaramente l'unica direzione possibile.

Un simile discorso diventa ancora più valido quando si pensa a dinamicità ed eterogeneità come **fattori caratterizzanti** dell'ambiente di applicazione del sistema informativo e non come eventualità meramente incidentali ad esso.

**Di fatto, per una compagnia moderna (ambiente dinamico ed eterogeneo per eccellenza), la semplicità offerta dalla Rete per realizzare gli scambi di informazioni tra le varie parti che la compongono è un fattore chiave per acquisire un consistente vantaggio di mercato.**

Spostando poi il problema dell'integrazione e della comunicazione **dal mondo esterno nel quale un'impresa vive alle dinamiche interne che essa sviluppa** si può analizzare la sempre più diffusa situazione che si viene a formare quando sistemi “legacy”<sup>iv</sup> completamente diversi tra loro (e molto probabilmente operanti su piattaforme software estremamente differenti) si trovino “costretti” a scambiare informazioni in formati non compatibili che essi non comprendono.

Costretti a un dialogo “forzato” al fine di gestire il processo produttivo, tali sistemi lo frammentano in una serie di operazioni pulviscolari, ciascuna delle quali viene eseguita da sub-unità differenti, con dati che passano più tempo in conversioni di formato piuttosto che in vere elaborazioni.

La causa principale di tutto ciò è che tali “sotto-sistemi” (“silos”, come vengono spesso definiti dalla letteratura sull’argomento) sono troppo spesso isolati, semplici “black-boxes” dedite solo al ciclo di “input-elaborazione-output” e non a un dialogo che potrebbe rendere molto più efficiente ed efficace il loro lavoro.

L’introduzione di un middleware<sup>v</sup> che agisca come “membrana” attivamente coinvolta nel processo di traduzione e mediazione tra i sistemi può sì mitigare questi problemi di integrazione, ma non risolverli.

Quando anche si avesse l’impressione che i problemi siano stati risolti, non si dovrà far altro che attendere un tempo arbitrariamente lungo fino al loro ripresentarsi, magari a causa dell’aggiunta di nuovi sistemi non afferenti al campo d’azione del middleware, che dovrà così essere riadattato o, nel peggiore dei casi, ricostruito da zero.

### **1.3. Le SOA e la loro applicazione nel contesto dell’integrazione di sistemi informativi aziendali**

Nel taccuino dei “chief architects” di svariate compagnie IT la task di priorità maggiore è quindi rapidamente diventata portare il dialogo tra i vari comparti aziendali evitando però quanto più possibile di sacrificare quell’infrastruttura informatica fatta di sistemi “legacy” della quale essi sono già dotati.

Il completamento di una tale “migrazione” comporterebbe riduzioni sui tempi e sui costi dell’integrazione (si parla in alcuni casi del 40% della spesa del comparto IT dell’azienda), portando allo stesso tempo nell’infrastruttura IT aziendale “agilità” e flessibilità tali da permettere l’implementazione di nuovi sistemi, o la sostituzione di quelli legacy, evitando il conflitto con quelli precedenti e minimizzando la portata di eventuali operazioni di ri-sincronizzazione e ri-taratura dell’insieme risultante (per uno studio sulle tematiche dell’adozione di soluzioni SOA in ambiti “real-world” si consulti [SMRW]<sup>vi</sup>).

Se ne deduce che la soluzione a un simile gap integrativo debba essere adatta ad ambiti eterogenei e frammentati, e volta ad assicurare flessibilità e scalabilità.

Ergo, ciò di cui oggi hanno bisogno le aziende “al loro interno” è **un’architettura che renda possibile il dialogo tra i sistemi legacy, i nuovi sistemi che vengono sviluppati e quelli che ancora dovranno svilupparsi** (si noti comunque come sia implicita l’importanza della scelta di una “lingua franca” per realizzare una siffatta comunicazione).

I vincoli ai quali un’architettura che intenda assolvere a questo compito deve sottostare sono essenzialmente tre:

- Efficiency (“Efficienza”)
- Responsiveness (“Reattività”)
- Adaptability (“Adattabilità”)

Gli obiettivi che poi è naturale porre sono:

- Permettere all’infrastruttura IT un più efficiente “service delivering” (l’attività di esposizione dei servizi offerti all’esterno)
- Permettere un più rapido “riallineamento” dell’infrastruttura alle esigenze di business
- Mascherare la complessità tecnico-tecnologica dell’ambiente IT (IT environment)
- Mascherare l’intrinseca eterogeneità di tale ambiente

(Per una completa panoramica sulle spinte propulsive che le esigenze di business hanno esercitato ed esercitano tuttora sulla definizione degli standard SOA si veda [BEA1])

Da un'architettura che segue tali linee guida scaturirebbero numerosi vantaggi chiave: una **ridotta complessità degli “inter-sistemi” sviluppati** unitamente a **un incremento delle possibilità di un loro riuso in contesti differenti**, operando **un minimo riadattamento degli stessi**.

Si raggiungerebbe inoltre la tanto agognata **“legacy integration”**, espressione indicante l'integrazione di applicazioni “ereditate” da precedenti e differenti tecnologie (o soggetti) di sviluppo.

L'architettura che tenta di rispondere alla richiesta di queste condizioni è stata già da qualche tempo individuata e formalizzata: si tratta dell'**architettura orientata ai servizi**.

Nel formalizzarne definizione e linee guida un errore, nel quale fortunatamente non si è incorso, sarebbe stato quello di limitarsi a dotare i vari sistemi di una nuova tecnologia standard per comunicare fra loro. Ciò avrebbe significato essenzialmente fornir loro un'altra “lingua” che sarebbe entrata a far parte delle tante che già affollano il panorama informatico.

Quindi se si era pensato che l'introduzione del meta-linguaggio XML fosse il cardine intorno al quale ruota la rivoluzione delle SOA si è incorso in un errore di fondo. Infatti, pur riconoscendone l'importanza centrale ai fini di una comunicazione “universalmente comprensibile”, in questo lavoro si tralascia di discorrere diffusamente del ruolo di XML in questo senso proprio perché le SOA hanno ben'altra “carica” innovativa.



Decisamente più proficuo ai fini del raggiungimento degli obiettivi preposti è stato invece il voler “insegnare” ai sistemi software come **interagire in un dialogo**, fatto sì di un linguaggio comune, ma realizzato **prima di tutto in un diverso procedere del sistema quando si tratti di relazionarsi con l'esterno** (intendendo in questo caso per “esterno” qualsiasi cosa che stia al di fuori del dominio del sistema e quindi non necessariamente esterno al confine dell'azienda o altra organizzazione), un procedimento che prevede una partecipazione molto più attiva del sistema, un metodo che impone **la ricerca della collaborazione tra i sistemi a tutti i costi**.

L'apertura agli scambi attivi con l'ambiente esterno modella i sistemi in una forma diversa e li considera come **servizi**.

Una buona approssimazione nella descrizione di questa architettura si ottiene guardando al paradigma del “Distributed Computing” dal punto di vista delle organizzazioni dedite all'implementazione non di soli “calcoli” e scambi di dati, ma all'automatizzazione di logiche e processi di business complessi, coinvolgenti diversi sistemi automatici.

Dopotutto un processo di business (o più processi correlati) e il calcolo del fattoriale hanno pur sempre in comune il fatto di essere entrambi “una sequenza finita e ordinata di operazioni elementari, volta al raggiungimento di un ben determinato scopo”. Sostanzialmente quindi si parla in entrambi i casi di algoritmi.

La differenza (che si percepisce come abissale, ma che tanto abissale non è) sta nella diversa dinamicità con la quale i due diversi algoritmi hanno a che fare (praticamente nulla nel caso di un algoritmo di calcolo, esasperata invece in un contesto di “e-Business”), dinamicità poi accentuata dall'eterogeneità delle situazioni che deve sapere oggi affrontare in una compagnia un'infrastruttura informatica degna di questo appellativo.

Quindi si può pensare che una SOA sia un organismo costituito di cellule dette “**Servizi**”.

La definizione di “Servizio” varia lievemente a seconda del contesto; comunque, fra le possibili, si può scegliere (per “sinteticità” e per il particolare punto di vista adottato in questo lavoro) quella che descrive il servizio come uno **specifico, ben determinato, non banale, insieme di funzionalità** (così come enunciato in [BEA2]).

A ogni modo caratteristiche comuni dei servizi (quale che sia la definizione che ad essi si associ) sono:

- l’essere **sistemi software**
- l’essere **autocontenuti**
- l’essere dotati di una **descrizione formulata attraverso un linguaggio formalizzato e per quanto possibile standard**
- il loro essere “disegnati” per un’**interazione “Machine-to-Machine”**
- il loro dover essere **facilmente accessibili e “rintracciabili”**
- il fatto che la comunicazione che intrattengono con l’ambiente esterno si basi esclusivamente su **scambi di messaggi**

“Traslando” queste direttive verso un punto di vista più tecnico (e sottendendo la natura software e autocontenuta del servizio) i punti chiave sono quindi:

- Un servizio agisce tipicamente in un contesto **di rete**
- Un servizio è destinato e pensato per una interazione **“Machine-to-Machine”**
- Un servizio dovrebbe essere accessibile in seguito a un’operazione di “discovery” per quanto possibile **automatica e semplice** (leggasi “il più possibile **trasparente** all’utente finale”)
- Le interazioni con il servizio devono poter avvenire attraverso una descrizione dello stesso che sia **“machine-processable”**
- Un servizio dovrebbe basarsi solo su comunicazioni **“message-based”**

Nell'introduzione alle caratteristiche peculiari di un servizio si è volutamente commesso un errore: si è detto, e qui sta l'errore, che un servizio è “un sistema software”.

L'errore è più sottile di quanto non si pensi poiché il servizio è in realtà **la descrizione astratta delle funzionalità e delle modalità di accesso ad esse che dovranno poi essere implementate in un sistema software detto “Agente”**, il quale è, in un certo senso, concettualmente “sottostante” (considerando più “alta” la sfera delle funzionalità e più “bassa” quella della loro implementazione) a quello che chiamiamo servizio.

Comunque si tratta di una distinzione che nel resto della trattazione non assumerà quasi mai particolare rilevanza (e qualora questo avvenga lo si farà notare senz'altro), pur avendone parecchia nel contesto teorico nel quale ci si sta per ora muovendo.

(Un approfondimento su questo argomento è comunque disponibile consultando i documenti del W3C, in particolare [WSA])

Resta da introdurre una seconda definizione di servizio che spiega in buona parte il successo che questi ultimi stanno riscuotendo, specialmente nel campo dell'integrazione dei diversi sistemi aziendali.

In base a questa “nuova” definizione (della quale riporteremo solo i punti che differiscono dalla precedente) un Servizio è:

- Registrabile in una Directory
- Utilizzabile indistintamente sia in locale che in remoto
- Componibile con altri servizi (ciò da luogo a servizi cosiddetti “compositi”)

Definizione che ricalca perfettamente alcuni aspetti peculiari di una risorsa IT, la quale deve essere anch'essa:

- Descrivibile in un linguaggio formalizzato

- Registrabile
- Facilmente “rintracciabile”
- Utilizzabile sia in locale che in remoto
- Componibile con altre risorse

Questo parallelo (descritto in [CAWP]) è illuminante ai fini della comprensione del perché un servizio “approssimi” così bene (“best fits”) una risorsa IT.

Cosa questa che, trasferita a un contesto con molteplici servizi, ci fa capire perché un’architettura SOA così bene descriva e permetta di gestire un insieme frammentato ed estremamente eterogeneo di risorse quali sono quelle che di solito compongono l’infrastruttura IT di un’azienda (specie di dimensioni medio-grandi).

In questo senso è senz’altro giusto asserire che una SOA assicura **modularità e flessibilità**.

Inoltre un altro vantaggio ottenibile con questa architettura è quello di rendere il linguaggio del Business molto più “business-worker oriented” (da prettamente “developer oriented” che era), a patto comunque che si adottino adeguate tecnologie di “service-orchestration” e “service-composition”.

Un ulteriore “principio” tecnico dei servizi, finora taciuto ma non certo di minore rilevanza rispetto agli altri è che un’architettura orientata ai servizi degna di tale nome **deve nascondere, nella descrizione e nell’utilizzo, la complessità di una comunicazione remota**.

Principio questo sul quale si basa il presente lavoro.

\*\*\*

Nonostante un approfondimento della teoria delle SOAs esuli dagli scopi di questa introduzione (dopotutto nel seguito si avrà modo di approfondire questi argomenti e di introdurne anche di nuovi), si vorrebbero aggiungere alcune ulteriori nozioni che possono sgombrare il campo da eventuali perplessità e dubbi.

Innanzitutto un servizio non è da considerarsi solo come un agente attivo, ma deve anche essere presa in considerazione la possibilità che esso **diventi oggetto di gestione da parte di altri servizi**, assunzione questa da tenere a mente quando si progettino le interazioni del servizio con il mondo esterno.

Secondariamente un altro concetto (tralasciato finora dall'essere esplicitato ma leggibile tra le righe della precedente esposizione) spesso citato come caratterizzante dei servizi è il loro essere **“Course-Grained”**, cioè il loro incorporare un insieme di funzionalità non troppo ridotto e non banale (cosa questa che, a ben guardare, rispecchia ancora meglio le unità di business che un servizio è chiamato a incapsulare e a “impersonare”).

Si rende poi necessario ribattere alla possibile obiezione secondo la quale la componibilità dei servizi entrerebbe in contrasto con il principio della “granularità-grossa” appena citato.

Si risponderà che i due principi non sono affatto in contrasto tra loro, poiché **componibilità** e **granularità** sono aspetti diversi in base ai quali progettare e realizzare una SOA. Il primo ha a che vedere con le relazioni che un servizio intrattiene con gli altri, il secondo riguarda l'insieme delle funzionalità del servizio.

Infine non si può non citare il principio per il quale i servizi dovrebbero essere **“loosely-coupled”**, vale a dire quel principio per il quale **le dipendenze tra i servizi dovrebbero essere minime**. Questo è una conseguenza del fatto che un servizio dovrebbe essere per quanto possibile course-grained, incapsulando una funzionalità di business il più possibile “autonoma e completa”.

Quanto appena espresso sull'accoppiamento dei servizi costituisce inoltre un vincolo importante per la già citata componibilità dei servizi.

\*\*\*

Dopo la doverosa introduzione a carattere prevalentemente teorico è ora possibile focalizzare l'attenzione sugli aspetti più tecnici del paradigma e l'impatto che questa tecnologia ha con il cosiddetto "real-world". Questo aiuterà a chiarire eventuali dubbi residui che la precedente trattazione non avesse eventualmente fugato.

Innanzitutto quando si parla di "Service-Orientation" non si entra affatto in rotta di collisione con la "Object-Orientation"(OO); ma questo era dopotutto evidente, poiché ci si trova di fronte a un'evoluzione della seconda, non a una rottura nei suoi confronti. Evoluzione che, per così dire, "estremizza" le idee fondanti dell'OO, avvicinandoli alla realtà (che ci si propone di modellizzare) ancora di più di quanto già l'OO non facesse.

"Autocontenimento", "Interfaccia" e gli altri concetti poc'anzi esposti sono un'eredità diretta del paradigma ad oggetti e sicuramente non delle novità.

Si pensi poi a quanto vantaggiosa in termini di **riutilizzo del codice** sia un'architettura Service-Oriented; e anche in questo caso l'OO sorride certo di compiacenza, al vedere una delle sue lezioni più importanti così ben applicata.

Da queste parole e dalle considerazioni precedenti emerge un quadro rassicurante e promettente ma, come si suol dire, "non sono tutte rose e fiori". Le difficoltà di adozione di un simile nuovo paradigma di sviluppo esistono e sono tali che, ancora oggi, a distanza di quasi cinque anni dalla loro introduzione, una certa diffidenza e quello che è quasi un "rifiuto inconscio" delle SOA ne stanno ritardando non poco la diffusione nel mercato.

Le ragioni di un simile ritardo sono riconducibili in buona parte alla solita diffidenza degli addetti del settore, che vedono innovazioni di carattere concettuale quasi come una minaccia alla loro già consolidata esperienza in un particolare campo.

Abbracciare un simile cambiamento è senz'altro impegnativo, tale da far affermare a qualcuno che:

*“[... ] it is a **cultural change**. You can't go into it pretending it's not going to be a challenge.”<sup>vii</sup>*

(Jaime Sguerra, Chief Architect alla “Guardian Life Insurance”; si veda [SMRW]).

Affermare il contrario sarebbe un punto di vista criticabilissimo, ma rimanere arroccati sulle proprie posizioni per la solita “paura del nuovo” è un grosso errore. Errore che può costare molto in termini di guadagno di efficienza e di risparmio sui costi d'integrazione, presenti e futuri.

In un senso più pratico le problematiche di adozione di una SOA possono essere ricondotte a quattro temi fondamentali:

- Anzitutto la **sicurezza**; sebbene molto lavoro sia stato svolto sul versante della standardizzazione ad esempio di certificati e altri meccanismi, si è ben lungi dall'approdare al bandolo della matassa. Questo significa che, in taluni campi (e.g. **e-Banking**), le SOA non “attecchiranno” prima di qualche tempo.
- Secondariamente, in campi basilari quali l'**orchestrazione dei Web Services** (vale a dire tutta quella serie di pratiche e di tecnologie finalizzate a dirigere e coordinare le interazioni fra diversi servizi in maniera coerente ed omogenea) gli standard **certi** con i quali iniziare a lavorare tardano ad arrivare.
- Non si può poi certo dire che i Web Services siano particolarmente veloci e performanti (discorso questo che però varia, e non poco, da implementazione a implementazione).
- Altro problema, annoverabile fra quelli più pressanti è poi **la penuria di tools di gestione** che siano completi e semplici da utilizzare.

- Infine si pensi alla “discontinuità” esistente tra **le operazioni di gestione di una SOA** e quelle **di management del registro che tiene traccia dei servizi** e della loro ubicazione, funzionalità non sempre realizzate con completezza e quasi mai rese disponibili all’insegna della semplicità.

Delle quattro categorie di problemi, la seconda e le ultime due sono le più pressanti; ma, mentre nel caso della cosiddetta “Web Services Orchestration” iniziano a sorgere delle soluzioni affidabili e destinate alla standardizzazione (BPEL ad esempio, a riguardo del quale si può consultare [BPE]), nel caso della gestione di una SOA e del registro a lei associato (punto questo non del tutto indipendente dalla problematica dell’orchestrazione) il mercato tarda ad offrire quella che di solito si designa con il termine di “kill-app”.

Al suo posto vengono spesso offerti prodotti rivolti alla gestione “a basso livello” dei messaggi scambiati tra i servizi (tematica questa affrontata nel Capitolo 7), con evidente perdita del (alto) livello di astrazione che dovrebbe consentire (e caratterizzare) una SOA.

In questo lavoro non si intende certo sopperire noi a questa mancanza, ma preme far notare come in alcune aree chiave, quali il **discovery**, il **publishing** e il **load-balancing** dei servizi, l’offerta di tools di gestione sia relegata ad ambiti enterprise (una soluzione completa e dedicata completamente alla gestione dei servizi quale ad esempio “Connected Services Framework”, della Microsoft, non è di sicuro “abbordabile” in ambiti medio-piccoli, così come non lo sono le altre presentate nel Capitolo 7) o destinata a essere composta da soluzioni di non semplice implementazione e gestione.

Quel che manca è un sistema software “rassicurante” e semplice da gestire.



Questo sistema deve aiutare un'azienda nel soddisfacimento di essenzialmente due esigenze:

- Prima di tutto deve permettere una **semplice e veloce gestione di una SOA in ambito Intranet**, senza penalizzare talune funzionalità avanzate, quali gestione del carico e sicurezza, che non possono non venire considerate fondanti di un'infrastruttura IT.
- Secondariamente deve trattarsi di una soluzione **facilmente portabile in un contesto Extranet (B2B e scenari di out-sourcing, quindi)**, nel passaggio al quale gli interventi richiesti sul sistema dovrebbero essere minimi.

Prioritaria importanza viene in questo modo data al rendere semplice l'introduzione del paradigma Service Oriented nella gestione interna di un'infrastruttura IT aziendale, poiché è altrimenti impensabile proporre l'utilizzo per gestire i rapporti della compagnia con l'esterno.

Nell'analisi preliminare del progetto ADES la ricerca di una soluzione che coprisse questi e altri obiettivi è partita da quello che già offre la tecnologia attuale.

I Web Services, attualmente la tecnologia preferenziale di realizzazione di una SOA, sono composti di quattro standards:

- **WSDL** (Web Services Description Language) per la definizione dell'interfaccia pubblica del servizio.
- **SOAP** (Simplified Object Access Protocol) per lo scambio di messaggi.
- **HTTP** come principale protocollo di comunicazione (pur non essendo questa l'unica possibilità).
- **UDDI** (Universal Discovery Description and Integration) per il discovery e il publishing dei servizi.

Se da una parte questa architettura è uno standard affidabile, dall'altra soffre di principalmente due problemi:

- Il primo riguarda SOAP e HTTP, del cui uso combinato si può dire che non sia ancora particolarmente performante.
- Il secondo e più grosso problema è lo standard UDDI e la difficoltà di implementazione prima e di gestione poi che presenta un tale meccanismo di publishing dei servizi.

Gran parte delle implementazioni UDDI sono **“pesanti”, centralizzate e vanno oltre i bisogni di chi vorrebbe invece un “provider” di servizi leggero e di facile gestione.** Un software che **partecipi più attivamente agli scambi tra i servizi, pur nascondendo le difficoltà legate alla loro gestione e al loro reperimento.**

Focalizzando solamente sul secondo punto, e tralasciando leggerezza e una gestione non certo immediata di UDDI, si può passare a considerare come molte implementazioni di questo standard soffrano del fatto di essere pesantemente centralizzate, con tutti i problemi che ne conseguono (hardware dedicato, possibili problemi nel caso il sistema dovesse essere spento o subisse guasti o rallentamenti, il bisogno di approntare un sistema ausiliario pronto all'intervento per far fronte a questa eventualità, etc.).

Altro problema è la presenza sul mercato di soluzioni “plasmate” su particolari esigenze di un sistema e non facilmente portabili in altri contesti.

Inoltre, come si diceva, lo standard UDDI è pesante (conseguenza questa del fatto che esso deve essere completo e capace di gestire una vasta gamma di casistiche di utilizzo).

Questi problemi sono quelli che si cerca di aggirare con questo lavoro.

### Note

---

- <sup>i</sup> Per sistema IT si intende il complesso (o parte del) sub-strato tecnologico al quale l'azienda affida l'automazione di un numero variabile delle sue attività. Prevalentemente un sistema IT è controllato da uno o più calcolatori digitali. Il concetto di sistema IT è simile a quello di "sistema informativo", ma, a differenza di quest'ultimo, viene maggiormente posto l'accento su una spiccata automazione e sulla "Enterpriseness" del sistema IT.
- <sup>ii</sup> Per "Security policing" si intende l'insieme di procedimenti, metodologie e architetture software che in un sistema presiedono all'attività di gestione di complessi scenari di sicurezza che comprendono, tra gli altri, meccanismi di gestione degli utenti, management di gruppi di utenti, organizzazione delle credenziali di accesso a diverse parti del sistema e attività di crittografia dei dati.
- <sup>iii</sup> Per "offshore outsourcing" si intende una pratica di "outsourcing" caratterizzata dall'essere estremamente disconnessa dall'azienda, soprattutto dal punto di vista geografico. A sua volta, per "outsourcing" si intende il particolare processo di delega di attività (o risorse), necessarie all'azienda, ad agenti economici esterni alla stessa (tipicamente altre aziende).
- <sup>iv</sup> Si designa con il termine "legacy" quella particolare categoria di sistemi informativi che sono "eredità" (appunto traducibile in Inglese con il termine "legacy") di precedenti attività di sviluppo, svolte con diverse metodologie e/o da diversi team di sviluppo (i quali non hanno adeguatamente calcolato la necessità di integrazione con altri sistemi di quello sviluppato). Si tratta di sistemi la cui sostituzione risulta particolarmente problematica (per ragioni di costo o di "familiarità" che il personale ha con essi acquisito), diventando in questo modo un'eredità "scomoda" con la quale fare i conti.
- <sup>v</sup> Con il termine "middleware" si intende un particolare strato software che svolge attività di mediazione (spesso "linguistica") e coordinamento tra gli strati superiori della struttura IT.
- <sup>vi</sup> I simboli tra parentesi quadre ("[" e "]") sono riferimenti alla Bibliografia, consultabile nelle pagine finali di questo lavoro.
- <sup>vii</sup> "Si tratta di un cambiamento culturale. Non lo si può intraprendere pretendendo di non dover affrontare una difficile sfida."

## **2. L'attività di brokering in una SOA**

ADES prende le mosse da alcuni principi del mondo dei Web Services (da ora in poi WSs) che riguardano le problematiche di reperimento degli stessi attraverso la rete, problematiche che in una certa qual misura cerca di risolvere lo standard UDDI (del quale si è già parlato nel Capitolo 1).

Ebbene tali problematiche sono quelle che investono le attività dette “di brokering” di un'architettura orientata ai servizi.

Nel seguito esse verranno esaminate in dettaglio, a partire da un'introduzione che si serve dello standard UDDI quale esempio.

### **2.1. Scopi e funzionalità di un sistema di brokering, in particolare relazione allo standard UDDI**

Riprendendo l'esempio di una rete di servizi web si pensi al singolo Web Service (da ora in poi WS) come identificato univocamente fra gli altri da una coppia “URI/<Nome del servizio>”, alla quale un agente esterno, che del WS voglia fare uso, fa riferimento per indirizzare le proprie richieste.

Lo spostamento di un WS comporta, nel migliore dei casi, la modifica di un file di configurazione dell'applicazione contenente l'“indirizzo” del WS e, nel peggiore, la ricompilazione della stessa.

Queste sono entrambe attività che non possono essere automatizzate agevolmente o la cui automatizzazione su una distribuzione magari ampia di applicazioni comporta grosse (quando non insormontabili) difficoltà.

Inoltre si vorrebbe trovare una via anche concettualmente più “elegante” di risolvere questi problemi.

Si è trovato che la soluzione migliore è quella di far puntare le richieste dell'applicazione alla voce che nel registro UDDI (si è parlato di UDDI nel capitolo precedente) identifica il WS, demandando completamente l'attività di aggiornamento degli indirizzi a UDDI stesso e rendendolo di fatto un "elenco telefonico auto-aggiornante" al quale si rivolge chi volesse usufruire di un servizio, comunicando un nome (quello del servizio) e ottenendo in cambio di poter usufruire del servizio in maniera completamente trasparente all'indirizzo dello stesso e di sue (dell'indirizzo) eventuali variazioni.

Questo fa del registro UDDI un agente software responsabile del reperimento e del conseguente "provisioning" (l'atto del rendere disponibile una particolare risorsa) del servizio richiesto a chi ne faccia richiesta, esattamente gli scopi principali ai quali deve assolvere un agente software particolare, detto "**Broker**" e che nel particolare contesto di un'architettura orientata ai servizi viene detto "**Broker di servizi**" (Service broker).

Una più rigorosa enunciazione delle funzionalità alle quali un sistema di brokering deve assolvere passa obbligatoriamente per i seguenti punti, i quali descrivono le funzionalità essenziali di un broker:

- **Selection:** vale a dire la selezione tra un insieme di servizi di quello richiesto dall'utente (compito a prima vista banale, ma non privo di interessanti problematiche).
- **Provisioning:** rendere disponibile il servizio al chiamante, una volta che ne sia stata effettuata la selezione.
- **Interaction:** la regolamentazione, nonché la mediazione, del dialogo tra i servizi.
- **Chaining:** la corretta orchestrazione e dell'ordine e dei ruoli che ciascun servizio ricopre all'interno di un'ideale catena di servizi (punto strettamente correlato al precedente).

Un particolare su cui porre l'accento è poi il fatto che un'azione che assolva ai compiti sopraelencati non agisce sul servizio in se, ma regola **gli scambi che tra i servizi avvengono**.

Questo significa che un broker tipicamente agisce, non solo fisicamente ma anche a un livello concettuale, sui **messaggi** che i servizi scambiano.

È poi necessario chiarire quale sia il campo di applicazione di un sistema di brokering; questo perché, in un periodo di transizione come quello nel quale ancora ci si trova, target sono, non solo le nuove piattaforme SOA e i sistemi legacy, ma anche e soprattutto un misto tra i due.

Quest'ultima nota chiarisce quanto oggi una soluzione informatica che vuole sperare di essere vincente non possa prescindere dal cercare versatilità e duttilità, senza compromessi che non siano imputabili a ragioni di costo (si noti come la categoria "tempi di sviluppo" non ricada nelle casistiche di applicazione del compromesso).

La domanda che ci si potrebbe senz'altro porre è quindi come e perché non lasciar fare a UDDI il lavoro di broker in una SOA.

Per rispondere bisogna chiarire alcune cose: UDDI non è di per sé un sistema software ma bensì uno standard in base al quale realizzare sistemi software. Questo ci fa capire come molte delle capacità di un broker realizzato sullo standard UDDI dipendono dalle scelte progettuali e architetturali di chi realizza un sistema simile.

Inoltre non ci si può fermare ai già citati problemi di molte delle attuali implementazioni UDDI (che ricordiamo essere: pesantezza dello standard e centralizzazione del ruolo del broker).

Bisogna pensare anche al fatto che un approccio al problema del brokering basato esclusivamente su UDDI non prevede di solito alcune funzionalità molto utili, quali potrebbero essere la gestione di un meccanismo di **"load-balancing"** delle richieste ai servizi, qualora in rete ne fossero presenti più copie o versioni differenti, in grado di gestire carichi diversi.

(A riguardo delle possibilità offerte da UDDI per il “versioning” dei servizi è utile consultare [CSZN])

Con UDDI tali dinamiche non rivestono un ruolo centrale e risultano di difficile implementazione.

Inoltre i problemi di sicurezza e di autenticazione vengono demandati ai servizi e non esiste la possibilità di stabilire per loro una politica veramente omogenea. La soluzione che più si avvicina a una tale gestione unitaria contempla la creazione di diversi registri che operino seguendo diverse “policies” di sicurezza, cosa questa ad evidente discapito dell'omogeneità richiesta e che risponde all'esigenza di unitarietà con il proliferare dei diversi registri (ingenerando un evidente paradosso).

Sia chiaro che tali soluzioni potrebbero essere inserite in un'implementazione che su UDDI si basi, ma il vero problema è che lo standard non prevede molte di queste funzionalità e ciò spesso significa diverse soluzioni sviluppate da diverse compagnie che abbiano implementato ognuna per conto proprio UDDI.

Inoltre (e si tratta di serie limitazioni per uno standard che si propone come “universal”) , ciò che rende UDDI poco attraente agli occhi dell'utente digiuno di tecnicismi sono la sua “enterprise-business orientation” e il fatto che quasi tutte le sue implementazioni offrano una vista troppo “tecnica” delle funzionalità esposte (una definizione più dettagliata dei concetti di “business user view” e di “technical view” sono presenti in [CSZN], pag. 4).

ADES risponde a tutto questo con due concetti: **semplicità e un alto livello di astrazione “user-oriented”**.

## **2.2. I concetti di “broker centralizzato”, “broker distribuito” e quello di “brokering di messaggi”**

Nel particolare contesto delle varie implementazioni dello standard UDDI una caratteristica che assume particolare importanza è la “centralizzazione” del broker in un'unica macchina; caratteristica riscontrabile sia nella maggior parte delle implementazioni UDDI, sia in soluzioni che propongono funzionalità di brokering non riconducibili allo standard UDDI.

Quest'ultimo è il caso delle soluzioni presentate nel Capitolo 7 e di altre non analizzate in questo lavoro, quale Microsoft SQL Server 2005<sup>®</sup>, che presenta una funzionalità di service brokering basata sul routing asincrono dei messaggi scambiati tra i servizi e su estensioni al linguaggio “Transact-SQL” (T-SQL).

Nell'ultimo caso, nonostante esista la possibilità di far interagire diverse installazioni di SQL Server su macchine diverse per gestire complessi flussi di messaggi, nonostante questo non si può parlare di una vera e propria gestione distribuita delle funzionalità di brokering del prodotto.

Gran parte degli sforzi fatti nella realizzazione di questo lavoro sono stati diretti in direzione della distribuzione del carico di lavoro delle attività di brokering, sia che si parlasse di una distribuzione “fisica” su più macchine, sia che si prendesse in considerazione una distribuzione “logica” delle attività.

In buona sostanza si è deciso di non realizzare un'unica monolitica applicazione che risiede su un'unica macchina e che si occupi della localizzazione e del provisioning dei servizi a chi ne faccia richiesta.

Si è invece deciso di seguire l'esempio dei meccanismi di routing che avvengono al livello di rete, in particolare quello di Internet (si veda a questo proposito [TANB]).



Per riassumere brevemente (e senza scendere troppo nel dettaglio), il routing in una rete composta di diversi nodi collegati tra loro da più linee è riconducibile ad algoritmi particolari (detti algoritmi di routing) eseguiti su più macchine dette routers, che provvedono allo smistamento del traffico su questa o quella linea.

Lo smistamento avviene in base a considerazioni che principalmente tengono conto del carico di una linea piuttosto che di un'altra e che cercano il cammino più breve tra i due nodi che desiderino comunicare.

Come sia stato applicato l'esempio del routing di rete al sistema ADES verrà discusso approfonditamente nel Capitolo 4; per ora preme evidenziare solo come una simile strategia di scelta del percorso migliore nel caso di una SOA si riduca a trovare l'indirizzo di un servizio e a fornirlo al chiamante del servizio, il tutto in maniera trasparente al chiamante e al servizio stessi.

E preme ancora di più far notare come una simile attività possa essere svolta, come dicevamo, anche su un calcolatore centrale, dando luogo a un brokering di servizi **“centralizzato”**.

Non è questo il caso di questo lavoro. È stato infatti scelto di realizzare un'applicazione leggera e destinata all'installazione su più macchine, che consenta loro di “collaborare” ai fini del compimento delle task di “selection” e di “provisioning” dei servizi.

È questa la differenza principale che sussiste tra il sistema ADES e la maggior parte dei tools in commercio. Ed è questa differenza che rende interessante la progettazione e la realizzazione del sistema distribuito oggetto di questo lavoro, **un broker di servizi “distribuito” piuttosto che un broker di servizi “centralizzato”**.

\*\*\*

Un ultimo concetto che si deve analizzare nel contesto delle operazioni di brokering è riconducibile a quelli poc'anzi esposti poiché, quando si parla di **“brokering di messaggi”**, si parla in realtà di un brokering “spurio” (intendendo il termine “spurio” in un’accezione meno forte di quella tradizionale) dei servizi, un’attività che si riduce a un routing di livello applicativo dei messaggi scambiati tra i servizi.

È questa una task di “basso livello” che richiede spesso (come si vedrà molto bene nel Capitolo 7) una conoscenza profonda dei diversi formati dei messaggi scambiati e che, in generale, vanifica l’alto livello di astrazione fornito dalla possibilità di poter pensare in maniera “orientata ai servizi”, astrazione fornita dalla trasparenza proprio di quei meccanismi che il brokering di messaggi non nasconde, anzi, rende ben evidenti e centrali ai fini dell’elaborazione.

Task che si è quindi deciso di non porre a diretta conoscenza dell’utente del software realizzato, cercando in questo modo di schermarlo da dettagli eccessivamente “tecnici” e tentando di raggiungere anche un bacino d’utenza per così dire “manageriale”.

### **3. Ambiente software e requisiti di esecuzione del sistema ADES**

Il sistema ADES consta essenzialmente di:

- una libreria per lo sviluppo di servizi che vogliano sfruttare le funzionalità di broker distribuito da ADES offerte
- un broker distribuito, incluso in tale libreria
- alcuni tools di amministrazione per la gestione da remoto del software e altri per la gestione locale del sistema e dei servizi “ADES-Powered”

I requisiti di esecuzione di questa piattaforma software sono comunque esigui e di facile installazione; alcuni poi sono già presenti all’atto dell’installazione dei sistemi operativi descritti nel paragrafo 3.2, “Sistemi operativi e runtime .NET supportati”.

#### **3.1. Requisiti hardware**

Dal punto di vista hardware un PC di uso domestico può facilmente ospitare un’installazione della soluzione ADES.

I dettagli seguenti sono quindi forniti a scopo di pura completezza, in quanto un qualsiasi computer in commercio da quattro anni a questa parte supera abbondantemente tali configurazioni hardware:

- CPU: 90 MHz, Pentium<sup>®</sup>, equivalente o superiore (si parla quindi di un qualsiasi processore Intel<sup>®</sup> o AMD<sup>®</sup>)
- RAM: 96 MB minimi
- 4-10 MB di spazio su disco, a seconda del tipo di installazione (del solo software ADES, senza considerare quelli necessari per il .NET framework)

- La possibilità di accesso a una rete data da:
- Un qualsiasi modem 56k
- Una scheda di rete Ethernet 802.3 compatibile
- (altri dispositivi per la comunicazione remota quali ad esempio un modem GPRS, un cavo seriale o un modem ADSL)
- Un lettore CD-ROM 24X o superiore

I requisiti qui indicati tengono conto della sola installazione della piattaforma ADES, senza tenere in considerazione eventuali servizi sviluppati in base a essa ed eventualmente distribuiti a corredo della stessa.

### **3.2. Sistemi operativi e runtime .NET supportati**

L'ambiente ADES poggia sul framework di sviluppo Microsoft .NET<sup>®</sup>.

Questo, dal punto di vista dei sistemi operativi supportati, taglia fuori dal mercato di deployment del software l'intero pianeta UNIX<sup>®</sup>/Linux, quello MacOS<sup>®</sup> e altri quali le piattaforme Sun<sup>®</sup> Solaris<sup>®</sup>.

Al momento in cui si scrive questo è sostanzialmente un assunto vero, ma lo scenario futuro potrebbe variare completamente e, soprattutto, rapidamente.

È infatti comparsa sulla scena da poco più di tre anni una variante Open Source del .NET Framework; si tratta del progetto Mono ([www.mono-project.com](http://www.mono-project.com)), di recente patrocinato dalla Novell<sup>®</sup> ([www.novell.com](http://www.novell.com)) il quale può contare su una già nutrita schiera di sviluppatori che hanno contribuito a maturare non poco il progetto (del quale sono state rilasciate a partire da Giugno 2004 le prime versioni non “beta” e sufficientemente stabili).

Obiettivi dichiarati del progetto sono quelli di fornire alla comunità Open Source un framework che ricalchi alla perfezione la libreria di classi .NET e che fornisca gli strumenti per poter sviluppare tramite i linguaggi C#, VisualBasic .NET e persino JAVA<sup>®</sup>.

Ciò che più interessa (perlomeno ai fini di questo lavoro) è però la portabilità che esso garantisce ai software sviluppati in uno dei linguaggi supportati: oltre alle più diffuse piattaforme Linux, si parla di codice compatibile con l'ambiente MacOS<sup>®</sup>, sistemi Sun<sup>®</sup> Solaris<sup>®</sup> e altre configurazioni, passando per piattaforme ancora più esotiche (quale QNX<sup>®</sup>).

Attualmente, pur essendo il porting del progetto ADES per la piattaforma Mono un obiettivo di una certa importanza, è necessario eseguire ADES sfruttando il framework Microsoft<sup>®</sup> e non si garantisce il funzionamento del software al di fuori dei sistemi operativi seguenti:

- Windows 98
- Windows 98 SE
- Windows Millennium Edition (Me)
- Windows NT 4 (richiesto il ServicePack 6a)
- Windows 2000 (qualsiasi versione è supportata ma è consigliato il ServicePack 2)
- Windows XP Home Edition
- Windows XP Professional Edition
- Windows Server 2003 (qualsiasi versione)

Si noti come resti escluso dalla lista Windows 95, che non è supportato dal .NET Framework.

È poi necessario che il sistema destinazione di un'installazione del .NET framework sia equipaggiato di almeno la versione 5.1 di Internet Explorer<sup>®</sup> e di quella 2.0 del Windows Installer<sup>®</sup> (anche noto come MSI 2.0, "Microsoft Installer").

La lista degli OS supportati va soggetta a ulteriore rimaneggiamento qualora si volesse usufruire di ADES come framework di sviluppo Service oriented.

In questo caso l'installazione del .NET SDK (oltre al .NET Framework) è praticamente obbligatoria e restringe l'elenco degli OS supportati alla seguente lista:

- NT 4 (SP 6a)
- 2000 (qualsiasi versione, ma è consigliato il SP2)
- XP Professional
- Server 2003 (qualsiasi versione)

La versione del framework .NET supportata e utilizzata per lo sviluppo del software è la 1.1; in particolare si è fatto ricorso al Service Pack 1 della stessa per risolvere alcuni bugs nelle librerie "System.Windows.Forms".

La versione 1.0 del runtime .NET potrebbe presentare dei problemi di compatibilità con il sistema ADES, pertanto se ne sconsiglia fortemente l'utilizzo.

Un'installazione del genere (consistente cioè del solo .NET framework 1.1) non copre comunque alcune funzionalità di ADES che, benché opzionali, sono decisamente utili.

Tra di esse figura un'interfaccia di amministrazione remota, implementata ricorrendo alla tecnologia lato server ASP.NET. Ciò implica che si abbia a disposizione un server ASP.NET per la pubblicazione di questa interfaccia di amministrazione.

Si può ricorrere al server IIS<sup>®</sup> ("Internet Information Services", versioni 5.x o superiori), disponibile come installazione opzionale dei sistemi Windows a partire dal 2000 in poi (e disponibile sul supporto di installazione del sistema operativo).

In alternativa è possibile ricorrere a prodotti Open Source o comunque freeware; per lo sviluppo di ADES si è dimostrato più che adatto allo scopo il web server "Cassini", un tempo parte del progetto "Web Matrix" ([www.asp.net](http://www.asp.net)), caratterizzato dall'essere semplice, leggero e Open Source.

Ulteriori requisiti opzionali sono da tenere in considerazione qualora si utilizzi ADES per la realizzazione di servizi che facciano uso delle classi di accesso ai dati del .NET framework: in questo caso è richiesta l'installazione delle librerie MDAC 2.7 (Microsoft Data Access Controls).

Infine se si prevede l'accesso alle informazioni rese disponibili dal provider WMI (Windows Management Instrumentation) è ovviamente necessario che questo sia installato sul computer in uso.

\*\*\*

Quale utile riassunto di quanto appena detto si presentano le specifiche hardware e software del PC utilizzato per lo sviluppo del sistema:

- Architettura x86
- Processore AMD® Athlon XP® 1800+ (~1533 MHz)
- 256 MB RAM DIMM
- Modem 56K
- Scheda Fast Ethernet 802.3
- Disco rigido 20 GB, dei quali:
  - ~30 MB utilizzati per lo sviluppo del sistema vero e proprio
  - ~87 MB occupati dall'installazione del .NET framework (1.1)
  - ~241 MB occupati dall'installazione del .NET SDK (1.1)
  - ~113 MB utilizzati per l'installazione dell'ambiente di sviluppo Microsoft® Visual Studio .NET 2003®
  - ~162 MB utilizzati per il software Microsoft® Office Visio 2003®(per un totale di ~633 MB, escludendo però alcune librerie)
- Sistema operativo Microsoft® Windows XP Professional® (5.1.2600 Service Pack 1)
- File system NTFS
- Internet Explorer 6.0 SP1
- MSI 3.1

- MDAC 2.7
- WMI (5.1.2600)
- Microsoft® .NET Framework 1.1 SP1
- Microsoft® .NET SDK 1.1
- Microsoft® Visual Studio .NET 2003®
- Microsoft® Cassini Web Server (parte del progetto “Web Matrix”)
- Mozilla Firefox 1.5 (utilizzato per il testing dell’interfaccia di amministrazione remota “Limbo”)

A corredo di questa presentazione dei requisiti software si pensa che sia utile consigliare per lo sviluppo di un’architettura orientata ai servizi e basata sulla tecnologia ADES l’adozione di un sistema della famiglia “server” di Windows, quale Windows XP Professional SP1 (si sconsiglia l’utilizzo del Service Pack 2), Windows 2000 Professional (aggiornato all’ultimo Service Pack) o, meglio ancora, Windows Server 2003.

### **3.3. Requisiti di rete**

Un ultimo appunto va infine dedicato ad alcuni requisiti del software riguardanti l’eventualità di un suo utilizzo in una rete dotata di un firewall.

In alcune modalità di funzionamento di ADES sussiste la possibilità di utilizzare porte non standard, diverse cioè da quelle alle quali stanno di solito in ascolto i cosiddetti “well-known services” di livello applicativo.

In questo caso esiste la possibilità che i pacchetti in transito su tali porte vengano bloccati da politiche (“policies”) di sicurezza restrittive del firewall.



È il caso di una configurazione dell'ambiente ADES che utilizzi il protocollo TCP su una porta “non standard” (esemplificativamente la 8090); ciò che accadrebbe se un firewall esercitasse un “blocco” di tutti i pacchetti transitanti su porte che non siano la 80 (protocollo HTTP), la 25 (protocollo SMTP) e poche altre (e comunque non la 8090), sarebbe l'ovvia impossibilità a funzionare del sistema ADES.

Per ovviare a simili problemi ADES prevede l'utilizzo del protocollo HTTP e della porta 80 (anche se i protocolli non sono strettamente vincolati all'uso di una determinata porta è di solito consigliato usufruire di coppie “protocollo-porta” ben note e convenzionali).

Altro problema potrebbe essere un firewall che effettua controlli anche sul contenuto dei pacchetti, non solo sulle porte che essi utilizzano.

In questo caso un tipo di contenuto non testuale veicolato tramite il protocollo HTTP verrebbe anch'esso irrimediabilmente bloccato.

ADES può però porre rimedio anche a questo, in quanto permette l'utilizzo della formattazione SOAP (un dialetto XML e quindi di tipo testuale) sul protocollo HTTP.

Di come questo sia possibile, quali protocolli di rete ADES utilizzi e in che modo si discuterà comunque più approfonditamente nel prossimo capitolo, dedicato all'analisi della struttura e delle modalità di funzionamento del sistema.

## **4. Analisi, progettazione e sviluppo del sistema ADES**

Questo capitolo copre la descrizione dettagliata del software ADES, a partire dall'analisi del contesto service-oriented in cui esso si inserisce fino alle problematiche più interessanti presentatesi durante il suo sviluppo.

Delle architetture orientate ai servizi si è già profusamente discusso nel capitolo 1 (pag. 8), ma si aggiungeranno in relazione a questo progetto alcuni dettagli che si sono tralasciati finora.

Innanzitutto la business-orientation dello sviluppo orientato ai servizi si ripercuote anche sulla facilità d'uso di un sistema sviluppato con l'approccio service-oriented (o perlomeno sarebbe desiderabile che così fosse).

Formati dei messaggi scambiati, conversioni tra questi formati e passaggi del processo di instradamento (routing) dei messaggi sono tutti dettagli dei quali non si dovrebbe mettere a parte l'utente finale del sistema, almeno quando si tratta di un contesto non prettamente enterprise e non si hanno fondi, tempo e risorse da dedicare all'aggiornamento del proprio dipartimento IT su un nuovo sistema, per quanto valido esso possa essere.

Discorso ancor più calzante quando si parla di sistemi che “impongono” un cambiamento di paradigma come il passaggio a un'architettura service-oriented in effetti richiede.

ADES è stato pensato per essere facile e intuitivo da usare, così come si ritengono semplici a sufficienza l'architettura e l'astrazione che offre all'utente.

Un dettaglio piuttosto importante è quello relativo alla complessità dell'ambiente di rete nel quale il broker oggetto di questo lavoro si muove.

Si è già detto che l'architettura orientata ai servizi dovrebbe nascondere una tale complessità e ADES si sforza in tal senso di “annullare” persino la percezione che un utente potrebbe avere dello stare operando in un contesto distribuito.

Inoltre si sono prese, per quanto riguarda l'interfaccia esposta dal framework al programmatore, alcune decisioni che hanno allungato i tempi di sviluppo, ma che hanno permesso di offrire un accesso alle funzionalità del framework che fosse il più semplice e intuitivo possibile.

Questo significa che non solo si è cercato di seguire come principale linea guida la semplicità di utilizzo dell'**ambiente** ADES, ma anche la facilità di fruizione del **framework** ADES (della dualità fisico-concettuale del framework/environment ADES si discuterà meglio nei paragrafi successivi).

### 4.1. Introduzione alla metodologia di sviluppo del sistema ADES

Il lavorare su un sistema di una certa complessità quale è ADES ha comportato una certa spesa (temporale) per approntare un sistema software che si prendesse carico di automatizzare quanto più possibile i vari processi di sviluppo.

A costo di una iniziale fase di “non-produttività” (per quanto riguarda il codice ADES) in cui si è allestito tale sistema si è ottenuto un vantaggio innegabile in termini di automatizzazione delle operazioni più ripetitive e più soggette a errori che si sono incontrate nello sviluppo del software.

Si sono messe così insieme alcune preziose nozioni ed esperienze che vengono da metodologie di sviluppo “vecchie” e “nuove”; si è infatti ricorsi a un misto di sviluppo “Use-Case driven” (soprattutto per quanto riguarda le fasi di analisi e di progettazione), a concetti ereditati dall’“Extreme Programming” (si è fatto ricorso estensivo agli “Unit tests”) e al paradigma della “Continuous Integration” che ha raccolto e unito i precedenti concetti sotto un'unica “pipeline” di costruzione del sistema (tale ventaglio di tecnologie sono sempre più spesso annoverate come principali costituenti dell'emergente “Agile Modeling”).

Tale pipeline si risolve in un approccio sia iterativo che incrementale schematizzabile in tre fasi: “build”, “testing” e “consolidating” (rispettivamente “costruzione”, “test” e “consolidamento” del codice).

Dall’insieme di attività automatizzate e integrate nel sistema di build resta esclusa l’attività di modellazione, la quale entra però a far parte non solo delle fasi iniziali dello sviluppo, ma di ogni iterazione dello stesso.

Questo significa che il ciclo di vita dello sviluppo è iterativo, incrementale e composto delle seguenti quattro fasi:

- **“Modeling”**: è la fase di modellazione del sistema (e dell’ambiente con il quale dovrà interagire) e sfocia in una serie di diagrammi UML (ma non solo) i quali potranno essere tradotti in codice. Alcuni tools (e Visio è tra questi) permettono la generazione automatica dello scheletro del codice rappresentato dal diagramma UML; questo, oltre a rappresentare un certo risparmio di tempo e di possibili errori dovuti al processo di “traduzione”, è di fondamentale importanza per il “riallineamento” del codice (due punti più avanti, alla voce “Consolidating”, si chiarirà questo concetto in maniera più esaustiva).
- **“Coding”** (o “Building”): rappresenta la fase di traduzione del modello UML in codice vero e proprio; si tratta inoltre di scrivere i tests che verranno usati nella fase successiva per verificare le funzionalità del sistema.
- **“Testing”**: è una fase delicata e sulla quale è sempre bene spendere più tempo possibile; si tratta di applicare al sistema frutto di questa iterazione i tests che sono stati scritti nella fase precedente (in questo senso si può dire che il procedimento di “testing” cominci nella e si sovrapponga alla fase di “coding”), correggendo gli eventuali errori riscontrati e verificando che il sistema risponda in modo corretto ed efficiente alle funzionalità richieste.

- **“Consolidating”**: si tratta di una parte del processo di sviluppo di solito non contemplata nei manuali di ingegneria del software (eventualmente viene indicata come semplice “verifica” preliminare alla “chiusura” di un’iterazione e “apertura” della successiva), ma che invece riveste un’importanza particolare, poiché è in questo punto dell’iterazione che si scrive la maggior parte della documentazione e che hanno luogo il “riallineamento” e la “ri-taratura” del codice nei confronti del modello. Va da se che questa fase sia anche cruciale per non perdere il contatto tra “ciò che andrebbe fatto” e “ciò che si sta facendo”; a questo proposito si sottolinea come la documentazione del codice (da svolgersi comunque trasversalmente a ogni fase dello sviluppo) rappresenti una “testimonianza” e una verifica dell’adempimento del software ai requisiti specificati nel modello e pertanto questo momento del ciclo di sviluppo è sicuramente quello ideale per la sua produzione. Si noti inoltre come il confronto fra modello e codice si svolga tramite tecniche di “reverse-engineering” (e come questo richieda strumenti adeguati).

Il processo appena descritto si riassume nella Figura 4-1.

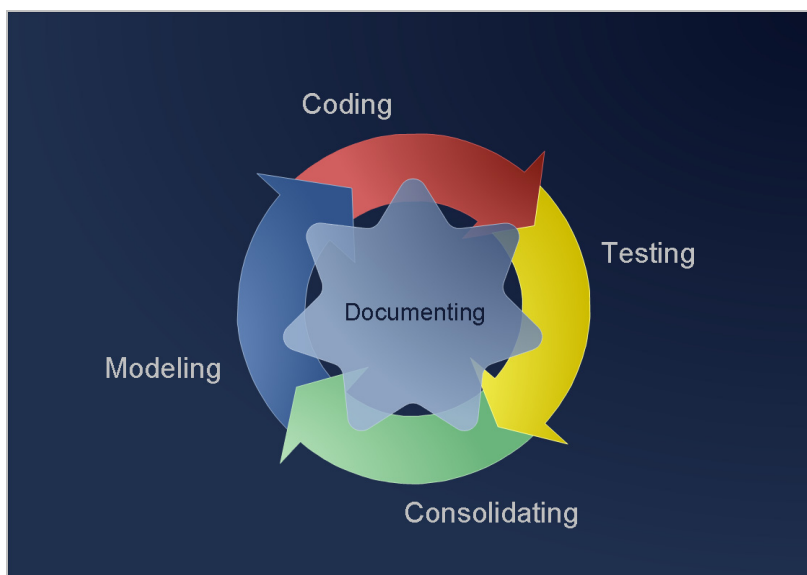


Figura 4-1 - Processo di sviluppo adottato per il sistema ADES

All'atto pratico un simile approccio si risolve nell'impiego combinato di svariati tools di sviluppo, fra i quali:

- **CruiseControl.NET** ([www.thoughtworks.com](http://www.thoughtworks.com)) è il sistema di “continuous-integration” che raccoglie e coordina il funzionamento di tutti gli altri tools, gestendo la costruzione automatica di ADES a intervalli regolari (decisi da un timer) e notificando la buona riuscita o meno dell'operazione (e documentando ogni passaggio e ogni fase del processo in una forma semplice, schematica e accessibile tramite una semplice interfaccia web).
- **NAnt** ([nant.sourceforge.net](http://nant.sourceforge.net)) è il sistema di build che si fa carico di compilare e testare il progetto (per quanto riguarda la fase di testing si veda il punto successivo). È integrato in CruiseControl.NET e si occupa di compilare, testare, pubblicare e salvare una copia di backup del codice. Questo significa che, poiché il processo viene ripetuto a intervalli regolari di tempo (tipicamente pochi minuti), si conserva del sistema una “history” e del codice pubblicato e delle copie di backup. Ciò che ne risulta è un innegabile sicurezza (poiché il salvataggio viene replicato su diversi sistemi di archiviazione in maniera completamente automatica) e la possibilità di ripristinare il proprio lavoro da un punto ben preciso senza perdere che pochi minuti per un'operazione di “roll-back” (ripristino a condizioni precedenti).

- **NUnit** ([www.nunit.org](http://www.nunit.org)) è il software che si occupa della fase di testing del sistema, per il quale è stata scritta e continua ad essere aggiornata una suite di tests che controllano le unità funzionali che lo compongono (operazione questa detta di “unit-testing”). Va da se che il ripetersi dei tests per ogni build del sistema comporti l'immediata identificazione di problemi derivanti dal commit di nuovo codice che eventualmente “de-allinei” o comprometta la precedente base di sorgenti; altrettanto ovviamente questo permette la risoluzione dei problemi non appena essi si presentano e scarica dal programmatore la responsabilità di effettuare i tests a ogni nuova build, perché è il sistema automatico a prendersi carico di compilare, eseguire e presentare i risultati dei tests.
- **Microsoft® Visio® 2003** ([office.microsoft.com](http://office.microsoft.com)) è il tool deputato alla creazione dei diagrammi UML (ma non solo) che hanno costituito le fasi di analisi e progettazione prima e quella di allineamento e sincronizzazione codice-modello poi. L'integrazione di Visio con Visual Studio ha permesso una “bidirezionalità” nella scrittura del codice che è stata determinante, oltre che per il già citato confronto continuo tra codice e modello, anche per la documentazione del software. In particolare, la possibilità di effettuare il “reverse-engineering” del codice dall'ambiente Visual Studio a quello di Visio ha costituito un vantaggio fondamentale nella fase di “consolidamento” del sistema nei confronti del modello.
- **Microsoft® Visual Studio® .NET 2003** ([msdn.microsoft.com/vstudio](http://msdn.microsoft.com/vstudio)), in particolare Visual C#® 2003, è l'ambiente all'interno del quale è avvenuta la scrittura del codice. Usato in combinazione con Visio ha permesso un approccio moderno e flessibile alla costruzione del sistema, supportato anche dalla copiosa documentazione fornita a corredo del .NET Framework.

- **NDoc** ([ndoc.sourceforge.net](http://ndoc.sourceforge.net)) è l'applicativo che permette la produzione della documentazione, rendendola disponibile in svariati formati. La sua importanza è stata tale che parti della presente relazione sono state schematizzate e sono “emerse” come fondamentali per una descrizione esauriente e compiuta di ADES già nella documentazione del codice che si è cercato di mantenere il più possibile completa e di alto livello fin dalle prime iterazioni dello sviluppo.

Un ultimo appunto vorrebbe essere un chiarimento sulla possibile ambiguità tra le espressioni “ciclo di vita dello sviluppo” e “ciclo di vita del software”; ovviamente in tutto l'arco di questo paragrafo ci si riferisce al ciclo di vita dello sviluppo, mentre resta sottinteso che l'insieme delle iterazioni dei vari cicli di sviluppo va a formare parte dell'arco di vita del software (la seconda espressione).

## **4.2. Dominio di utilizzo del sistema ADES**

L'utilizzo del software ADES, come già detto, mira a rendere più semplice la programmazione ad alto livello in un ambiente distribuito. “Ambiente distribuito” è un'espressione alla quale si è fatto ricorso spesso in precedenza ma, arrivati a questo punto della trattazione, deve essere circoscritta e meglio definita.

Ciò è necessario perché altrimenti si rischierebbe di perdere di vista quello che ADES “può fare” e quello che invece non “deve fare”.

ADES è un software pensato per facilitare la realizzazione di sistemi distribuiti orientati ai servizi, sistemi che hanno loro naturale collocazione in una rete informatica.

Non si parla di una qualsiasi rete ma di una tipologia ben precisa: ADES opera solo su reti TCP/IP.



Questo essenzialmente significa che gran parte delle reti odierne potrebbero ospitare con successo un'installazione di ADES.

Nonostante questo quando si è pensato e progettato ADES si pensava (e si pensa tuttora) a due tipologie di reti ben precise: le LAN aziendali (soprattutto le LAN Ethernet) e Internet. Il perché di un simile obiettivo dovrebbe essere chiaro dall'introduzione a questo lavoro, nella quale si è già abbondantemente discusso di integrazione di applicazioni.

Graficamente il suddetto “target” di impiego può essere riassunto nella Figura 4-2.

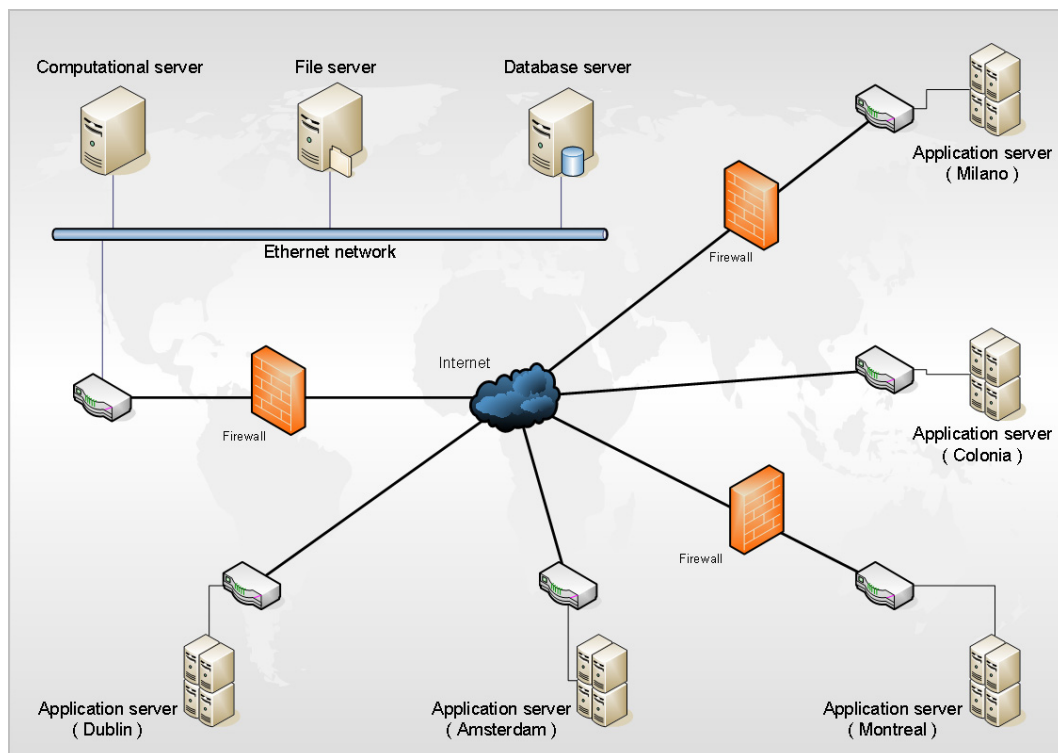


Figura 4-2 - Scenari di utilizzo del sistema ADES

La Figura 4-2 può essere suddivisa in due aree:

- In alto a sinistra abbiamo un tipico ambiente interno a una compagnia, basato su una rete Ethernet, alcuni server e collegato alla rete Internet.
- Proprio la rete Internet rappresenta l'altra area di interesse ed è rappresentata nelle restanti porzioni (in basso e a destra) della figura.

Sono presenti anche dei firewall e dei router che completano uno schema semplificato di quella che potrebbe essere la distribuzione fisica del sistema IT di una compagnia con sedi in diverse parti del mondo.

ADES è collocabile nelle due aree sopraindicate e, nel caso di una rete LAN interna alla compagnia, potrebbe ad esempio provvedere al brokering dei servizi che sono presenti sui vari server della stessa, mentre potrebbe essere utilizzato all'esterno di tale rete come "canale" tra i vari application servers che espongono le logiche di business di ciascuna sede.

Nella Figura 4-3 diventa chiaro il ruolo di ADES e come esso permetta di offrire una vista quasi "locale" dei servizi tra loro: in effetti la comunicazione avviene come se i due servizi interagenti fossero sulla stessa macchina.

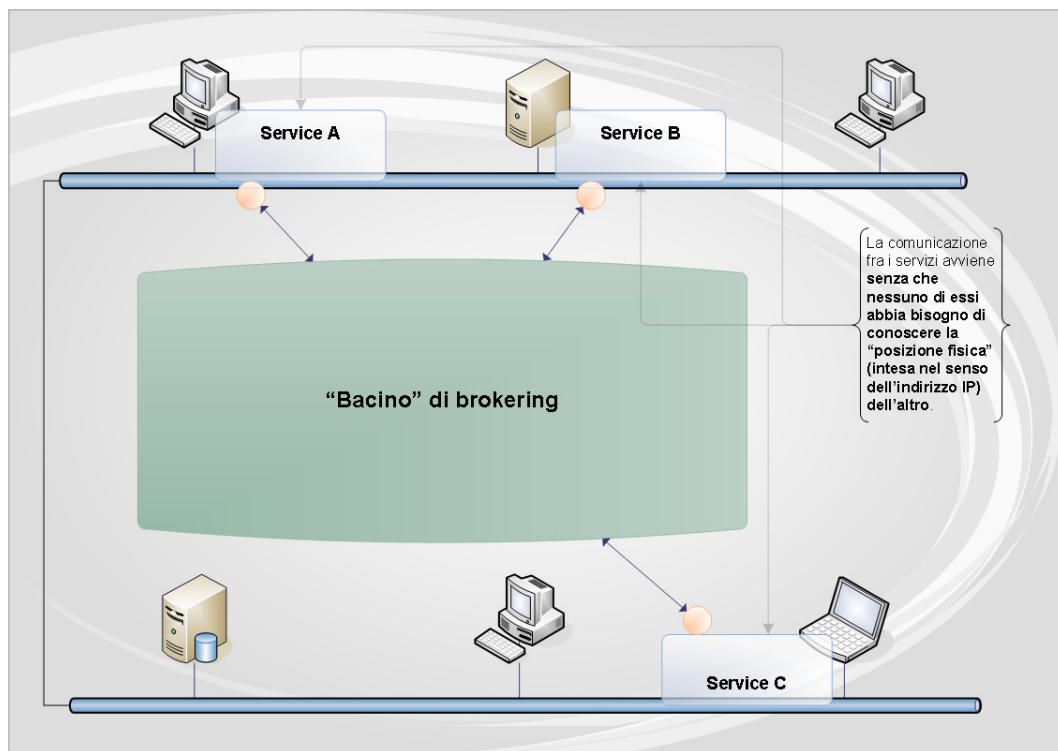


Figura 4-3 - Scenario Intranet di impiego del sistema ADES

Per quanto riguarda un contesto “extranet” di impiego, ADES agisce esattamente allo stesso modo. D'altronde in entrambi i casi ci si basa su end-points identificati da indirizzi IP.

A questo punto sarebbe naturale chiedersi che ne sarebbe di un servizio ospitato da ADES se variasse il suo indirizzo IP (domanda questa che ci rimanda direttamente alle riflessioni del paragrafo 2.1, a pag. 28).

Seppure basterebbe “scaricare” e “ricaricare” il servizio con il semplice cambiamento dell'indirizzo (funzionalità questa supportata dal sistema, ma da eseguirsi manualmente e da parte dell'utente), viene offerta la possibilità di utilizzare, all'atto della pubblicazione dei servizi, non solo indirizzi IP ma anche nomi DNS (“Domain Name System”, che ricordiamo essere, in breve, il servizio distribuito deputato alla risoluzione di nomi di dominio in indirizzi IP).

Questo significa che un servizio è localizzato da ADES indifferentemente sia ricorrendo a nomi DNS sia a indirizzi IP; cosa questa che risolve il problema in maniera semplice ed efficace.

Per quanto riguarda la questione firewall si rimanda alla discussione affrontata nel paragrafo 3.3 (a pag. 40) e al resto della trattazione.

### **4.3. Il sistema ADES nella fase di Analisi funzionale**

Una volta che si sia stabilito come procedere nello sviluppo del sistema e quale sia il suo dominio applicativo si intraprende la difficile e “rischiosa” fase di modellazione del sistema.

Secondo alcuni concetti tra i più utilizzati al momento è questa una fase che dovrebbe essere guidata dall'analisi delle funzionalità che il sistema dovrebbe offrire, ponendosi nei confronti di esso come se si considerasse una scatola nera ("Black-box") che acquisisce dati, li elabora e restituisce determinati risultati.

A questo proposito un tipo di approccio "Use-Case driven" (letteralmente "guidato dai casi d'uso") è di duplice importanza: permette innanzitutto l'esecuzione della suddetta analisi del sistema e lo fa aderendo ai formalismi riconosciuti e consolidati dell'UML.

È opportuno precisare che la scelta della versione 1.4 è stata dettata soprattutto dall'ancora relativa novità della versione 2.0 dello standard e in parte anche dalla maggior disponibilità di tools di modellazione.

Detto questo si può entrare nel vivo della trattazione.

\*\*\*

ADES è, dal punto di vista delle funzionalità principali, deputato a risolvere una sola categoria di problemi: **rendere del tutto trasparente agli agenti software che lo utilizzino la locazione "fisica" dei servizi dei quali essi fanno richiesta, nonché lo spostamento, l'eliminazione o la modifica degli stessi (servizi).**

Nonostante l'obiettivo sia uno solo le attività collaterali che esso richiede non sono poche e richiedono che ADES soddisfi altri requisiti funzionali.

Questi sono schematizzati nella Figura 4-4.

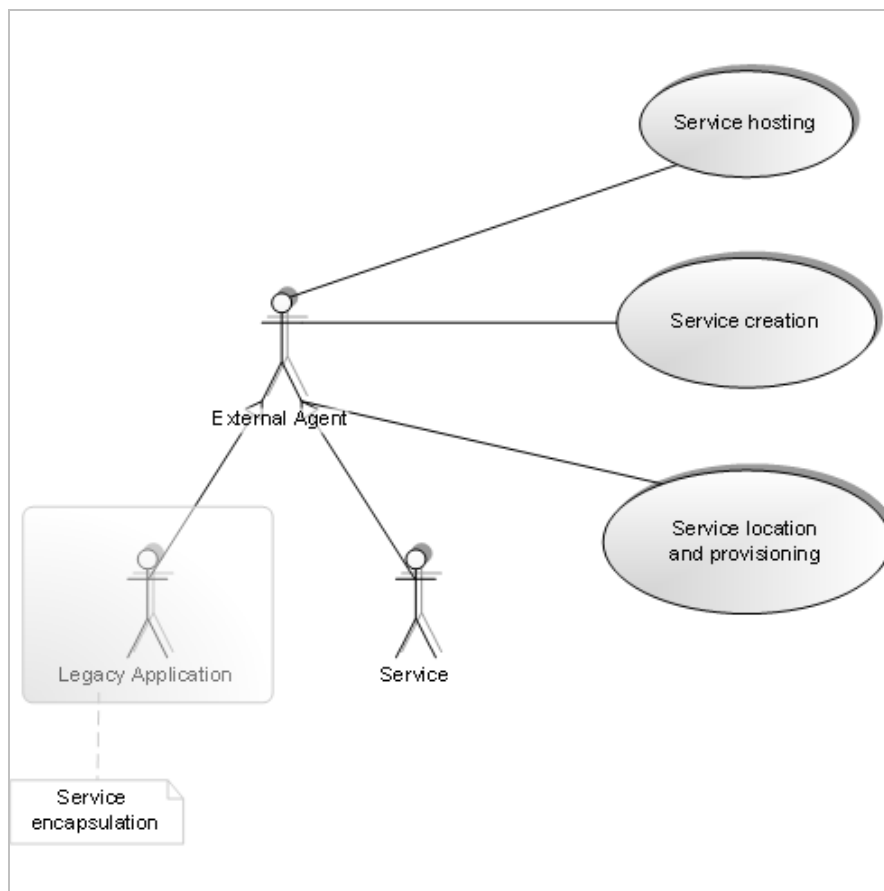


Figura 4-4 - Funzionalità principali del sistema ADES

L’esplicitarsi di questi requisiti in altri più dettagliati costituisce la vista dei casi d’uso del sistema ADES e permette il passaggio da un’analisi condotta dall’esterno del sistema a una più profonda, che coinvolge anche la maggior parte (seppur non tutte) delle dinamiche di funzionamento interne ad esso.

Come si può notare dalla Figura 4-4 è possibile distinguere tra i requisiti che ADES deve soddisfare tre macro-categorie (o anche “macro-aree” di funzionalità):

- **Service hosting**, vale a dire l’attività con la quale il sistema si prende carico di eseguire e rendere disponibile all’ambiente esterno un dato servizio.

- **Service location and provisioning**, cioè l'attività principale alla quale è dedicato il sistema e della quale si è discusso a livello teorico nel Capitolo 2 (a pag. 28).
- **Service creation**, ADES prevede alcune facilitazioni per la creazione dei servizi, liberando lo sviluppatore dall'onere di “re-inventare la ruota” ogni volta. Questo significa essenzialmente che il framework ADES fornisce un'astrazione di servizio (già dotata di alcune funzionalità essenziali) che potrà essere successivamente estesa a seconda delle esigenze.

Per ciascuna delle tre macro-categorie si tratta essenzialmente di scendere nel dettaglio, ramificando i casi d'uso e dettagliando il finora “grezzo” modello del sistema.

Per la prima macro-area di funzionalità, quella dell'hosting di servizi, una possibile ripartizione dei casi d'uso potrebbe essere quella in Figura 4-5 (di solito la soluzione ottimale a un problema non è una sola e in questo senso va inteso il termine “potrebbe”).

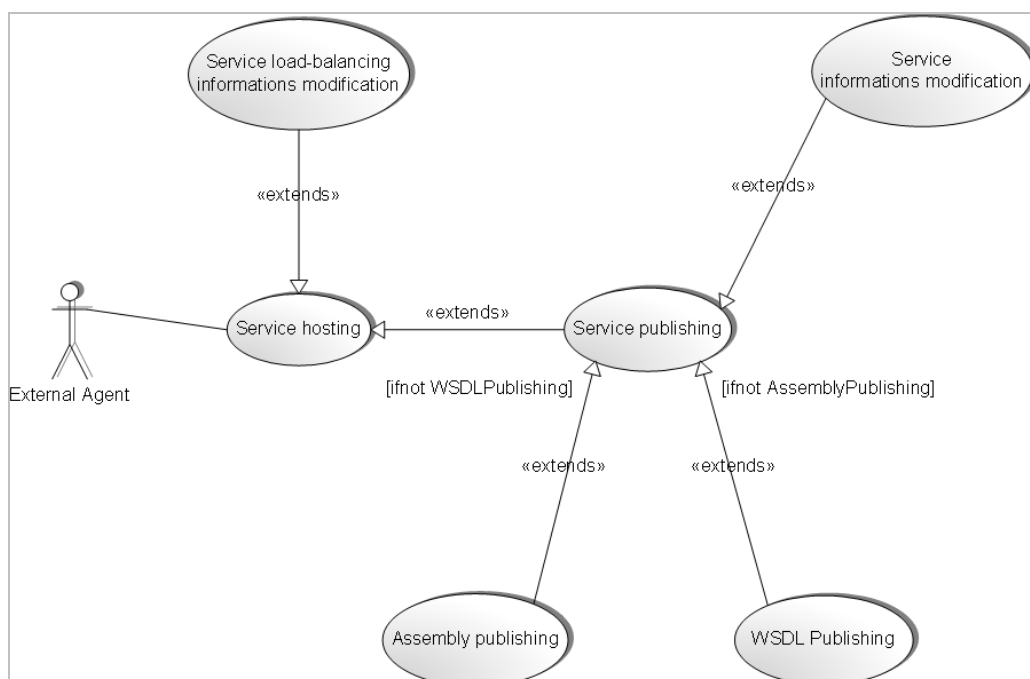


Figura 4-5 - Funzionalità di service hosting del sistema ADES

Benché possa sembrare intuitiva e semplice, l'interpretazione di tale diagramma non è banale. In primo luogo per la presenza di numerose relazioni di “extends” tra i casi d'uso.

A questo proposito è utile ricordare come tale tipo di relazione esprima una “estensione” di una data funzionalità con delle altre, vincolando però quest'estensione al verificarsi o meno di una ben determinata condizione.

Nella Figura 4-5 le relazioni di “extends” sono tutte vincolate a una ben precisa condizione (in alcuni casi tale condizione è stata esplicitata mentre in altri non si è fatto altrettanto per una questione di leggibilità del diagramma).

Quello che si profila da una lettura attenta della Figura 4-5 è uno scenario in cui ADES provvede a ospitare i servizi, pubblicandoli poi a tutta la rete che utilizzi la tecnologia ADES.

Tutte le relazioni di “extends” sono le opzioni che ADES permette di applicare alla pubblicazione di un servizio; esse sono riassunte e discusse nel seguente elenco:

- **Service hosting:** si tratta della sola attività di ospitare il servizio all'interno dell'ambiente ADES. Presa singolarmente essa è praticamente inutile, ma rappresenta un'unità atomica e ben distinta delle funzionalità offerte da ADES e in tal senso è stata modellizzata nel diagramma.
- **Service publishing:** è uno dei passaggi fondamentali del processo di funzionamento e ha comportato anche non pochi problemi dal punto di vista implementativo. Si tratta di estrarre dal servizio le informazioni rilevanti ai fini del suo utilizzo remoto, cosa questa che si riconduce a diffondere a tutti i nodi della rete ADES una qualche rappresentazione di queste informazioni. Nei due punti successivi si analizzeranno i possibili modi di strutturare tali informazioni (la possibilità di scelta tra l'uno e l'altro è lasciata all'utente).

- **Assembly publishing:** un servizio può essere pubblicato sottoforma di “assembly” .NET (si tratta di un file eseguibile o di libreria compilato da un linguaggio .NET) e in tal caso l'utilizzo da parte dei nodi destinatari della pubblicazione sarà immediato: basterà fare riferimento all'assembly (che ADES provvederà a salvare su disco) per poter utilizzare il servizio. Come fare riferimento al servizio sottoforma di assembly si vedrà in seguito quando si parlerà delle funzionalità di brokering di ADES.
- **WSDL publishing:** è questa una funzionalità pensata per fornire interoperabilità fra ADES e il mondo dei Web Services. Si tratta infatti di pubblicare un servizio fornendone una descrizione WSDL (“Web Service Description Language”, come già visto nei primi due capitoli). Questa funzionalità non è però implementata all'interno di ADES, ma è agevolmente accessibile facendo riferimento al servizio pubblicato attraverso un particolare formato della stringa di connessione al servizio (utilizzabile in qualsiasi browser o da codice, stabilendo una connessione con l'host che pubblica il servizio e purchè venga utilizzato una comunicazione HTTP con esso); si vedrà nei paragrafi successivi l'implementazione dettagliata di questo meccanismo.
- **Service informations modification:** è naturale offrire all'utente la possibilità di modificare alcune proprietà relative ai servizi pubblicati. In tal senso si è pensato di offrire un modo semplice per farlo evitando il ricaricamento del servizio (ma solo la sua ripubblicazione).
- **Service load-balancing informations modification:** ADES è stato pensato per gestire situazioni di load-balancing (“bilanciamento del carico”) dei servizi. Questo vuol dire che il sistema deve tenere traccia di alcune informazioni in proposito e permettere ovviamente la loro modifica.



Questo per quanto riguarda l'hosting dei servizi. Nel paragrafo 4.5 (a pag. 63) si discuterà di come questi requisiti funzionali siano stati effettivamente implementati in ADES. Così si farà pure nel seguito, differendo ogni volta la discussione del “come” si siano implementate le funzionalità discusse allo studio della progettazione del sistema (studio svolto nel paragrafo 4.5).

Ora si tornerà invece all'analisi di un'altra macro-area di funzionalità del sistema, quella di creazione dei servizi.

Quest'area fornisce allo sviluppatore la possibilità di creare servizi basati sulla tecnologia ADES; servizi che andranno poi a essere caricati nell'ambiente di esecuzione e pubblicati a tutti i nodi della rete.

Si è pensato che non sarebbe stato necessario fornire un ambiente integrato in ADES per la creazione di servizi e soprattutto si è supposto che sarebbe stato addirittura dannoso fornire una qualche sorta di linguaggio di marcatura o di formato XML (o altro) che indicasse ad ADES un particolare servizio e come caricarlo.

Preferendo rimanere sul terreno dell'integrazione (e della semplicità, anche di utilizzo) si è offerta la possibilità allo sviluppatore di creare assembly .NET con qualsiasi ambiente egli preferisca, vincolandolo a una sola condizione: il servizio sviluppato deve ereditare da una particolare classe del framework ADES, una classe che rappresenta un servizio astratto e che permette il riconoscimento da parte dell'environment ADES dei servizi “ADES-enabled”.

Questa semplicità traspare evidentemente dalla Figura 4-6.

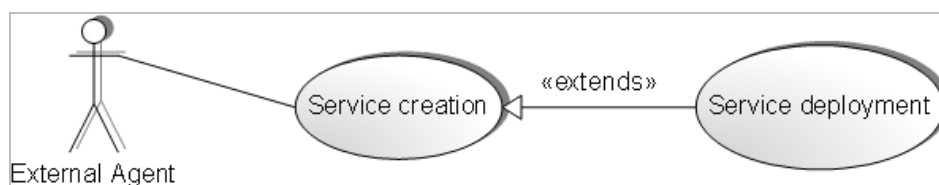


Figura 4-6 - Supporto alla creazione dei servizi da parte del sistema ADES

Si è trattato in questo caso di una scelta da intraprendersi immediatamente a cavallo delle due fasi, l'una di Analisi e l'altra di Progettazione.

\*\*\*

Non resta che analizzare le funzionalità principali del sistema, quelle di brokering dei servizi.

Esse sono riassunte nel diagramma UML di Figura 4-7.

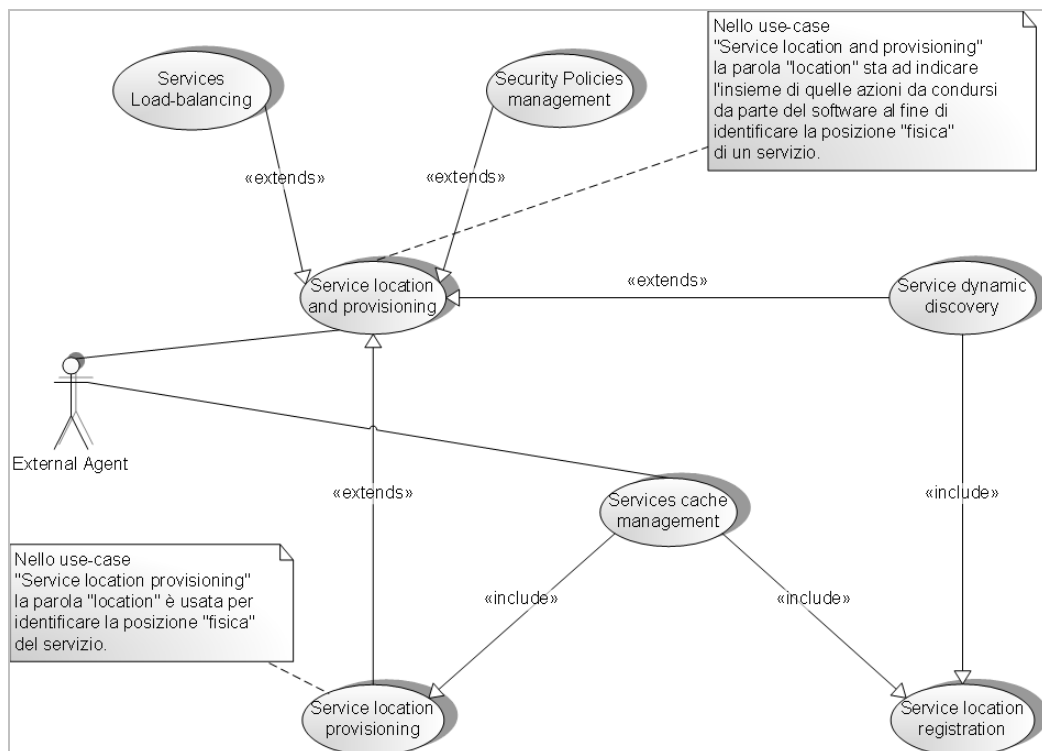


Figura 4-7 - Funzionalità di service brokering del sistema ADES

Si nota subito come le relazioni di “extends” siano ora accompagnate da relazioni di “include”, le quali rappresentano l’arricchimento della funzionalità “includente” con le funzionalità “incluse”. A differenza delle relazioni di “extends” si parla in questo caso di inclusione “obbligatoria” e svincolata da qualsiasi condizione.

Se si pensa che questo è il cuore del sistema ADES si potrebbe dire come il numero delle relazioni tra i casi d’uso è forse anche troppo esiguo.

In realtà molti dei casi d'uso della Figura 4-7 esprimono un insieme di funzionalità che va sotto un'unica etichetta, ma che potrebbe essere scomposto in numerose sotto-unità.

Ancora una volta si invoca l'esigenza di rendere quanto più leggibile e il meno affollato possibile uno schema che ha come scopo principale quello di **illustrare con chiarezza** il ragionamento che sta dietro le scelte progettuali e architetturali alla base di ADES.

In fin dei conti la scelta di un tale livello di dettaglio a questo punto ha permesso il concentrarsi sui particolari al momento della progettazione (e di solito si tratta di una scelta, questa, che paga).

Nella Figura 4-7 compaiono per la prima volta quattro funzionalità molto importanti che meritano una trattazione approfondita:

- **Services Load-balancing** si occupa del bilanciamento del carico sui servizi, provvedendo al delivering di un servizio piuttosto che di un altro a seconda delle differenze nel carico di lavoro tra i due. Vengono a questo proposito conservate informazioni per ogni servizio, informazioni che sarà compito del servizio tenere aggiornate. È comunque possibile da parte dei servizi o da parte dell'utente rinunciare a questa funzionalità.
- **Security-policies management** ha a che vedere con alcune funzionalità che sono state previste per ADES, ma che non sono state implementate a causa dell'esiguità del tempo a disposizione e della loro complessità. Si tratta di meccanismi di protezione dei servizi che permettono la loro gestione come gruppi isolati, tra i quali la comunicazione è gestita appunto tramite politiche di sicurezza differenti. L'implementazione richiede uno o più servizi ADES-enabled che si occupino di tenere traccia dei gruppi e dell'appartenenza dei servizi a ciascun gruppo. Si vedrà comunque di più a proposito della gestione delle politiche di sicurezza nel paragrafo 4.5.

- **Service dynamic discovery** è forse la funzionalità di più alto livello che ADES potrebbe offrire ma, per gli stessi motivi del punto precedente, essa resta perlomeno per ora sulla carta. Non si poteva però tralasciare di modellarla e presentarla perché rappresenta la possibilità che ADES offrirà nel prossimo futuro di gestire un'installazione del sistema su una rete senza che ci si debba preoccupare di configurare ogni nodo della rete con gli indirizzi (IP o DNS) degli altri nodi. Come già detto, questo è un qualcosa di assolutamente desiderabile in un sistema che si propone ad alto livello di astrazione “user-oriented”.
- **Services-cache management** è l'insieme delle funzionalità che gestiscono la cache delle informazioni sui servizi; informazioni che saranno scambiate o elaborate nei nodi ADES ai fini di provvedere alla localizzazione dei servizi. Si tratta di un'attività molto “pesante”, poichè i dati e le informazioni sui servizi dei quali deve tener conto il sistema non sono pochi e la loro organizzazione deve garantire flessibilità, robustezza e una futura estensibilità del framework (aspetto questo che si è considerato irrinunciabile fin dalla fase di analisi).

Proprio per la complessità delle funzionalità della cache dei servizi si preferisce dedicare loro un ulteriore approfondimento, affidandolo alla Figura 4-8.

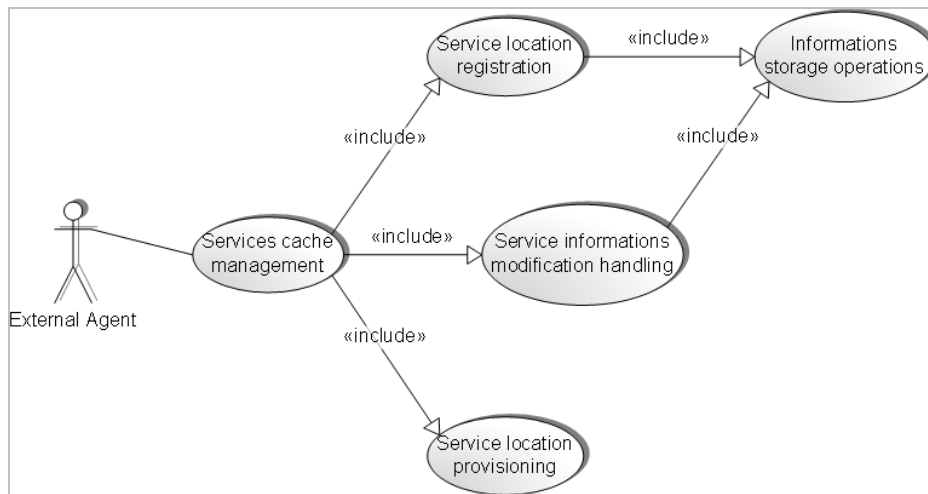


Figura 4-8 - Funzionalità di gestione della cache dei servizi

Si noti come venga previsto un meccanismo di persistenza per i dati riguardanti i servizi, precisamente nello use-case “Informations storage operations”.

E si consideri anche la presenza di sole relazioni di “include”, a riprova del fatto che la gestione della cache dei dati richiede un gran numero di operazioni.

Quasi superfluo precisare che tali funzionalità del sistema sono state riportate in Figura 4-7, in relazione alle attività di brokering del sistema, ma ovviamente esse sono da considerarsi trasversali e fondamentali per ogni parte dello stesso.

Nel corso della trattazione di questo paragrafo si è passati da un’analisi condotta “sulla superficie” del sistema a una che invece “attraversa” ed “esplora” i meccanismi interni ad esso. Si è però parlato solo del “cosa” sia richiesto al sistema, mentre si affida ai prossimi paragrafi la descrizione del “come” ADES adempia alle funzionalità richieste.

#### **4.4. La “dualità” del framework/environment ADES**

Innanzitutto si deve dire che le funzionalità presentate non sono un insieme omogeneo che ben si presta a essere fornito sotto l’egida di un unico strumento.

Ciò sarebbe auspicabile ma allo stesso tempo appesantirebbe e l’apprendimento da parte dell’utente di tutti gli aspetti del sistema e la costruzione dello stesso.

Poiché si è deciso fin dall’inizio di porre la semplicità davanti a tutto si è giunti alla conclusione che alcune funzionalità dovessero essere affidate a strumenti esterni a una eventuale libreria o eseguibile principale (si vedrà come si sia infine optato per una libreria).

È il caso dei servizi di “Dynamic-discovery”, che si è pensato di affidare a un “wizard” di installazione del sistema.

È il caso anche delle funzionalità concernenti la sicurezza, che si è deciso di implementare in minima parte all’interno del nucleo principale di ADES e di gestire invece estensivamente con un servizio esterno.

Si tratta di suddivisioni necessarie, che portano modularità all’ambiente e che separano gruppi di funzionalità con determinate caratteristiche (spesso difficilmente conciliabili tra loro).

Al momento della suddivisione di queste funzionalità si è potuto constatare come la differenza più importante sussistesse fra due gruppi di esse e come si trattasse di una differenza di tipo “dinamico”.

Vale a dire che alcune funzionalità, come quelle di creazione dei servizi, sono sostanzialmente “statiche” e fruibili “una volta e per sempre” (all’atto della creazione del servizio), mentre altre, quali i servizi di brokering, fossero da utilizzarsi più volte, ripetutamente e in modo ovviamente non predicibile.

Schematizzando, si tratta di funzionalità le prime che vengono rese disponibili da un **framework** “statico” (quindi disponibili a compile-time), mentre per le seconde si deve pensare a un **environment** dinamico e disponibile a runtime.

In questo concetto sta la doppia “personalità” del sistema ADES, dualità che permette la fruizione di funzionalità diverse in momenti diversi, ma potendo contare su un unico insieme di strumenti omogenei e fortemente integrati tra loro.

#### **4.5. Il sistema ADES nella fase di Progettazione**

Uno dei problemi più spinosi che il passaggio dalla fase di analisi a quella di progettazione ha portato è stato decidere che forma dare ai servizi dei quali si sarebbero gestite le attività.

Qualche riga fa si è parlato di assemblies .NET e si è perfino lasciato intendere che si sarebbe trattato precisamente di librerie (files “.dll” quindi).

Questi sono in realtà particolari che non rendono la vera difficoltà riscontrata nel modellare cosa il servizio sarebbe dovuto essere realmente; il problema infatti non è “che forma deve avere il servizio per il file-system e per l’ambiente di sviluppo adottato”, ma è “che forma e che relazioni un qualsiasi agente esterno che aspiri a diventare un servizio deve mostrare e intrattenere con il sistema ADES”.

Questo ragionamento si risolverà ovviamente nella descrizione fisica del servizio, ma le farà precedere una descrizione logico-funzionale che non è possibile differire o vincolare ad alcuna progettazione fisica del servizio.

Quando si pensa a cosa il sistema potrebbe “pretendere” da un servizio la prima delle attività che vengono in mente è quella che richiede da parte del servizio di farsi riconoscere; essenzialmente il problema è quindi informare ADES che si vorrebbe che un particolare elemento (file o documento, ma potrebbe anche trattarsi di dati salvati da qualche parte su un supporto di memorizzazione o altre possibilità) presente sulla macchina ospitante il sistema venga considerato come un servizio e come tale caricato e reso disponibile in tutta la rete.

Questo tipo di attività richiede una scelta e la scelta di ADES è richiedere unicamente che all’atto del caricamento del servizio venga specificato il nome del file che lo contiene.

Convenzionalmente il file è una DLL .NET il cui nome coincide con quello del servizio (nome con il quale il servizio verrà eventualmente pubblicato), ma si richiede che al suo interno un “tipo” (dal punto di vista del codice una “classe”) erediti dal modello di servizio che il framework ADES mette a disposizione.

Quello che avviene è schematizzato in Figura 4-9.

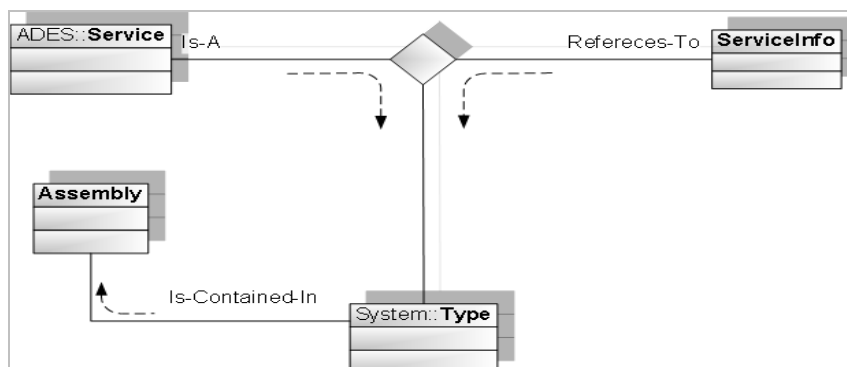


Figura 4-9 - Relazioni tra servizio e il sistema ADES

Per la corretta lettura del diagramma è richiesta una piccola digressione sulla libreria di classi del .NET framework.

Innanzitutto un assembly è un file su disco contenente codice destinato ad essere eseguito dal runtime .NET; è quindi naturale che un assembly contenga al suo interno dei tipi che forniscono determinate funzionalità.



Quando si ha bisogno di caricare dinamicamente un tipo da un file esterno si ricorre al meccanismo della “reflection” (presente in numerosi linguaggi, quali Java, C# e altri). Con tale meccanismo il contenuto di un assembly si presenta sottoforma di una collezione di tipi (derivanti da “System::Type”), caratterizzati dall’aver una particolare “signature” (o “firma”) e determinati rapporti di ereditarietà tra loro (e con le classi del .NET framework).

ADES, all’atto del caricamento di un assembly è capace di riconoscere in un tipo tra quelli presenti nel file un servizio che erediti dal prototipo di servizio principale del sistema, “ADES::Service”.

Ciò che naturalmente ne consegue è che il tipo in questione viene caricato dall’environment ADES e reso disponibile per la pubblicazione.

Questo però non spiega la presenza nel diagramma del tipo “ServiceInfo”. Ebbene si tratta di un tipo interno all’ADES framework, ogni istanza del quale tiene traccia di un particolare servizio.

L’associazione (da ora in avanti indissolubile) fra servizio caricato e oggetto di tipo “ServiceInfo” avviene all’atto del caricamento del servizio stesso.

Questo meccanismo, con un minimo di elaborazione, provvede al gravoso compito di caricare i servizi (che vengono ospitati in cartelle particolari su disco) e permette la gestione delle informazioni a loro correlate tramite semplici strutture dati che si intuisce facilmente costituire il principale oggetto di elaborazione da parte del sistema ADES e che forniscono l’implementazione dei requisiti funzionali di “Services cache management” della Figura 4-8 (paragrafo 4.3, a pag. 61). Il tutto va a costituire il sottosistema “ADES::ServicesCacheManagement”.

Ripetendo quanto già detto a proposito del modo in cui l’ambiente fa riferimento ai servizi, un’operazione di loading di un servizio si risolve nei seguenti passaggi logici:

- Viene caricato l’assembly contenente il tipo che implementi il servizio

- Viene individuato (tramite il meccanismo della “reflection”) all’interno di tale assembly il tipo che implementa il servizio
- Il tipo viene caricato all’interno del runtime .NET e reso disponibile per l’utilizzo remoto
- Il servizio così ottenuto verrà eventualmente pubblicato

Del procedimento appena illustrato sono state descritte nel dettaglio (nella Figura 4-9) le fasi 1, 3 e 4, ma della fase 2 ben poco è stato detto.

Sarebbe in effetti opportuno chiarire cosa sia il tipo “ADES::Service” e quali peculiarità esso abbia che permettono una fruizione remota dei servizi “user-defined” (definiti dall’utente) che lo estendono.

Il compito di chiarire questi aspetti spetta per la maggior parte alla Figura 4-10.

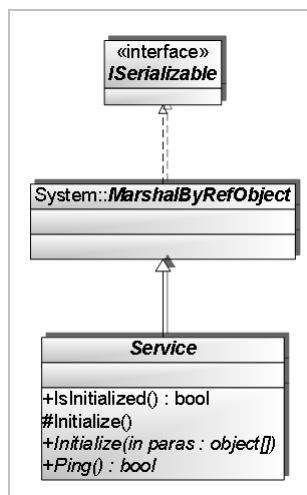


Figura 4-10 - Tipo "ADES::Service"

In essa possiamo vedere come “ADES::Service” sia solo la parte finale di una gerarchia per la maggior parte composta da oggetti messi a disposizione dal .NET framework.

Si tratta precisamente di: un'interfaccia “ISerializable” con la quale si segnala al framework che di un oggetto può essere salvato lo stato, in modo da garantirne la persistenza o la capacità di poter essere trasportato attraverso una rete (ed è questo in effetti il caso preso in esame); al di sotto di “ISerializable” sta poi il tipo astratto “MarshalByRefObject”, una classe molto importante del framework .NET che permette la creazione di oggetti cosiddetti “remotizzabili”.

Qui è doveroso un breve excursus sulla tecnologia sottostante al sistema ADES e che ne rende possibile il funzionamento, .NET Remoting.

Un oggetto è remotizzabile quando è possibile “passarlo” ad ambiti esterni a quello di esecuzione del corrente processo e/o della macchina che il processo esegue.

È questo il caso di oggetti che debbano essere messi a disposizione in un contesto distribuito per l'utilizzo da parte di applicativi che risiedano su macchine diverse da quella che “espone” l'oggetto, ma è anche il caso dei problemi di IPC (“Inter-Process Communication”, comunicazione inter-processo) che spesso sono una vera spina nel fianco per gli sviluppatori.

I meccanismi messi a disposizione dalla tecnologia .NET Remoting mirano a rendere molto più semplici la risoluzione di questi problemi e sono alla base dell'implementazione Microsoft dei WebServices. Essi si basano pesantemente sull'oggetto “MarshalByRefObject” che, opportunamente implementato in una classe, dà la capacità agli oggetti sue istanze di essere “pubblicati” su una rete e utilizzati da un contesto remoto (rispetto a quello nel quale essi si trovano).

Quest'ultima affermazione chiarisce quindi il ruolo che ha nella gerarchia di Figura 4-10 il tipo “ADES::Service”: esso permette, alle classi che lo implementino, **di essere pubblicate per l'utilizzo distribuito**.

Ora è chiaro anche un secondo (e forse più importante) motivo per il quale i servizi che vogliano essere utilizzati all'interno di ADES debbono discendere dalla classe "ADES::Service": questo è un modo per fornir loro la possibilità di varcare le soglie del dominio nel quale vengono eseguiti e per poter essere utilizzati in contesti remoti (rispetto a quel dominio).

Oltre a ciò la classe "ADES::Service" fornisce anche alcuni metodi di utilità, quale la proprietà `IsInitialized`, e una sorta di "contratto" definito dalla presenza di alcuni metodi astratti che devono essere implementati dalle classi derivate (`Initialize()` e `Ping()`).

Nel seguito si vedrà come le attività collaterali a questo meccanismo lo nascondano completamente all'utente finale, al quale è dato esclusivamente di sapere che, implementando la classe "ADES::Service" e racchiudendola in un assembly, otterrà il suo caricamento (con i meccanismi che vedremo in seguito) come servizio e la relativa pubblicazione.

A proposito di queste attività collaterali è indispensabile vedere subito quali sottosistemi si siano individuati come possibile "copertura" delle funzionalità richieste dai casi d'uso del paragrafo 4.3. Essi sono presentati nella Figura 4-11.

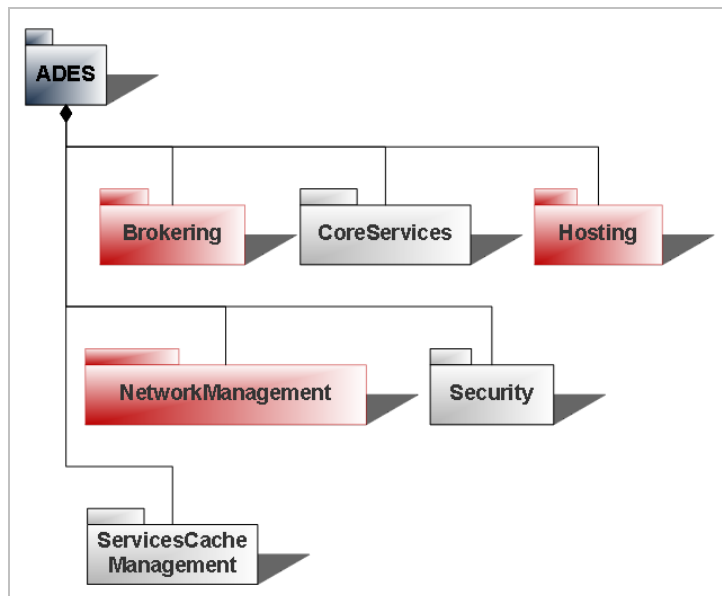


Figura 4-11 - Sottosistemi del sistema ADES

Nella figura sono evidenziati in rosso i sottosistemi dei quali si porterà a conoscenza l'utente e nei quali sono presenti gli entry-points attraverso i quali è possibile utilizzare le funzionalità di ADES.

È inoltre da far notare come anche il sottosistema “ADES”, radice della gerarchia, contenga classi e altri tipi, non riportati però in figura.

Prima di passare all'attenta disamina di ciascun sottosistema, si precisa che d'ora in avanti i termini “Sottosistema” e “Namespace” (letteralmente “Spazio dei nomi”) verranno utilizzati indifferentemente e con lo stesso significato.

Nel seguito verranno esaminati tutti i namespaces della gerarchia in Figura 4-11 e, per ognuno di essi, verranno chiariti composizione e quali funzionalità del modello di Analisi esso implementi.

\*\*\*

Si inizierà con il namespace “ADES::ServicesCacheManagement”, al quale sarà dedicata una serie di diagrammi che spiegheranno come questo sottosistema gestisca le strutture dati perno del sistema, quelle relative ai servizi.

Nel primo diagramma della serie, nella Figura 4-12, si può notare come la classe “ServiceInfo” sia stata specializzata in due classi figlie ognuna delle quali si occupa di gestire una ben distinta situazione:

- **ExternalServiceInfo** si occupa di gestire i riferimenti e i dati riguardanti servizi “esterni” alla macchina che esegue il sistema ADES. In buona sostanza si tratta di una struttura dati che tiene traccia dei servizi invocabili dal sistema ADES, ma presenti su altri nodi della rete.
- **HostedServiceInfo** è deputata alla gestione dei servizi che sono stati caricati localmente al sistema ADES e che verranno eventualmente pubblicati in tutta la rete.

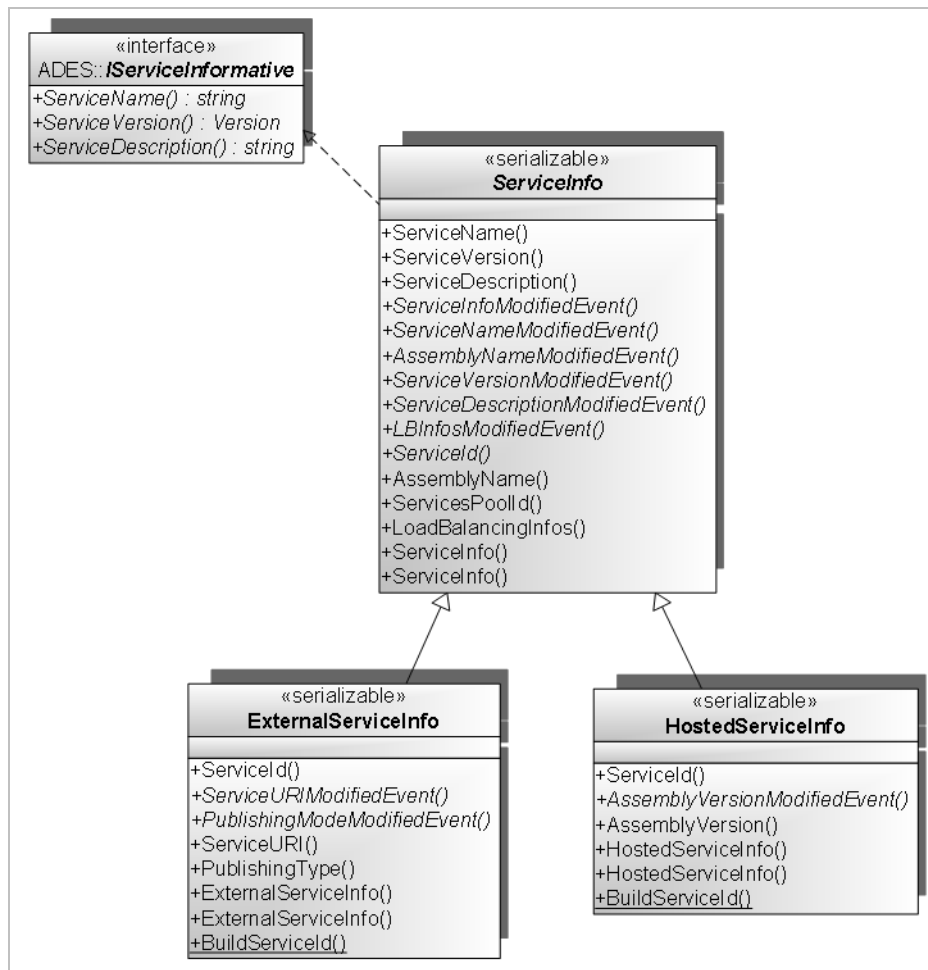


Figura 4-12 - Gerarchia di ereditarietà delle classi ServiceInfo

In Figura 4-12 è presente anche un tipo (si tratta in questo caso di un interfaccia), “**IServiceInformative**”, il cui compito è innanzitutto fornire coerenza all’attuale e alle future versioni di ADES.

Si tratta di un’interfaccia che specifica quali proprietà devono essere fornite da un tipo che desideri incapsulare e rendere disponibili dati riguardanti i servizi (siano essi “hosted” o “external”).

Le informazioni sono:

- `ServiceName` che si occupa di fornire il nome del servizio
- `ServiceVersion` che fornisce la versione del servizio
- `ServiceDescription` che ne da una descrizione di tipo testuale e a livello informativo

È poi presente una proprietà molto importante che permette di ottenere alcune informazioni rilevanti ai fini della realizzazione delle politiche di bilanciamento del carico che ADES si propone di offrire. Questa proprietà è `LoadBalancingInfos` e contiene un oggetto di tipo “`LBServiceInfo`”, presentato nella Figura 4-13.

Esso appartiene al namespace “`ADES::Brokering::LoadBalancing`”.

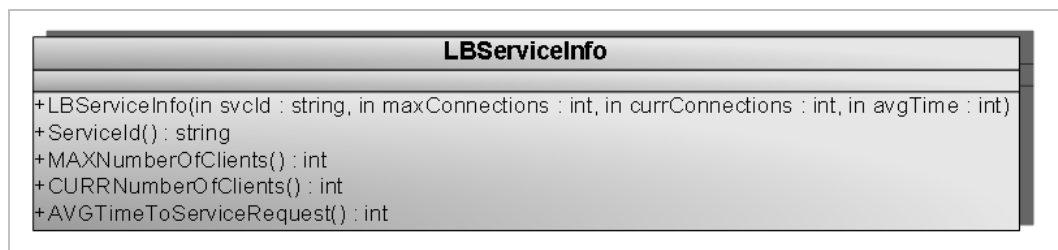


Figura 4-13 - Tipo "LBServiceInfo"

Il bilanciamento del carico è un innegabile vantaggio, ma richiede attiva collaborazione da parte dei servizi, ai quali in fase di inizializzazione viene dato modo di poter fare riferimento a se stessi nell'ambiente ADES, in modo da poter così accedere e modificare le informazioni di load-balancing.

Qualora comunque non si desiderasse usufruire delle funzionalità di bilanciamento del carico si può tranquillamente tralasciare di utilizzare tale proprietà.

Procedendo nella trattazione, la Figura 4-14 rappresenta le eccezioni di questo sottosistema, tutte derivanti da “`BasicException`”, presente nel namespace “`ADES`” e radice della gerarchia di tutte le eccezioni del sistema ADES.

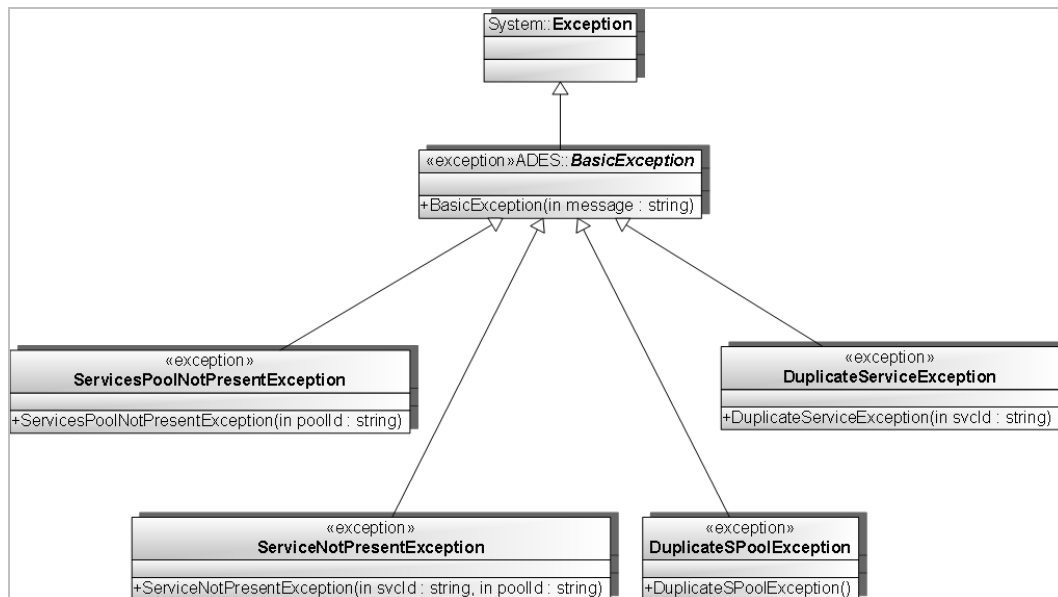


Figura 4-14 - Gerarchia di eccezioni del sottosistema "ADES::ServicesCacheManagement"

“ADES::BasicException” è un tipo astratto che fornisce solo alcune funzionalità di base, non può essere istanziato, ma è cruciale per fornire un modello unico, coerente e omogeneo per la gestione delle eccezioni nel sistema ADES.

Questo essenzialmente significa che tutte le eccezioni del sistema ADES debbono derivare da “BasicException”, di modo da fornire la consistenza e l’omogeneità che si dicevano.

Proseguendo, la Figura 4-15 rappresenta una classe fondamentale per la gestione di collections di servizi. È infatti impensabile non fornire un sistema per raggruppare (e anche isolare) servizi che abbiano proprietà comuni.

Si è pertanto realizzato un contenitore (una pool) di servizi che ne semplificasse la gestione in tal senso.

Inoltre, come si vedrà meglio in seguito, una simile ripartizione favorisce anche la gestione delle “Security policies” per il sistema ADES.



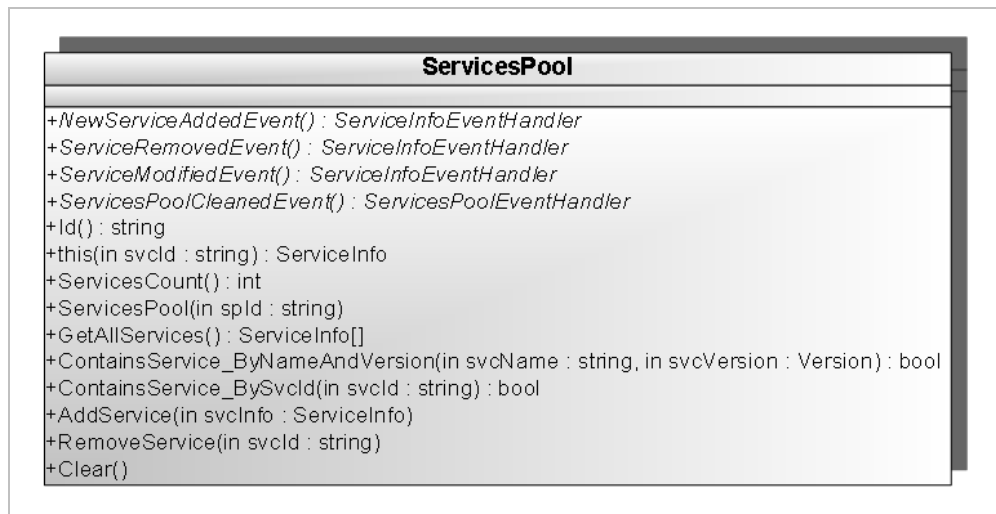


Figura 4-15 - Classe "ServicesPool"

La Figura 4-15 descrive alcune operazioni che permettono la gestione di un insieme di servizi, tra le quali:

- Operazioni di ricerca sull'insieme dei servizi:
  - `ContainsService_ByNameAndVersion()`
  - `ContainsService_BySvcId()`
  - l'indicizzatore `this []`, un particolare costrutto del linguaggio C# che permette l'utilizzo di un insieme di dati come fosse un array associativo
- Operazioni che apportano modifiche alla pool:
  - `Id`, una proprietà rappresentante un identificativo univoco per un oggetto "ServicesPool"
  - `Clear()`, che permette la rimozione di tutti i servizi presenti
  - `ServicesCount`
- Operazioni che agiscono sui servizi:
  - `AddService()`
  - `RemoveService()`

Una simile struttura dati si è rivelata molto preziosa per modellare la gestione dei servizi e nonostante abbia introdotto un ulteriore livello concettuale, sia dal punto di vista dello sviluppo sia da quello dell'interfaccia offerta all'utente, mette a disposizione alcuni interessanti scenari di gestione di insiemi di servizi.

Uno di questi scenari viene presentato nella Figura 4-16 e ha permesso, attraverso la realizzazione di alcune gerarchie di ereditarietà, di raggruppare grandi quantità di codice di gestione in oggetti riutilizzabili.

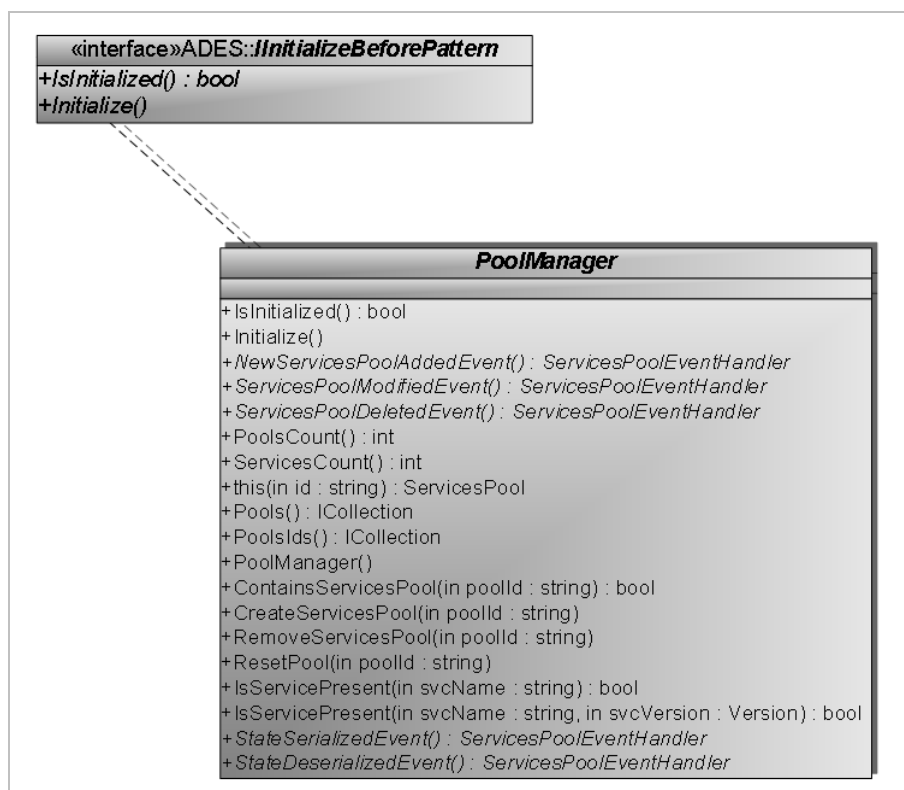


Figura 4-16 - Classe "PoolManager"

La Figura 4-16 rappresenta una classe astratta che assomma in se molte delle operazioni di gestione di più pools di servizi (si noti poi come essa implementi un pattern, "IInitializeBefore", che verrà discusso nel paragrafo 5.3).

Essa basta a consentire una gestione centralizzata delle stesse, in quanto le classi che da essa derivano potranno usufruire di un unico modello gestionale.

Tra i numerosi metodi di utilità offerti dal “PoolManager” sono riconoscibili tre categorie ben precise che permettono di:

- Ricercare particolari servizi o determinate pools
- Creare, rimuovere o modificare le pools
- Controllare, tramite svariati eventi, eventuali modifiche a esse

Viene infine incapsulato, tra le funzionalità di questa classe, un meccanismo per la “serializzazione” (operazione di salvataggio dello stato di un albero di oggetti su un supporto di memorizzazione) delle pools; meccanismo questo offerto in modo completamente trasparente all’utente e allo sviluppatore delle classi che da “PoolManager” derivino.

Meccanismo che permette di salvare lo stato delle informazioni correlate ai servizi, garantendo loro la persistenza anche tra un’esecuzione e l’altra del sistema ADES.

Comunque sia, la complessità di questo sottosistema è solo apparente, in quanto si può benissimo notare dalla Figura 4-17 come gli oggetti con i quali si avrà a che fare siano essenzialmente tre e quanto le relazioni tra di essi siano estremamente semplici e intuitive.

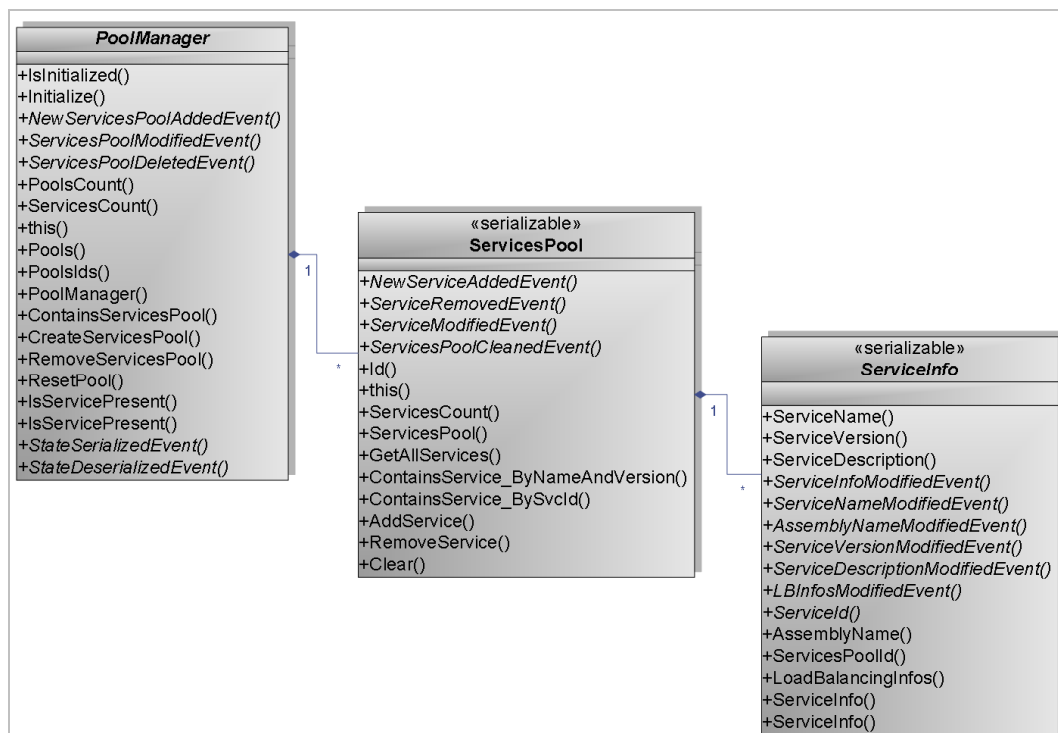


Figura 4-17 - Schema di funzionamento del namespace "ADES::ServicesCacheManagement"

In Figura 4-17 sono presentate le relazioni di composizione che intercorrono tra le principali classi del namespace "ADES::ServicesCacheManagement".

Si tratta di sole relazioni di composizione che vorrebbero stare a significare come il "PoolManager" contenga diverse pools, che a loro volta contengono oggetti "ServiceInfo", in una gerarchia che mette al livello più elevato di granularità gestionale il manager di un insieme di pools (il "PoolManager"), appena più sotto le pools stesse e infine, alla base della gerarchia, le strutture dati contenenti le informazioni sui servizi (gli oggetti "ServiceInfo", nelle diverse "incarnazioni" di "HostedServiceInfo" e "ExternalServiceInfo").

Questo excursus sulle strutture dati e i tipi che più intimamente contribuiscono alla gestione dei servizi nel sistema ADES trova un naturale sbocco nella descrizione dell'architettura che ADES implementa in risposta ai requisiti funzionali espressi nel paragrafo 4.3.

Tale architettura è riassunta nella Figura 4-18.

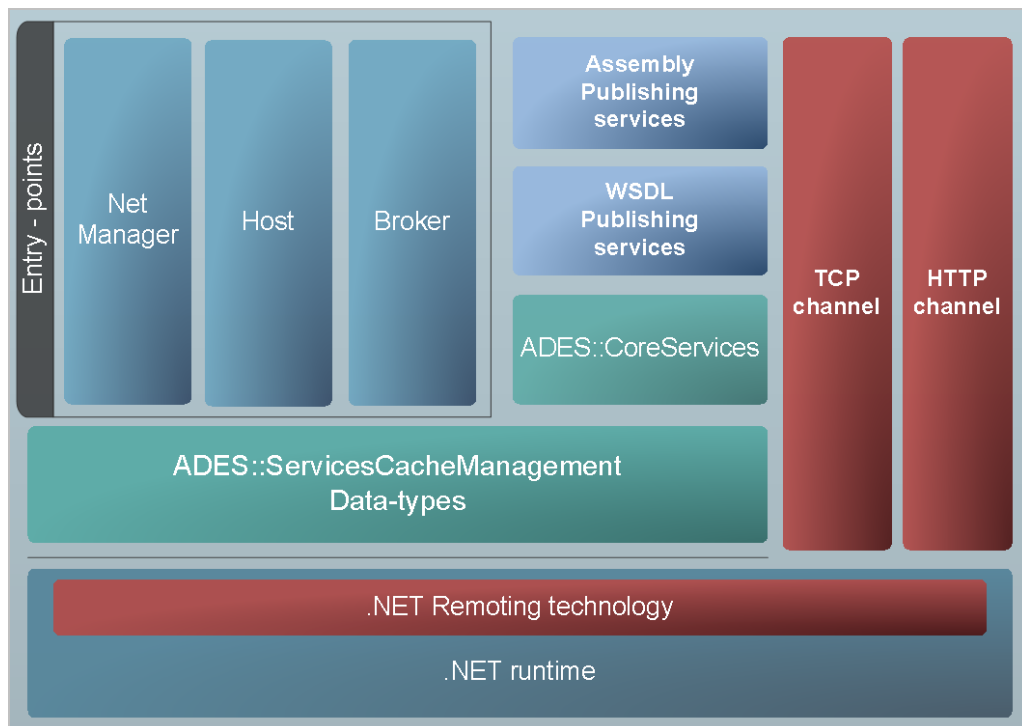


Figura 4-18 - Architettura del sistema ADES

Di tutti gli elementi che la compongono è riconoscibile il gruppo delle strutture dati del namespace “ADES::ServicesCacheManagement”, dei quali si è appena discusso.

Al di sopra di esso sta un gruppo di oggetti, raggruppati dall’etichetta “Entry-points”, dalla quale è possibile intuire come i tre oggetti in questione siano le unità funzionali con le quali l’utente finale del sistema si trova a interagire.

Essi sono pertanto l’astrazione di più alto livello che viene fornita a chi farà uso del codice ADES (non degli eventuali strumenti ad ancor più alto livello con i quali esso verrà distribuito).

Di essi verrà ora presentata un’analisi guidata dai soliti diagrammi UML, a partire dall’ oggetto “Host” che, come si può facilmente intuire, permette il caricamento e il delivering dei servizi (le funzionalità schematizzate in Figura 4-5, a pag. 54).

In Figura 4-19 è visibile come queste funzionalità siano state implementate in un unico tipo il quale assomma in se sia le funzionalità di hosting (metodo `HostService()`) che quelle di publishing dei servizi (metodo `PublishService()`), aggiungendovi anche alcune utilità di ricerca e “ricognizione” sui servizi (metodi `IsServicePublished()`).



Figura 4-19 - Oggetto "Host"

L’oggetto “Host” appartiene al namespace “ADES::Hosting” ed è il primo analizzato in questo lavoro ad implementare una pattern di sviluppo tra i più usati e conosciuti, il pattern “Singleton”.

Senza entrare troppo nel dettaglio si tratta di far sì che di un oggetto a runtime sia presente una sola istanza. In questo senso va letta la proprietà `Instance` che, appunto, restituisce l’unica istanza dell’oggetto “Host”.

In seguito si potrà apprezzare come sia semplice utilizzare tale oggetto e quali schemi ricorrenti di utilizzo siano stati pensati per esso e per gli altri oggetti del framework ADES (si veda il paragrafo 5.3, a pag. 97).

\*\*\*

Quello che viene presentato in Figura 4-20 è invece il cuore pulsante della tecnologia ADES, l’oggetto che incarna le funzionalità di service brokering.

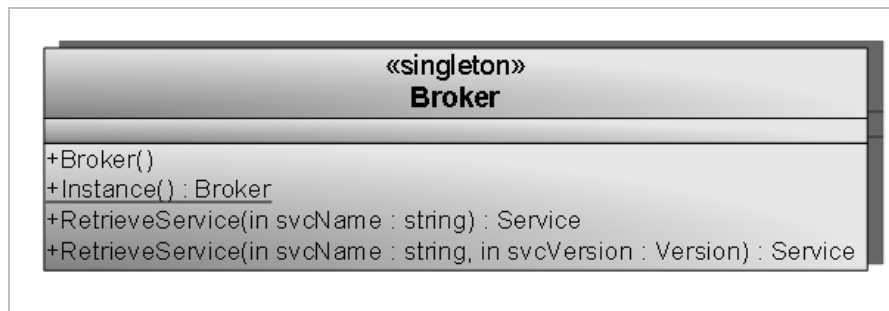


Figura 4-20 - Oggetto "Broker"

Ad un primo sguardo si potrebbe obiettare che esso sia eccessivamente semplice. Al solito l'apparenza trae in inganno: negli unici metodi `RetrieveService()` sono assommate le funzionalità di service location, provisioning, e load-balancing presentate dettagliatamente a livello teorico nel Capitolo 2 (a pag. 28). Il fatto che **due soli metodi** assolvano al compito per il quale il sistema ADES è nato la dice lunga su quanti sforzi siano stati fatti in direzione della semplicità.

C'è poi inoltre da aggiungere come dal diagramma manchino alcune funzionalità ereditate da un oggetto che si è già analizzato (precisamente a pag. 74), l'oggetto "PoolManager".

Si tratta di funzionalità per la gestione delle pools che sono indispensabili anche per l'oggetto "Host" e che sono state inserite nel "PoolManager" ai fini di fornire un'interfaccia di programmazione coerente a tutti quegli oggetti che si occupano della gestione di più pools.

Per chiarire quali siano queste funzionalità, oltre alla descrizione della Figura 4-16, si può far riferimento anche alla Figura 4-21, nella quale è rappresentata la gerarchia degli oggetti "PoolManager", "Host" e "Broker". Si può notare come gli ultimi due, derivando dal primo, ereditino appunto le funzionalità di gestione delle pools di servizi.

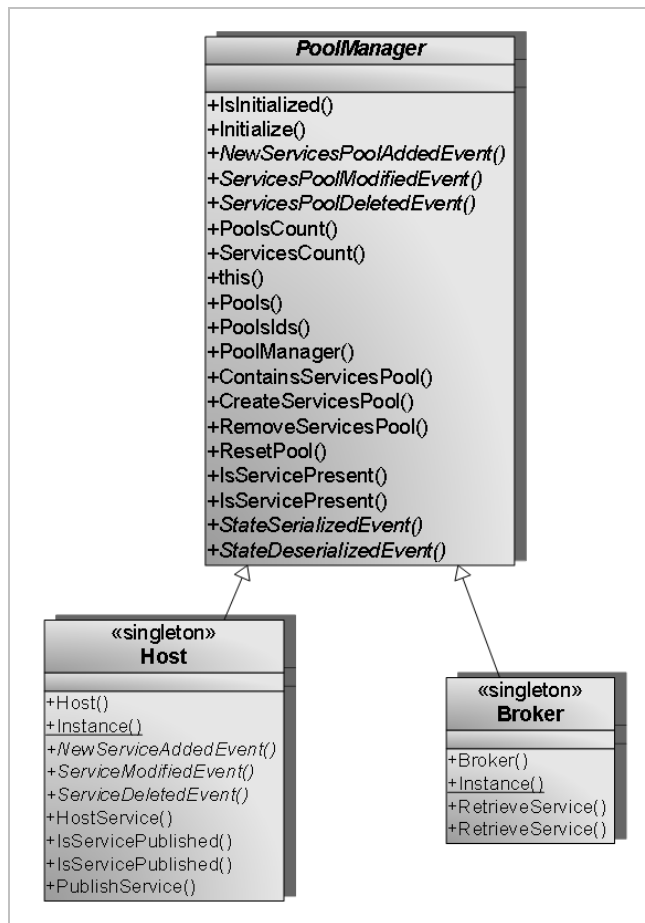


Figura 4-21 - Gerarchia degli oggetti di gestione delle pools

Si noti poi come anche l'oggetto "Broker" sia stato realizzato aderendo al pattern "Singleton".

Come già accennato non è compito di questa parte della trattazione l'analisi delle modalità di utilizzo di tali oggetti, in quanto si è pensato di dedicare a questo particolare aspetto un paragrafo apposito, il 5.3.

Pertanto si proseguirà nell'analisi del modello della Figura 4-18, nel quale è presente un'altra unità funzionale appartenente al gruppo degli entry-points, il "NetManager" (presentato in Figura 4-22).



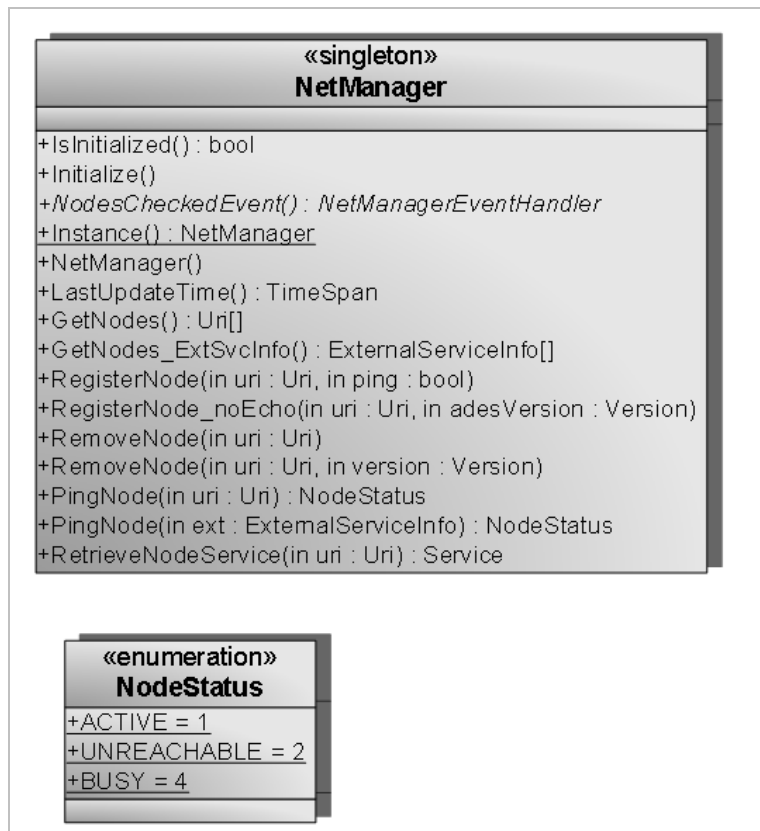


Figura 4-22 - Oggetto "NetManager" ed enumerazione "NodeStatus"

Abbastanza intuitivamente si può dire che esso si occupa di gestire i collegamenti e la topologia della rete (si ricordi a questo proposito l'analogia con i routers di una rete Internet che si era fatta all'inizio del paragrafo 2.2), scambiando con gli altri nodi della rete ADES una serie di informazioni che mirano a ricostruire una mappa completa di tutti gli host.

È questa un'attività "collaborativa" che permette a ogni nodo di avere sempre un elenco completo degli altri (sottoforma di una lista di indirizzi DNS o IP).

Tra le numerose possibilità offerte da questo oggetto vi è quella di effettuare un'operazione di ping dei nodi della rete. Una tale operazione è effettuabile anche nei confronti dei singoli servizi e ha come possibili risultati i valori dell'enumerazione "NodeStatus" (sempre in Figura 4-22).

Tali valori sono:

- `NodeStatus.ACTIVE`: segnala che un particolare servizio remoto è attivo e disposto ad accettare richieste e connessioni
- `NodeStatus.BUSY`: segnala che un particolare servizio remoto è sì attivo ma non disposto ad accettare connessioni, perché ha raggiunto un limite massimo (di solito stabilito durante lo sviluppo del servizio) di hosts ad esso collegati
- `NodeStatus.UNREACHABLE`: segnala invece che un determinato servizio remoto risulta irraggiungibile (“offline”)

Questo comportamento è comune a due tipi diversi di “pinging”: il ping di un servizio e il ping di un nodo. Ma mentre nel primo caso i risultati possibili sono tre (quelli dell’elenco precedente), nel secondo avremo che un nodo può essere solo in due stati distinti, attivo o irraggiungibile (rispettivamente `NodeStatus.ACTIVE` e `NodeStatus.UNREACHABLE`); questo perché si suppone che un nodo debba essere sempre in grado di reagire alle richieste degli altri, ai fini del continuo e costante “auto-aggiornamento” della rete ADES.

In effetti a ben guardare la Figura 4-10 (a pag. 66) si nota come il tipo restituito dall’invocazione del metodo `Ping()` su un servizio non è un’enumerazione “NodeStatus” ma bensì un booleano (quindi `true` o `false`).

Questo comportamento è desiderabile per minimizzare il traffico delle informazioni trasferite e comporta che a monte della chiamata al metodo `Ping()` si svolga un’operazione di “interpretazione” del risultato della sua esecuzione.

Interpretazione che convenzionalmente vuole associato a un risultato `true` un valore `NodeStatus.ACTIVE` e a un risultato `false` il valore `NodeStatus.BUSY`. Un valore `NodeStatus.UNREACHABLE` dovrebbe invece scaturire dal fallimento della chiamata remota al metodo, che testimonierebbe l’irraggiungibilità del servizio.

Altra nota non di poco conto riguarda il comportamento di default dei servizi ADES; infatti, qualora non specificato diversamente all'atto dell'implementazione della classe "ADES::Service", un'operazione di ping su un servizio potrà avere solo due esiti: o verrà restituito `true` o la chiamata remota non riuscirà affatto.

Utilizzando ancora una volta la Figura 4-18 (a pag. 77) possiamo individuare alcuni punti che questa trattazione del sistema ADES non ha ancora coperto. Si tratta innanzitutto dell'unità funzionale "ADES::CoreServices", che rappresenta alcuni servizi base che non si poteva prescindere dal codificare internamente al framework ADES.

Si tratta inoltre di oggetti le cui funzionalità sono fondamentali sia per la gestione della rete dei nodi ADES, sia per quella della pubblicazione dei servizi (che pure non è stata, per ora, analizzata nel dettaglio).

In Figura 4-23 è presente un esempio notevole di implementazione della classe "ADES::Service"; si tratta quindi di un primo servizio remoto che è possibile ottenere sfruttando le funzionalità di creazione del framework ADES.

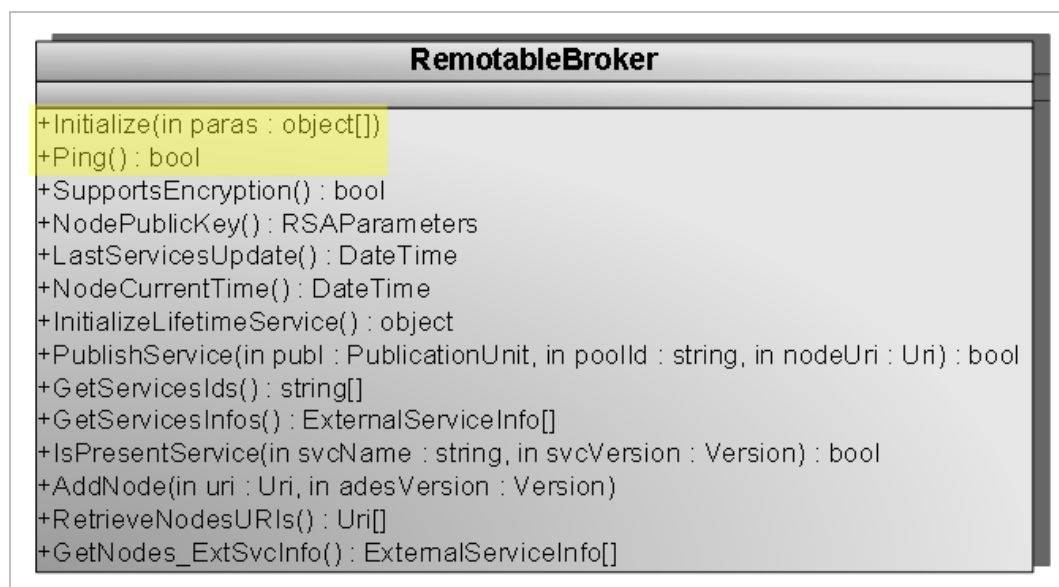


Figura 4-23 - Servizio "RemotableBroker"

Non solo l'oggetto in questione ha una notevole valenza esemplificativa (si notino a questo proposito i metodi di "ADES::Service" che sono stati implementati), ma si tratta di un servizio cruciale ai fini del funzionamento dell'intero sistema.

Si tratta infatti del servizio al quale ADES fa riferimento quando cerca di instaurare una comunicazione con un nodo remoto. Pertanto in esso sono presenti numerosi metodi atti sia ad operare cambiamenti che a ricavare informazioni su un nodo.

Per questo motivo tali metodi e tali proprietà meritano una trattazione approfondita, nonostante i meccanismi dei quali si parlerà non sono visibili all'utente finale del framework.

I metodi derivati direttamente dal prototipo di servizio "ADES::Service" sono:

- `Initialize()`: si tratta di un metodo al quale è stata fornita un'implementazione di base, in quanto il servizio "RemotableBroker" non necessita di particolari operazioni preliminari al suo utilizzo.
- `Ping()`: di questo metodo si è già parlato, confrontandolo con l'enumerazione "NodeStatus".

È inoltre presente un metodo derivato da "MarshalByRefObject":

- `InitializeLifetimeService`: l'implementazione che di esso si è fornita consente al servizio "RemotableBroker" di "rimanere in vita" in attesa di connessioni per tutta la durata dell'esecuzione del sistema ADES.

Il metodo deputato alla pubblicazione dei servizi è invece:

- `PublishService()`

I metodi che restituiscono informazioni sui servizi sono invece:

- `LastServicesUpdate`: proprietà che permette di conoscere l'intervallo temporale (calcolato in ore-minuti-secondi-millisecondi) nel quale sono state apportate modifiche ai servizi ospitati dal nodo.

- `GetServicesIds()`: che permette di conoscere tutti gli identificativi dei servizi ospitati dal nodo ADES.
- `GetServicesInfos()`: che permette di conoscere informazioni dettagliate sui servizi ospitati da un particolare nodo.
- `IsPresentService()`: che permette di conoscere se un particolare servizio è presente presso un determinato nodo.

Alcuni metodi sono poi dedicati all'aggiornamento della rete dei nodi ADES:

- `AddNode()`: che permette di aggiornare la lista dei nodi conosciuti da un particolare host ADES con un nuovo nodo.
- `RetrieveNodesURIs()`: un metodo che restituisce l'elenco completo degli indirizzi DNS o IP dei nodi "conosciuti" da un particolare host.
- `GetNodes_ExtSvcInfo()`: come sopra, ma vengono ottenute informazioni più dettagliate sui nodi, sottoforma di oggetti "ExternalServiceInfo" (si ricordi che per ADES un altro nodo non è altro che un servizio remoto).

Un altro insieme di metodi è poi deputato a gestire la sicurezza di base (si veda il paragrafo 5.4) del nodo:

- `SupportsEncryption`: è una proprietà che informa se il nodo supporti o meno la crittografia delle informazioni di pubblicazione (funzionalità questa supportata solo su sistemi XP Professional o superiori)
- `NodePublicKey`: una proprietà contenente la chiave pubblica per la crittografia dei dati di pubblicazione.

Le funzionalità offerte dal "RemotableBroker" sono gli unici punti di contatto che i nodi della rete ADES hanno tra loro e questo ha determinato un certo compromesso quando si è stabilito di inserire funzionalità non sempre omogenee tra loro in un'unica interfaccia.

Dell'architettura ADES non restano che da discutere le funzionalità di pubblicazione, che consentono la diffusione di un servizio in tutti i nodi della rete. Esse si basano su due tipi diversi di pubblicazione: pubblicazione attraverso assembly .NET e pubblicazione WSDL (la stessa dei Web Services).

Il perché si sia scelto di offrire un meccanismo doppio è semplice: si vuole dare la possibilità al sistema di funzionare sia in un contesto nel quale tutti i nodi della rete sono host "ADES-enabled" (in questo caso la pubblicazione è affidata agli assembly .NET), sia in uno in cui ADES deve poter offrire una certa interoperabilità con altre tecnologie, soprattutto quella dei Web Services.

Inoltre una pubblicazione WSDL dei servizi, oltre all'interoperabilità, consente di aggirare alcune politiche di sicurezza dei firewall (si veda il paragrafo 3.3).

Tale meccanismo è affidato essenzialmente a due strutture dati, riportate in Figura 4-24.

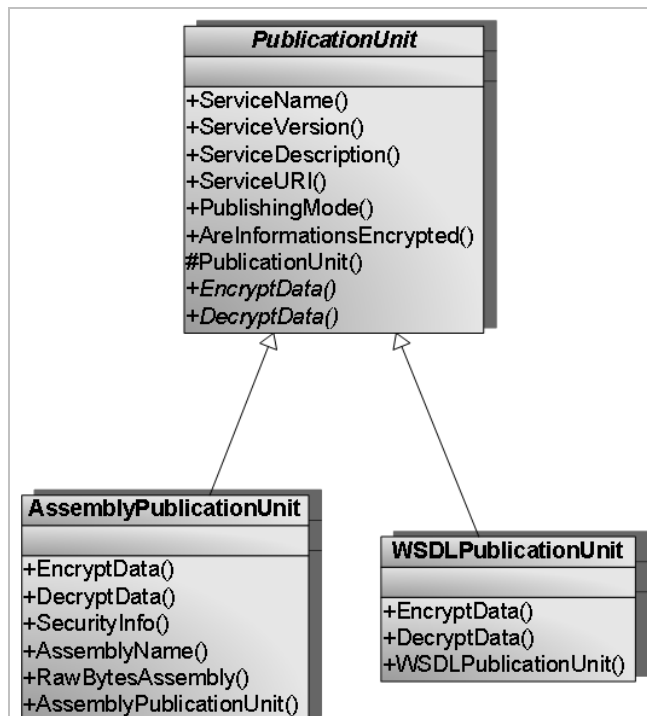


Figura 4-24 - Gerarchia delle classi di pubblicazione dei servizi



Essa corrisponde ad un particolare formato:

Si tratta in effetti di qualcosa di simile a un normale indirizzo web. In effetti un nodo ADES è visibile a una rete (attraverso il protocollo HTTP) come se fosse una qualsiasi delle risorse Internet alla quale si è abituati ad accedere da un semplice browser HTML.

Ma per il resto non si tratta di niente di particolarmente “esotico”.

Il risultato è un documento XML; precisamente si tratta di un documento WSDL. Questo garantisce una certa interoperabilità tra ADES e un eventuale sistema esterno che si interfacci ad esso via Web Services.



In aggiunta a queste considerazioni è utile schematizzare accuratamente il formato standard della stringa di connessione ai servizi esposti da ADES attraverso il protocollo HTTP e la pubblicazione di tipo WSDL.

Questo perché tale formato è utilizzato anche all'interno del framework per l'accesso remoto ai servizi (in particolare viene usato all'interno del broker per il reperimento di un servizio una volta che se ne sia accertata la posizione).

In Figura 4-27 è illustrato questo formato.

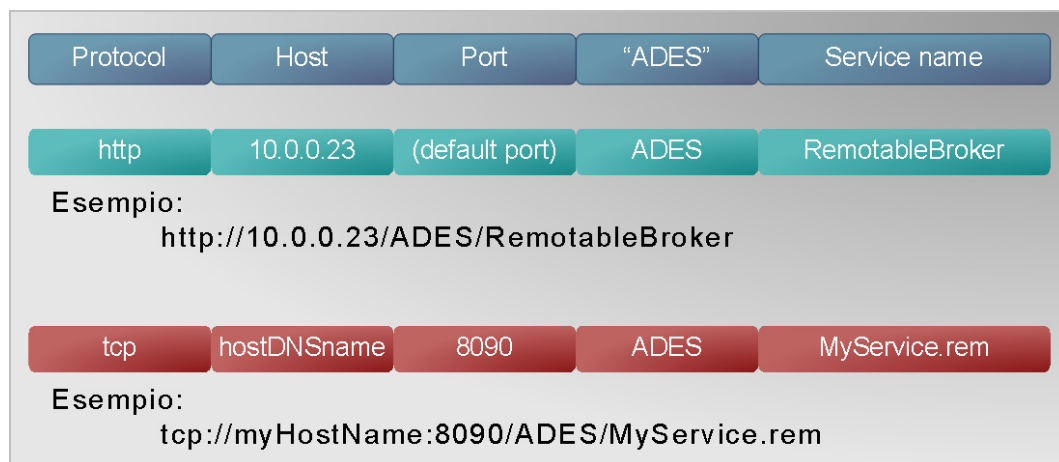


Figura 4-27 - Formato delle stringhe di connessione ai servizi

In ultima analisi, del modello che si è deciso di implementare non restano che da discutere alcune unità funzionali molto utili, ma che trovano posto in diversi punti nevralgici del sistema.

La loro trattazione risulterà quindi meno omogenea della precedente, dedicata invece a insiemi ben distinti di unità funzionali.

Si inizierà quindi parlando della classe "ADES::Configuration", una sorta di repository per tutte le variabili di configurazione del sistema.

Essa viene presentata in Figura 4-28.

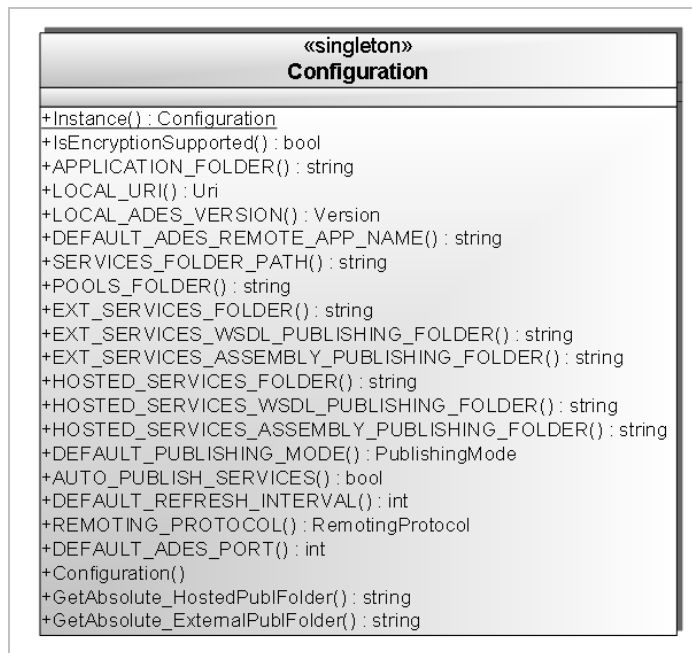


Figura 4-28 - Tipo "ADES::Configuration"

Un altro oggetto contribuisce a rendere consistente e semplice l'accesso, non solo interno, ad alcune funzionalità di ADES. Si tratta dell'oggetto "ADES::Utils", presentato in Figura 4-29 e presenta alcuni metodi di utilità per il framework e per i suoi fruitori.

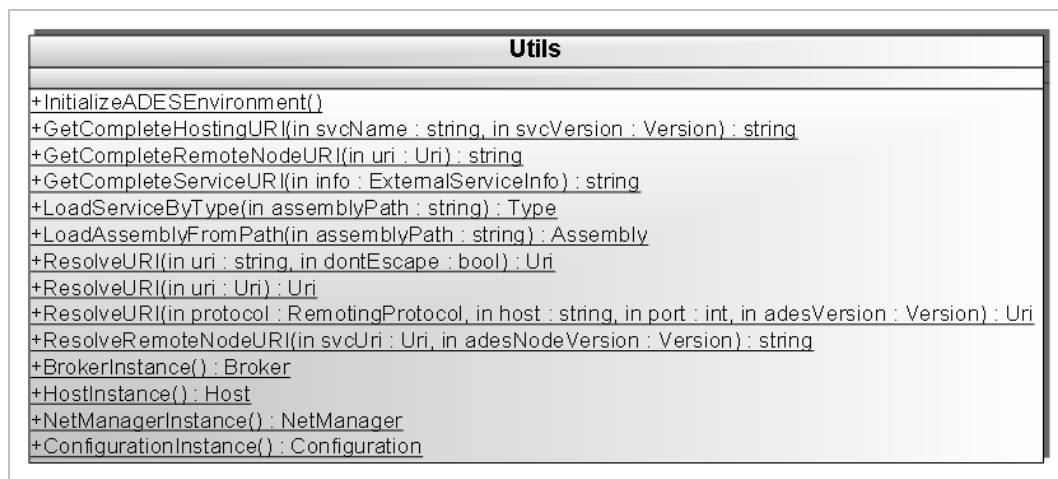


Figura 4-29 - Tipo "ADES::Utils"

Infine verrà brevemente presentato il ruolo degli oggetti in Figura 4-30.

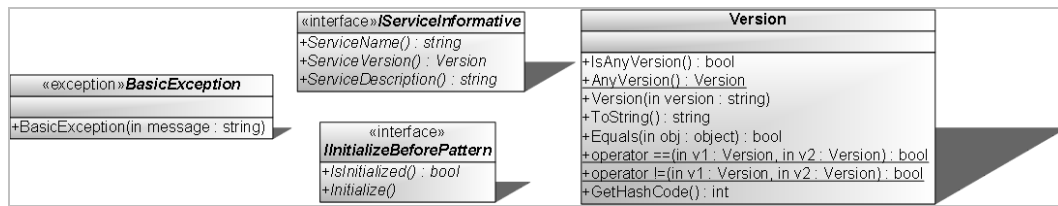


Figura 4-30 - Vari oggetti di utilità del namespace "ADES"

Si tratta di alcuni tipi molto utili, per alcuni dei quali verrà fornita una spiegazione dettagliata nei paragrafi successivi:

- L'eccezione "ADES::BasicException", della quale si è già parlato.
- L'interfaccia "ADES::IServiceInformative"; anche di essa si è già parlato.
- L'interfaccia "ADES::IInitializeBeforePattern", utilizzata in tutti i contesti in cui si voglia fornire a un oggetto un accesso vincolato a un'inizializzazione preliminare; di essa si parlerà più approfonditamente nel paragrafo 5.3.
- L'oggetto "ADES::Version", un tipo utilizzato per il versioning dei servizi (e per il futuro versioning di diverse distribuzioni ADES); anche per esso verrà detto qualcosa di più nel paragrafo 5.3.

## **5. Aspetti avanzati di progettazione e sviluppo del sistema ADES**

In questo capitolo verranno discussi alcuni punti particolarmente importanti del processo di sviluppo del framework/environment ADES.

A partire dalla descrizione del modello di deployment fino a quella riguardante la sicurezza, in questo capitolo verranno illustrati alcuni principi avanzati in base ai quali ADES è stato realizzato.

### **5.1. Distribuzione “fisica” delle funzionalità del sistema ADES**

Finora si è parlato poco dell’aspetto che ADES presenta in termini di eseguibili, librerie, cartelle e files di configurazione. Questo particolare punto della trattazione fa parte del modello di deployment del sistema, ma si è deciso di rinunciare ai formalismi dell’UML per la sua trattazione, preferendo la descrizione di tale modello in termini di files e cartelle.

Questo anche grazie alla semplicità che ADES da questo punto di vista offre.

Nella Figura 5-1 è rappresentata una possibile dislocazione di files e cartelle di una distribuzione ADES.

Si tratta di una delle possibili varianti, in quanto è possibile utilizzare diverse disposizioni a seconda delle esigenze.

Sono stati evidenziati alcuni dettagli di una certa importanza:

- in giallo (alla destra della figura) la cartella che ospita tutte le librerie, “bin”, direttamente “immersa” tra i files che costituiscono l’applicazione web “Limbo”, utilizzata per il controllo remoto di un nodo ADES.
- in verde la cartella “Pools” che contiene gli stati serializzati delle pools.

- in violetto il sistema di cartelle che, di default, ospita i servizi e le loro informazioni di pubblicazione.

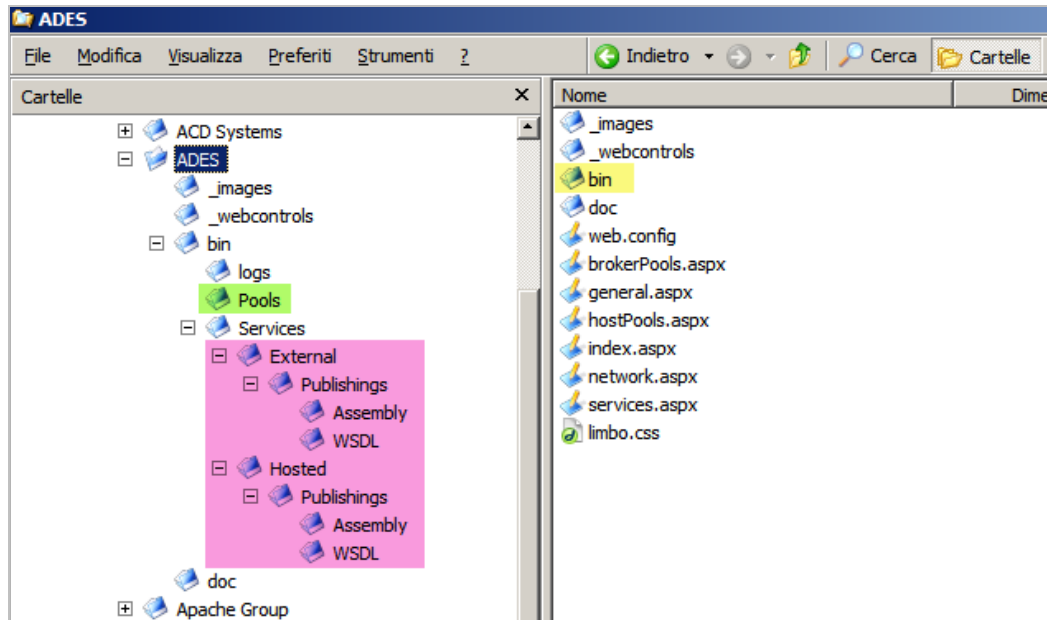


Figura 5-1 - Distribuzione fisica del sistema ADES – Cartelle

L'organizzazione delle cartelle che ospitano i servizi vede una divisione tra servizi “external” e servizi “hosted”. All'interno di queste due suddivisioni trova poi posto un'ulteriore distinzione, quella tra informazioni di pubblicazione in forma di assembly o in forma di documenti WSDL.

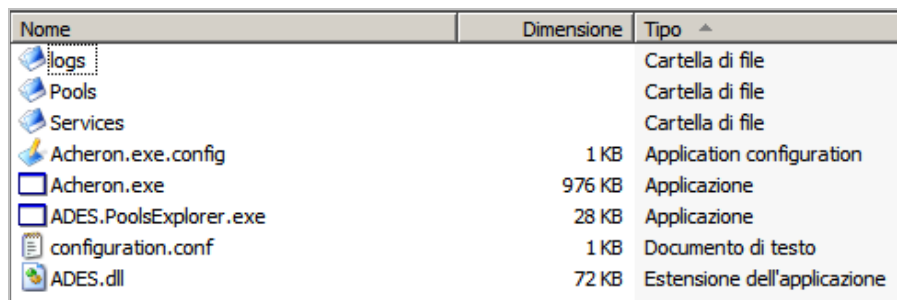
Si può già notare come l'albero delle cartelle sia piuttosto semplice e gran parte delle sue ramificazioni obbedisca all'architettura di ADES che si è già vista.

Nella Figura 5-2 viene invece presentato il dettaglio della cartella “bin”, che ospita la libreria “ADES.dll”.

**ADES.dll è la libreria che contiene il framework ADES.**

Per poter offrire un ambiente di esecuzione essa ha però bisogno di un applicativo che ne esponga a runtime le sue funzionalità.

In questa prima release di ADES è stata realizzata un'applicazione, "Acheron" (il file "Acheron.exe" in Figura 5-2), che offrisse sia l'ambiente runtime che un'interfaccia ad alto livello per ADES.



Nome	Dimensione	Tipo
logs		Cartella di file
Pools		Cartella di file
Services		Cartella di file
Acheron.exe.config	1 KB	Application configuration
Acheron.exe	976 KB	Applicazione
ADES.PoolsExplorer.exe	28 KB	Applicazione
configuration.conf	1 KB	Documento di testo
ADES.dll	72 KB	Estensione dell'applicazione

Figura 5-2 - Distribuzione fisica del sistema ADES - Applicazioni e librerie

Come si può notare il numero di componenti di deployment finora è piuttosto esiguo, considerato anche che quanto visibile in Figura 5-2 comprende anche un programma di utilità "PoolsExplorer" (l'eseguibile "ADES.PoolsExplorer.exe" in figura), il file di configurazione per il primo avvio di Acheron ("Acheron.exe.config"), quello per ADES ("configuration.conf") e la cartella all'interno della quale vengono conservati i files di log del sistema ("logs").

## 5.2. Publishing dei servizi tramite interfacce

Un altro aspetto del quale ancora poco si è detto riguarda il meccanismo di pubblicazione dei servizi attraverso la rete ADES.

Nel caso della pubblicazione WSDL non sussiste alcun problema: si imposta questa funzionalità sull'ADES environment e si può ottenere il documento WSDL di descrizione del servizio puntando all'indirizzo del servizio con il parametro "?WSDL" "appended" in coda alla stringa di connessione (si vedano per maggiori dettagli la Figura 4-25, la Figura 4-26 e la Figura 4-27).

Un meccanismo più elaborato deve essere invece studiato quando la pubblicazione avvenga sottoforma di assembly.

Piuttosto intuitivamente la diffusione dell'assembly contenente il servizio è assolutamente da evitare, per almeno due motivi: in primo luogo non si dovrebbe diffondere sulla rete un componente di business così importante (esso infatti potrebbe venire intercettato, analizzato e modificato da utenti malintenzionati) e in secondo luogo si tratta di un carico eccessivo con il quale si rischierebbe di creare un sovraccarico della rete.

In effetti è preferibile affidare alla rete solo le informazioni necessarie all'invocazione del servizio, vale a dire **l'interfaccia** del servizio.

L'attenta valutazione del termine "interfaccia" porta con sé la soluzione al problema.

Si tratta di un meccanismo semplice: affidare alla rete un'interfaccia contenente solo le signatures dei metodi invocabili sul servizio e realizzare il servizio implementando tale interfaccia.

In questo modo non viene diffuso alcun codice, né sussiste il rischio di modifiche ai servizi apportate durante il transito sulla rete (in quanto il servizio vero e proprio non attraversa la rete).

Il tutto si risolve essenzialmente nell'effettuare due compilazioni differenti: una della sola interfaccia, il cui risultato verrà posto nella cartella di pubblicazione dei servizi ("`<ADES_app_folder>/bin/Services/Hosted/Publishings/Assembly`", si faccia riferimento anche alla Figura 5-1); un'altra coinvolgente l'interfaccia e il servizio che la implementi il cui risultato va ad essere inserito nella cartella principale dei servizi "Hosted" ("`<ADES_app_folder>/bin/Services/Hosted`", sempre in riferimento alla Figura 5-1).

Diverso codice viene coinvolto nei due diversi processi di compilazione; il primo (quello della sola interfaccia) coinvolge il seguente listato<sup>i</sup>:

```
// ...  
public interface IDummyService  
{  
    void MyDummyMethod();  
    string AnotherDummyMethod(int para1);  
}
```

Il risultato va a costituire l'assembly "ADES.DummyService.dll", il quale deve essere inserito nella cartella "Services/Hosted/Publishings/Assembly".

La seconda compilazione coinvolgerà invece il seguente codice:

```
// ...  
public interface IDummyService  
{  
    void MyDummyMethod();  
    string AnotherDummyMethod(int para1);  
}  
  
public class DummyService: ADES.Service, IDummyService  
{  
    public void MyDummyMethod()  
    {  
        // ...  
    }  
  
    public string AnotherDummyMethod(int para1)  
    {  
        return para1.ToString();  
    }  
}
```



Anche in questo caso il risultato sarà l'assembly "ADES.DummyService.dll" (il cui contenuto sarà ovviamente diverso dal precedente), ma il deployment verrà effettuato nella cartella ""Services/Hosted".

Il procedimento qui descritto fa parte di un insieme di "patterns and practices" (modelli e pratiche di programmazione e progettazione ben noti e dimostratisi efficienti) riconosciuto e consolidato, pertanto il suo utilizzo conferisce una certa solidità al meccanismo di assembly publishing del sistema ADES.

In seguito, precisamente nel paragrafo 5.3, si vedrà come operare sul versante che "riceve" la pubblicazione del servizio per la fruizione dell'interfaccia trasmessa.

### **5.3. Metodi, pattern di utilizzo e schemi ricorrenti per l'accesso alle funzionalità del sistema ADES**

Il sistema ADES è stato pensato per un utilizzo il più possibile intuitivo e veloce, spesso basato sull'utilizzo di schemi logici ricorrenti. È poi stata ovviamente prevista la presenza di una buona documentazione (si veda la Figura 5-3) e di un buon manuale utente che spieghino le logiche di accesso alle funzionalità offerte.

Questa sezione vorrebbe essere una sorta di manuale utente in versione ridotta, utile a illustrare non solo "come ADES funzioni", ma anche a spiegare il "perché funzioni" così.

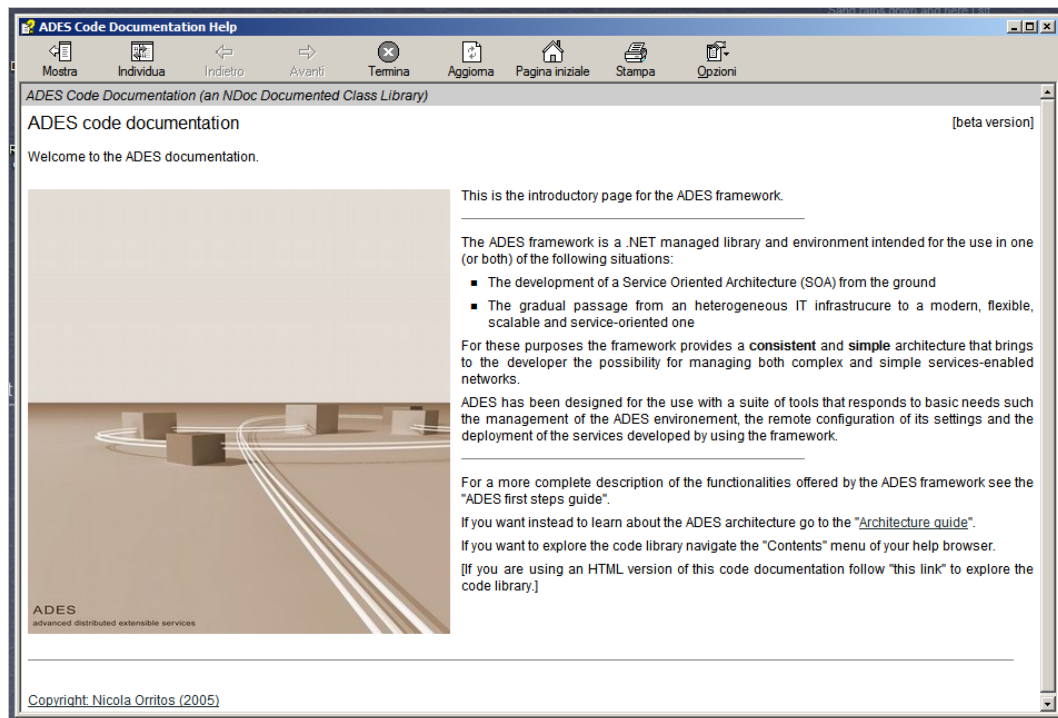


Figura 5-3 - Documentazione a corredo del framework ADES

Il modo forse migliore di condurre questo tipo di analisi è probabilmente presentare il normale flusso di lavoro che si affronta quando si lavora con ADES. In particolare è utile iniziare dedicandosi alle procedure da intraprendere al fine di effettuare il caricamento di un servizio.

Il servizio non è un'entità che possa essere utilizzata direttamente se non dal sistema ADES: lo sviluppatore non deve occuparsi di altro che di fornire ad ADES le informazioni necessarie a identificare e caricare il servizio; in pratica il servizio viene manipolato attraverso un “proxy” che lo rappresenta. Inoltre, sempre dal punto di vista dello sviluppatore, un servizio non viene preso in considerazione singolarmente ma si fa sempre riferimento alla pool che lo contiene.

A caricare e pubblicare i servizi è il tipo “ADES::Hosting::Host”, al quale deve essere fornito un oggetto “HostedServiceInfo” che descriva il servizio.

Questo procedimento è descritto dal seguente listato:

```
using ADES.Hosting;
using ADES.ServicesCacheManagement;

// [...]

// L'unico parametro del costruttore 'HostedServiceInfo'
// rappresenta il nome dell'assembly contenente
// il servizio.
HostedServiceInfo info = new HostedServiceInfo(
    "ADES.DummyService");

    if(!Host.Instance.IsInitialized)
        Host.Instance.Initialize();

    if(!Host.Instance.ContainsServicesPool("newPool"))
        Host.Instance.CreateServicesPool("newPool");

    try
    {
        // Il terzo parametro, di tipo booleano, comanda
        // la pubblicazione (se 'true') o meno (se 'false')
        // del servizio
        Host.Instance.HostService(svcInfo, "newPool", true);
    }
    catch(DuplicateServiceException)
    {
        // Servizio già caricato
    }
```

Questo codice porta con se alcuni quesiti; in primo luogo ci si potrebbe domandare perché debba essere inizializzato prima dell'utilizzo l'oggetto "Host".

Ebbene esso deve essere inizializzato perché il suo stato deve poter essere recuperato dal disco, insieme a tutte le informazioni sui servizi e sulle pools legati a una precedente esecuzione dell'ambiente.

Il comportamento di inizializzazione è fornito dal contratto specificato nell'interfaccia "ADES::IInitializeBeforePattern", già incontrata in Figura 4-30 (a pag. 91) ed è ricorrente anche in altre unità funzionali del framework, quali il broker, il "NetManager" e l'oggetto "ADES::Configuration".

L'implementazione del contratto fornita da tali oggetti prevede il sollevamento di un'eccezione nel caso si faccia uso di tali oggetti senza che essi siano stati precedentemente inizializzati.

L'oggetto "ADES::Utils" espone comunque una semplice "scorciatoia" che si incarica di inizializzare automaticamente e rendere disponibile l'istanza di questi oggetti; si tratta delle proprietà "Utils.xxxxxInstance", dove "xxxxx" sta per il tipo del quale si desidera ottenere l'istanza.

Si parlerà quindi di diverse proprietà, illustrate nel seguente elenco:

- una proprietà `Utils.HostInstance`
- una proprietà `Utils.BrokerInstance`
- una proprietà `Utils.NetManagerInstance`
- una proprietà `Utils.ConfigurationInstance`

L'utilizzo di tali proprietà libera il programmatore dai compiti di verifica e di inizializzazione di questi oggetti.

Tramite l'utilizzo di una simile strategia le righe precedenti diventerebbero:

```
// [...]
Host hostInstance = Utils.HostInstance;

    if(!hostInstance.ContainsServicesPool("newPool"))

// [...]
```

L'esecuzione dei due listati precedenti comporta l'immediato caricamento dell'assembly "ADES.DummyService.dll" e la pubblicazione del servizio in esso contenuto.

Questo sempre che siano stati correttamente collocati nelle opportune cartelle l'assembly contenente il servizio e quello contenente l'interfaccia (si veda il paragrafo 5.2).

In poche righe si è così ottenuto il caricamento di un servizio, lasciando ad ADES il compito di diffondere nella rete un assembly di pubblicazione dello stesso.

Per quanto riguarda la creazione del servizio sono stati utilizzati il codice e gli assemblies risultanti dal procedimento analizzato nel paragrafo 5.2.

Ora, supposto che sia avvenuta la pubblicazione WSDL del servizio, che l'host nel quale sia stato caricato abbia nome DNS "myHost" e che la porta utilizzata sia la "8090", la stringa di connessione al servizio sarebbe:

```
http://myHost:8090/ADES/Dummy.rem
```

Questa stringa è la stessa utilizzata dal meccanismo interno di brokering del sistema e permette all'oggetto "ADES::Brokering::Broker" l'accesso e il delivering del servizio "ADES.DummyService".

Il meccanismo con il quale invece il servizio viene reso disponibile dal broker verso l'utente non richiede l'elaborazione di nessuna particolare stringa di connessione e si riduce al codice seguente:

```
using ADES;
using ADES.Brokering;

// ...

Broker broker = Utils.BrokerInstance;
ADES.Service dummyService;
```

```
dummyService = broker.RetrieveService(  
    "ADES.DummyService");  
  
// oppure:  
Version ver = new ADES.Version("1.0.534");  
dummyService = broker.RetrieveService(  
    "ADES.DummyService",  
    ver);  
  
// oppure (e in questo caso il codice è perfettamente  
// equivalente alla prima chiamata effettuata al  
// metodo 'broker.RetrieveService(string)'):  
dummyService = broker.RetrieveService(  
    "ADES.DummyService",  
    ADES.Version.AnyVersion);  
  
ADES.DummyService.IDummyService resultingService =  
    dummyService as ADES.DummyService.IDummyService;  
  
// ...
```

In appena tre righe sono condensate:

- **Funzionalità di service location**, eseguita localmente sulla lista dei servizi e dei relativi nodi di appartenenza che il broker possiede e che aggiorna a intervalli regolari di tempo.
- **Funzionalità di service provisioning**, poiché il servizio restituito viene fornito in modo trasparente al chiamante (per il quale il servizio sarà una variabile locale rispetto al suo contesto di esecuzione e rispetto alla macchina sulla quale viene eseguito il codice precedente).

Si noti poi come il broker sia in grado di condurre una ricerca su diverse versioni dei servizi, confrontandole con quelle fornite dall'utente.

Qualora poi si desiderasse una qualsiasi versione di un servizio ADES fornisce la prima disponibile.

Il meccanismo di versioning è poi particolarmente interessante, per la presenza dell'oggetto "ADES::Version".

Esso offre un meccanismo semplice ed efficace, basato sull'utilizzo di sole stringhe incapsulate in oggetti "Version", anziché di numeri di major, minor revision e di build (sui quali si basa invece l'oggetto "System::Version" messo a disposizione dal .NET Framework).

Questo ha poi significato il poter utilizzare caratteri jolly ("wildcards") che effettuassero confronti tra versioni in maniera piuttosto semplice.

Pertanto le seguenti righe di codice sono tutte egualmente riconducibili alla creazione di un oggetto "ADES::Version" "legale" e perfettamente utilizzabile:

```
ADES.Version version1 = new ADES.Version("1.0.1682");
ADES.Version version2 = new ADES.Version("alpha");
ADES.Version version3 = new ADES.Version("Beta");

// Versioni "wild-carded"; entrambe possono essere
// utilizzate in un confronto con altre versioni
// e il risultato sarà sempre che le due versioni
// confrontate saranno uguali
ADES.Version version4 = new ADES.Version("*");
ADES.Version version5 = ADES.Version.AnyVersion;
```

Le versioni che utilizzano caratteri jolly (ad ora ADES prevede un solo carattere jolly, "\*") sono molto utili in diversi contesti e gli operatori di confronto più comuni (`==`, `!=`, `ADES.Version.IsEqual()`) sono già predisposti per il loro utilizzo (è stato cioè effettuato il loro overloading).

Un ulteriore aspetto molto importante su quale si desidera focalizzare l'attenzione riguarda come sia possibile all'utente specificare le informazioni di load-balancing del servizio, all'atto del caricamento.

La struttura dati "ADES::Brokering::LoadBalancing::LBServiceInfo" preposta a questo compito è stata già presentata nella Figura 4-13 (a pag. 71).

Il codice seguente ne descrive l'utilizzo:

```
HostedServiceInfo info = new HostedServiceInfo(
    "myService");

// Il primo parametro del costruttore
// è l'Id del servizio che si sta caricando,
// il secondo rappresenta il massimo numero di connessioni
// che questo servizio è disposto ad accettare,
// il terzo parametro rappresenta il numero di connessioni
// correntemente attive per il servizio;
// l'ultimo parametro infine descrive il tempo
// medio di risposta del servizio alle richieste remote
LBServiceInfo lbInfos = new LBServiceInfo(
    info.ServiceId,
    200,
    10,
    10);

info.LoadBalancingInfos = lbInfos;
```

Anche questa struttura dati è di immediato utilizzo e può essere aggiornata ogniqualevolta cambi il carico di lavoro del servizio (o il carico che il servizio può supportare).

Come già detto si tratta di un'operazione questa che richiede il coinvolgimento del servizio.



#### 5.4. Gestione della sicurezza nel sistema ADES

Visti i ridotti tempi di sviluppo alcune funzionalità molto interessanti non sono state implementate.

È questo il caso della gestione della sicurezza, alla quale in ADES è dedicata un intero sottosistema, “ADES::Security”.

Quello che finora è stato implementato è la trasmissione sicura delle informazioni di pubblicazione, che potrebbero essere intercettate durante la trasmissione.

Esse vengono crittografate con un algoritmo a chiave privata e la chiave viene a sua volta crittografata con uno a chiave pubblica.

Queste informazioni vengono poi trasmesse insieme fino a destinazione, dove verranno opportunamente de-crittografate.

In Figura 5-4 sono visibili le classi che si occupano di tali operazioni.

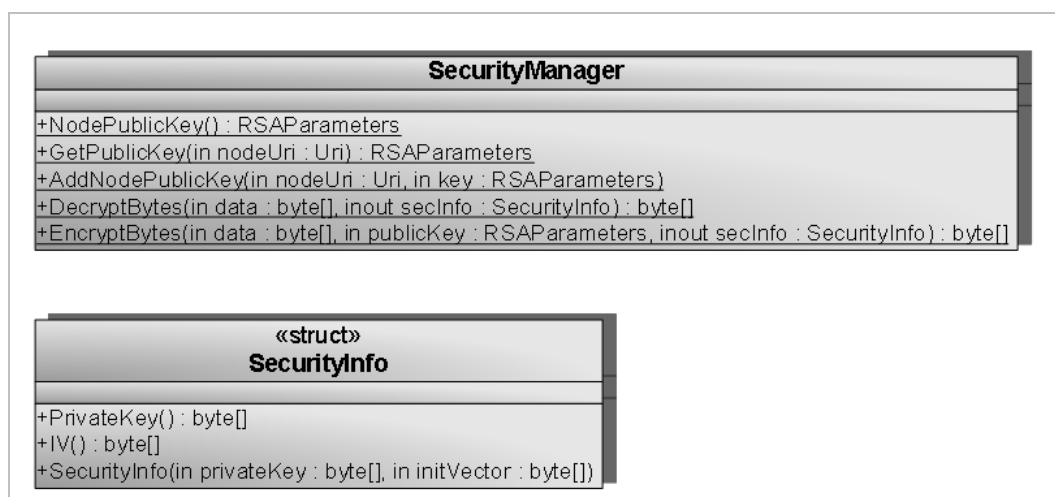


Figura 5-4 - Classi per la crittografia delle informazioni di pubblicazione

Questo non è comunque abbastanza; è infatti stato studiato un meccanismo per la gestione di politiche di sicurezza (“security policies”) più complesse, basato sui concetti di “identità” (identities), contesti di sicurezza (“security contexts”) e su quello già incontrato di pool di servizi.

L’idea alla base di questo meccanismo è illustrata nella Figura 5-5.

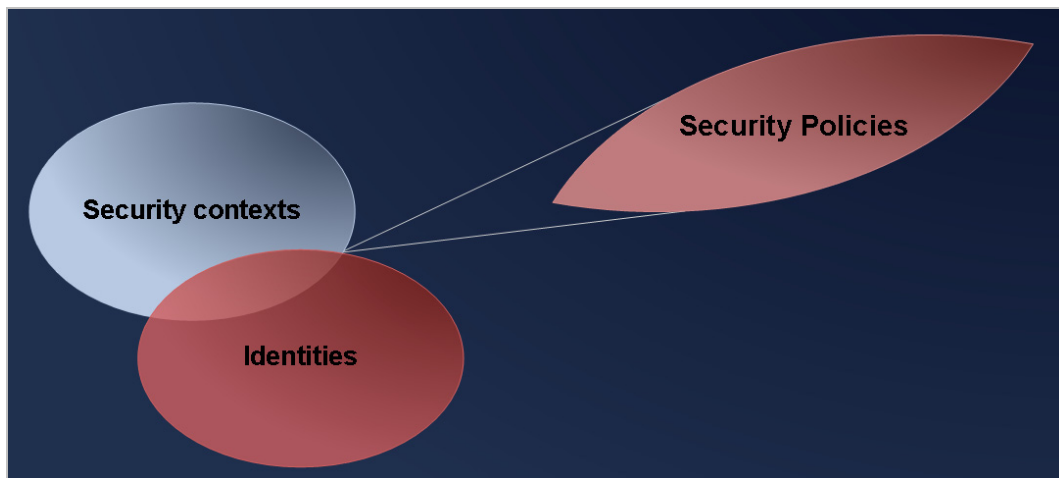


Figura 5-5 - Meccanismi di security-policing nel sistema ADES

Si nota come le possibili politiche di sicurezza in una rete ADES altro non siano che il risultato dell'intersezione di contesti di sicurezza e delle identities abilitate a operare in quei contesti.

Il significato del termine “operare” indica essenzialmente l'atto di esecuzione di un servizio.

Questo significa che i servizi esistono all'interno di un ben determinato contesto di sicurezza e che solo alcune identità (hosts o utenti del sistema ADES) possono accedere ad essi ed eseguirli.

Questo ragionamento ci porta direttamente all'organizzazione in pools dei servizi, che non a caso è stata studiata accuratamente in previsione di meccanismi che isolassero diverse categorie di servizi.

Questo significa che una pool di servizi “appartiene” a un (e uno solo) contesto di sicurezza, sul quale zero o più identities hanno diritti di esecuzione, come schematizzato in Figura 5-6.

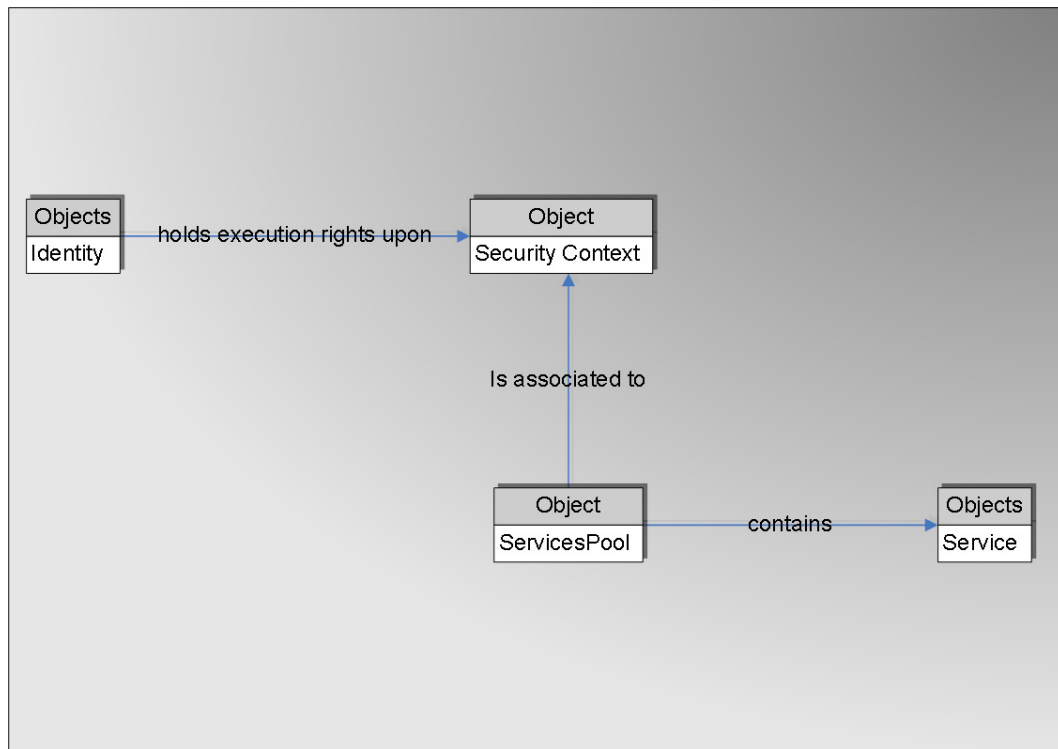


Figura 5-6 - Dettaglio dei meccanismi di security-policing del sistema ADES

Un problema è senz'altro la realizzazione di un simile sistema.

Una possibile soluzione sarebbe quella di utilizzare un servizio incorporato in ADES che abbia i diritti massimi di esecuzione e che mantenga aggiornata in tutta la rete una lista dei contesti di sicurezza e delle identità ad essi associati.

Gli assembly ADES potrebbero poi venire agevolmente “segnati” (con un meccanismo di “assembly signing” previsto dal .NET Framework) all’atto dell’installazione della rete, così come gli assembly dei servizi.

A questi ultimi sarebbe poi possibile applicare un attributo (un costrutto del .NET Framework utile per l’aggiunta di metadati agli assemblies) che specifichi le credenziali (in termini di identities) richieste per l’esecuzione degli stessi.

Si noti come la “digital signature” sugli assemblies sia necessaria affinché questi non vengano manomessi o “impersonati” da altri assemblies, costruiti ad hoc da malintenzionati.

## Note

---

- <sup>i</sup> Il linguaggio utilizzato per l'esposizione degli esempi che richiedano l'utilizzo di codice è il C# (in particolare, la versione utilizzata nel .NET Framework 1.1)

## **6. Tools grafici e interfacce utente a corredo del sistema ADES**

Il discorso sulla semplicità di utilizzo del sistema ADES, la sua “business-user orientation”, le piacevoli astrazioni e lo “spirito” stesso con il quale ADES è stato progettato e realizzato sono vani fino a che non si fornisce all’utente un meccanismo tale che la gestione dell’ambiente diventi semplice, intuitiva e accessibile.

Questo essenzialmente significa che all’utente **devono** essere forniti gli strumenti adeguati e tali da permettere il completo, efficace, efficiente e proficuo sfruttamento delle funzionalità offerte da ADES.

Appunto di tali strumenti si discute in questo capitolo, fornendone una rapida carrellata e dedicando un particolare approfondimento alla “vista” che essi offrono dell’intero progetto.

### **6.1. Una vista ad alto livello di astrazione del sistema ADES e delle applicazioni “Acheron” e “Limbo”**

Si è in precedenza discusso della dualità “framework/environment” del sistema ADES (precisamente nel paragrafo 4.4, a pag. 62) e pertanto in queste righe non si farà altro che mostrare come tale modello architetturale sia stato trasposto anche al più alto livello di astrazione del sistema, quello più a stretto contatto con l’utente (destinatario finale dei benefici o degli handicap derivanti dalle scelte architetture).

Finora, quando si è accennato a una possibile distribuzione del software non si è posto l’accento sulla vasta gamma di possibilità che ADES offre: in fin dei conti le funzionalità di brokering e di sviluppo, essendo codificate in un’unica libreria, garantiscono versatilità e portabilità estreme.

Ma, cosa ancora più importante, lasciano ampia libertà a chi volesse offrire un'interfaccia personalizzata al sistema, con tutti i vantaggi che ne conseguono. Allo stesso tempo si impone però la necessità di offrire a chi non abbia né il tempo né le risorse da dedicare alla realizzazione di tale interfaccia personalizzata una serie di soluzioni preconfezionate che sfruttino appieno il protocollo ADES e che ne espongano l'astrazione architetturale coerentemente e consistentemente.

Questo compito non certo semplice (si tratta pur sempre di fornire una sorta di implementazione “di riferimento” dei meccanismi del software) è assolto in questo caso da due software, simili eppure complementari: “Limbo” e “Acheron”. I rapporti fra queste applicazioni e una rappresentazione dell'attuale evoluzione di ADES sono presentati in Figura 6-1.

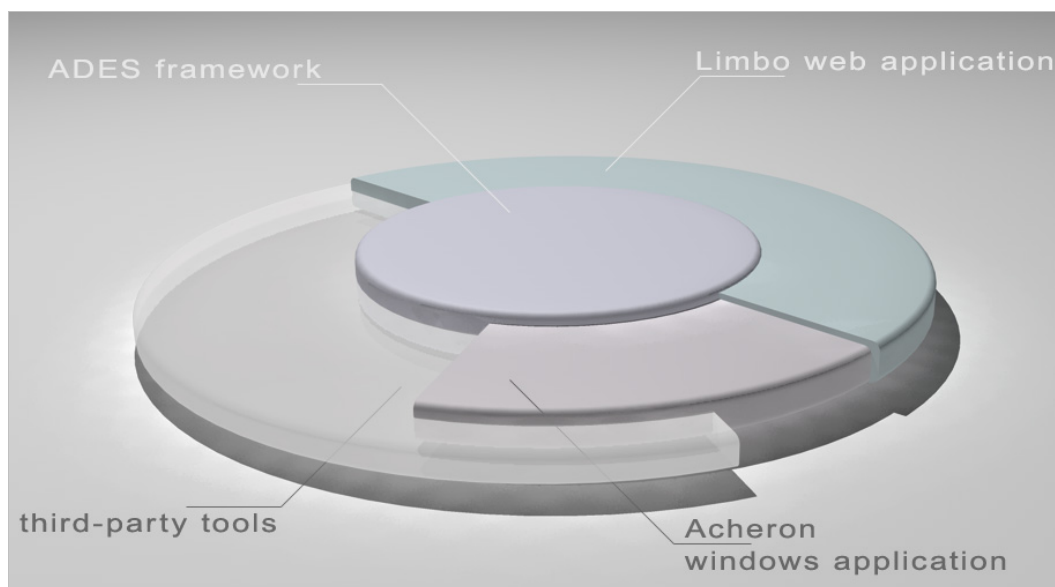


Figura 6-1 - Il sistema ADES e gli applicativi a corredo

Si notano essenzialmente due cose: la prima è la possibilità di poter personalizzare come meglio si crede la soluzione ADES, mentre la seconda è il ruolo centrale del framework, intorno al quale ruotano le funzionalità offerte da Limbo e Acheron.

## 6.2. La windows application Acheron

Dei due applicativi, Acheron è senza dubbio l'applicazione che permette il controllo più “fine” e accurato del sistema, offrendo per ogni componente dello stesso vaste possibilità di monitoraggio ed intervento.

Si tratta inoltre di un'applicazione che offre alla rete esterna l'accesso al nodo ADES locale.

Si è pertanto realizzata un'applicazione destinata all'esecuzione continua e in background. L'esecuzione continuata impone però un'invasività minima dell'applicativo.

Acheron trova pertanto naturale collocazione nella system tray del desktop (si tratta, per intendersi, del gruppo di icone che appaiono accanto all'orologio di Windows), come visibile in Figura 6-2, nella quale appare l'icona rossa dell'applicazione.

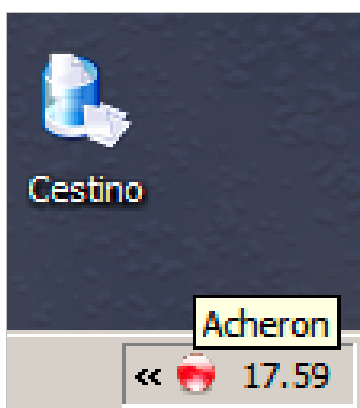


Figura 6-2 - La system tray icon dell'applicativo Acheron

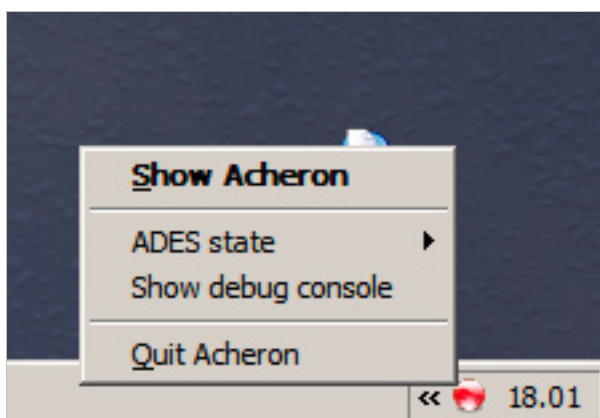


Figura 6-3 - Il menu disponibile per l'icona dell'applicativo Acheron

Da tale icona è possibile accedere, attraverso un menu a comparsa (visibile in Figura 6-3), all'applicazione vera e propria.

Essa è essenzialmente composta da una finestra principale e da numerose finestre di dialogo (attivabili a partire da essa) che permettono l'accesso a tutte le funzionalità del sistema.

Il flusso di lavoro è modularizzato in più finestre e tra quelle che raccolgono funzionalità concettualmente simili si è cercato di mantenere una certa consistenza anche grafica.

La finestra principale riporta alcune proprietà riguardanti il sistema operativo ospitante ADES, alcune riguardanti il .NET environment e altre direttamente connesse ad ADES stesso, così come visibile in Figura 6-4.

Attraverso il menu di questa finestra (localizzato nella sua parte superiore) è poi possibile spostarsi attraverso diverse possibili combinazioni di finestre, ciascuna delle quali svolge un diverso lavoro sui diversi oggetti dell'ambiente, quali servizi, pools, gli ADES entry-points e gli oggetti di configurazione.



La struttura grafica più ricorrente e maggiormente utilizzata è una griglia di proprietà, sicuramente familiare a chi abbia utilizzato i prodotti di sviluppo Microsoft.

Tale griglia permette l'accesso alle proprietà degli oggetti, sia in modalità di lettura che in modalità di scrittura, riproponendo un flusso di lavoro familiare, consistente e uguale per tutte le finestre del sistema.

Fornisce poi una descrizione delle singole proprietà e la loro ripartizione in categorie concettuali diverse.

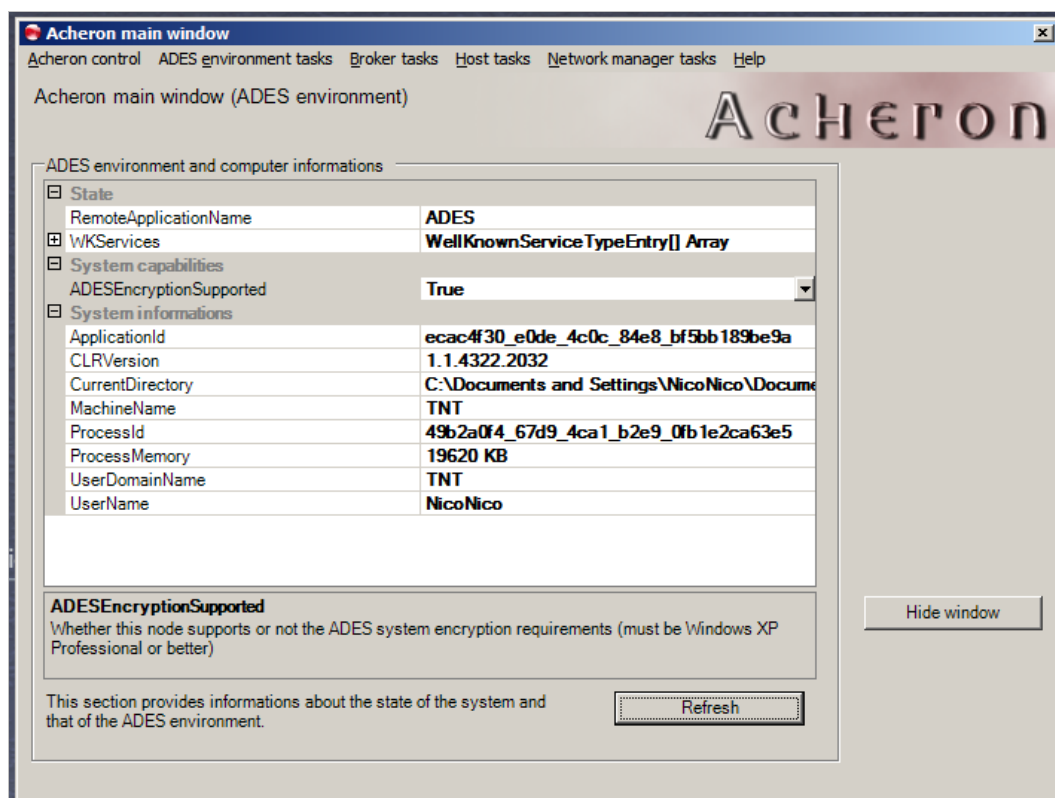


Figura 6-4 - Finestra principale dell'applicazione Acheron

Delle finestre alle quali è possibile accedere a partire da quella principale la prima presentata è quella dedicata alle impostazioni del sistema (Figura 6-5) alle quali viene offerto un comodo accesso (sia in lettura che scrittura).

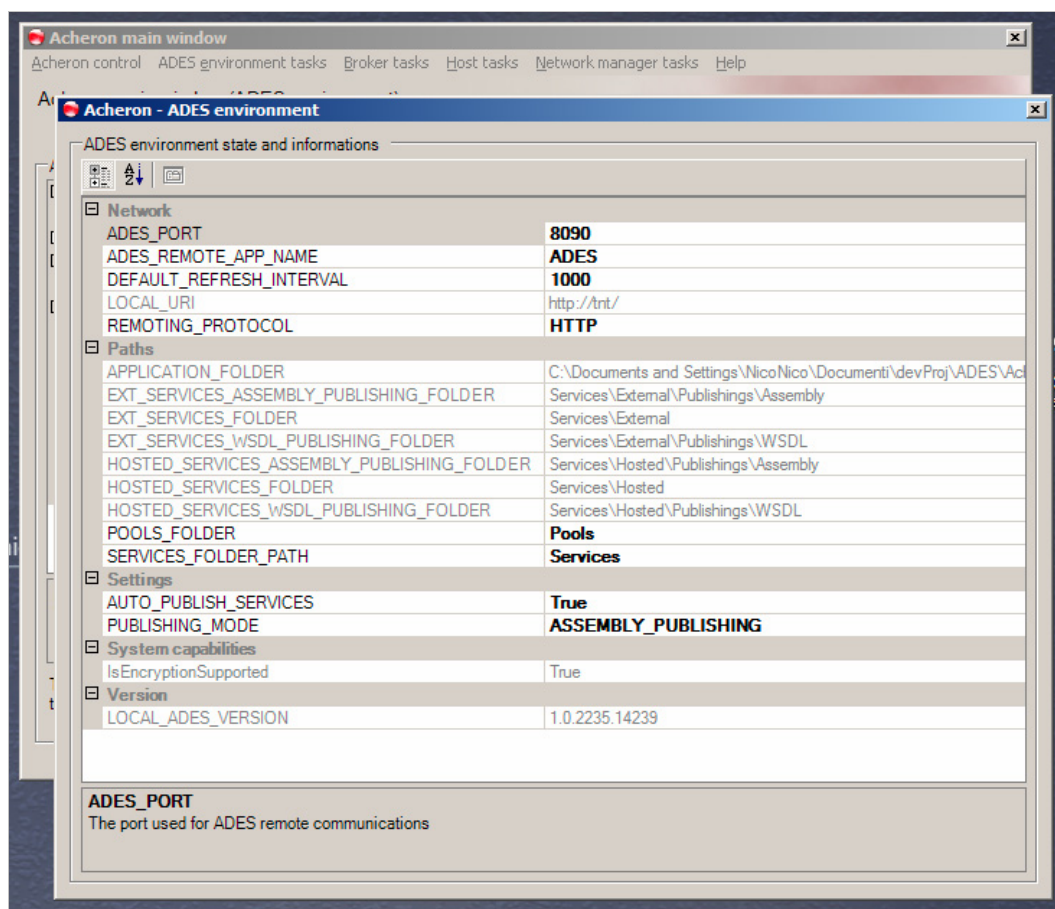


Figura 6-5 - Finestra di configurazione del sistema ADES fornita dall'applicativo Acheron

Altre due finestre sono poi dedicate alle informazioni sull'hosting dei servizi e delle relative pools che li contengono (Figura 6-6).

Quest'ultima combinazione di finestre rispecchia fedelmente l'architettura che ADES segue per la gestione dei diversi gruppi di servizi e riporta ad alto livello le scelte e le linee guida seguite nello sviluppo della libreria.

Le stesse finestre sono dedicate alla gestione di servizi e pools per il broker, poiché si tratta di funzionalità assimilabili fra loro.

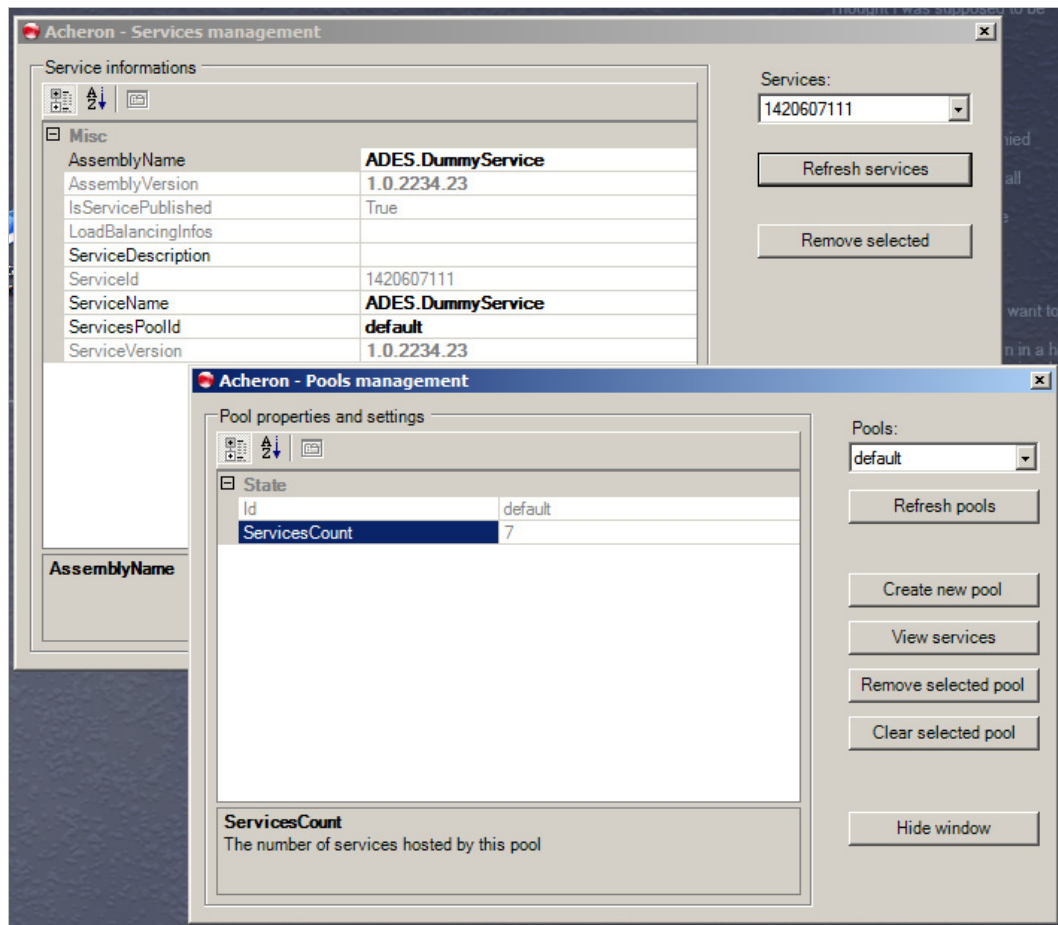


Figura 6-6 - Finestre di personalizzazione dei servizi "hosted" e delle relative pools

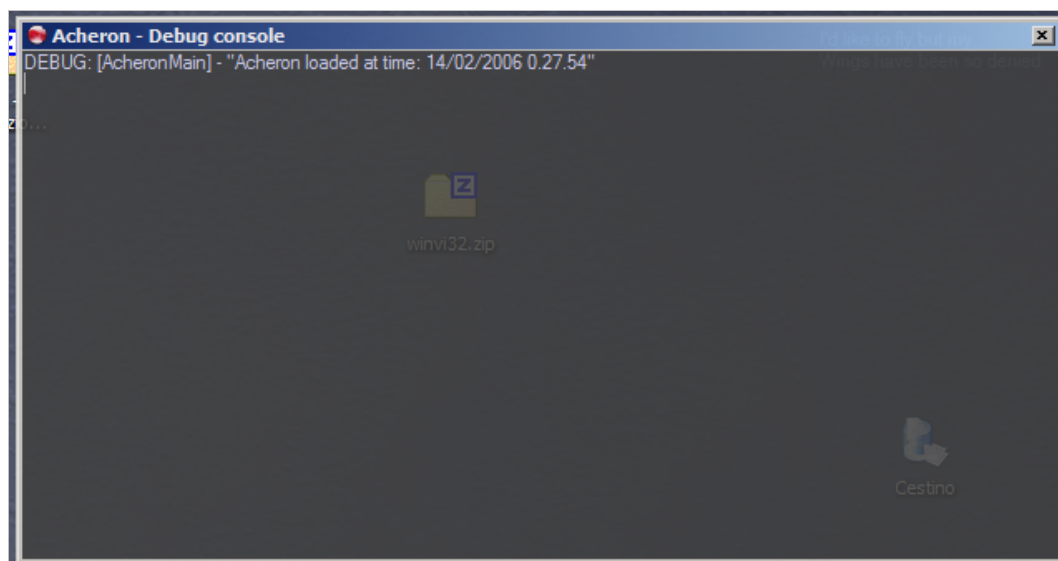


Figura 6-7 - Console di debug e di report degli eventi del sistema

Il sistema offre poi una comoda finestra semitrasparente che riporta le informazioni di esecuzione del sistema, catturate direttamente dal flusso di output della console associata al programma (si veda la Figura 6-7).

Una cosa che senza dubbio si nota è quanto l'applicazione sia graficamente “spartana”. Questa caratteristica di Acheron è riconducibile essenzialmente a due fattori: la già citata esigenza di “non invasività” dell'applicativo è il primo e quella di “non dispersività” è il secondo.

Si tratta di mettere un utente a confronto con un sistema che, per quanto si sforzi di rendere tutto più semplice, deve pur sempre offrire delle funzionalità piuttosto complesse; arricchire l'interfaccia di eccessive funzionalità avrebbe sicuramente rischiato di scoraggiare l'utente.

Ciò non toglie che una futura release della suite potrebbe introdurre in Acheron il meccanismo dei plugins, permettendone estensibilità e arricchimento.

### **6.3. La web application Limbo**

La scelta di privilegiare semplicità e fruibilità la si ritrova anche nel caso della seconda applicazione sviluppata per il progetto ADES, “Limbo”.

Per essa si era non a caso inizialmente scelta una presentazione affidata alle parole “minimal administrative interface for ADES” (poi però mutata in “web administrative interface for ADES”): nel termine “minimal” è racchiuso lo spirito e lo sforzo di offrire un'interfaccia il più possibile semplice e chiara (si veda a questo proposito la Figura 6-8 e la Figura 6-9).

Il risultato è una web application sviluppata in tecnologia ASP.NET che presenta un comodo menu di navigazione e una sezione nella quale vengono proposti i vari flussi di lavoro, peraltro simili a quelli già visti per Acheron.



Figura 6-8 - Vista d'insieme della web application Limbo

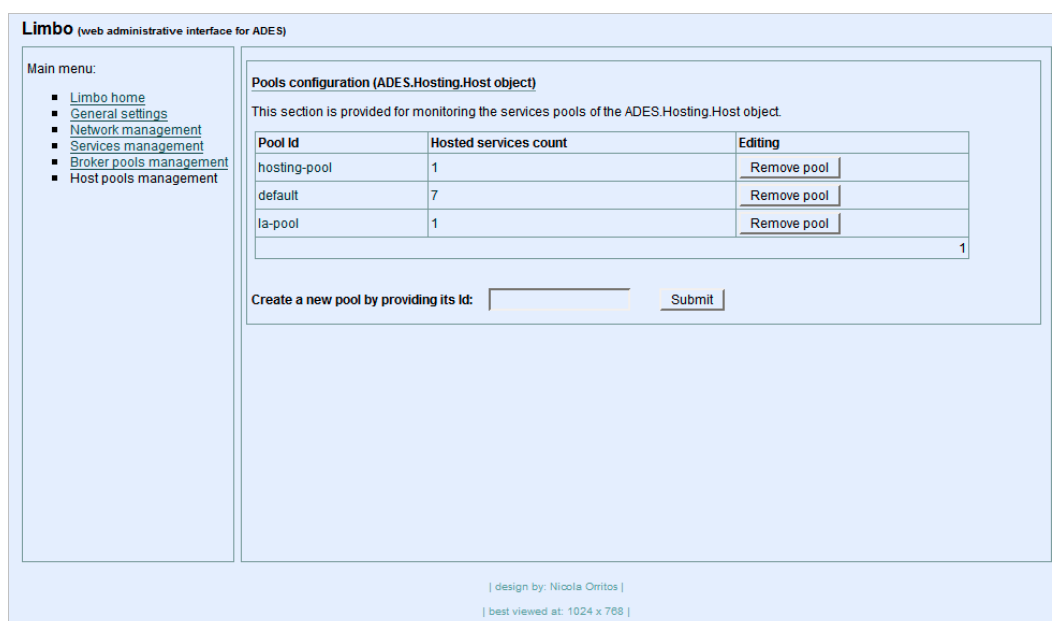


Figura 6-9 - Pagina di controllo delle pools dell'oggetto "Host"

Nella Figura 6-10 è visibile un dettaglio della sezione di configurazione delle proprietà del sistema, durante un operazione di editing di una di esse.

Setting name	Current value for this setting	Editing
EXT_SERVICES_ASSEMBLY_PUBLISHING_FOLDER	Services\External\Publishings\Assembly	<input type="button" value="Edit"/>
HOSTED_SERVICES_FOLDER	Services\Hosted	<input type="button" value="Edit"/>
HOSTED_SERVICES_WSDL_PUBLISHING_FOLDER	Services\Hosted\Publishings\WSDL	<input type="button" value="Edit"/>
HOSTED_SERVICES_ASSEMBLY_PUBLISHING_FOLDER	Services\Hosted\Publishings\Assembly	<input type="button" value="Edit"/>
PUBLISHING_MODE	ASSEMBLY_PUBLISHING	<input type="button" value="Edit"/>
AUTO_PUBLISH_SERVICES	True	<input type="button" value="Edit"/>
DEFAULT_REFRESH_INTERVAL	1000	<input type="button" value="Edit"/>
REMOTING_PROTOCOL	<input type="text" value="TCP"/>	<input type="button" value="Update"/> <input type="button" value="Cancel"/>
ADES_PORT	8090 <input type="button" value="Current settings"/>	<input type="button" value="Edit"/>

Figura 6-10 - Flusso di lavoro di modifica delle proprietà del sistema ADES attraverso Limbo

Si tratta di un'operazione speculare a quella presente nella finestra di Acheron presentata in Figura 6-5 (a pag. 114) e questa osservazione costituisce un ottimo spunto per discutere della complementarietà delle due applicazioni.

Acheron e Limbo sono effettivamente due modi diversi di esporre le funzionalità del sistema ADES; l'uno è un'applicazione che permette il controllo "in-depth" dell'ambiente, l'altro ne consente l'accesso remoto ed espone una interfaccia semplificata per esso.

L'uno agevola l'amministrazione completa del sistema, l'altro agisce quasi da "telecomando" dello stesso (pur essendo ovviamente possibile utilizzarla per controllare un nodo ADES dallo stesso host nel quale esso è ospitato).

La scelta tra le due applicazioni è, a tuttora, obbligatoria. Inoltre l'attuale stadio di sviluppo delle due non ne permette l'esecuzione concorrente: questo significa che non è possibile avviare l'una se l'altra è già in esecuzione.

Inoltre non è possibile avviare due istanze di Acheron, a meno di non vedere comparire l>alert dialog presente in Figura 6-11.

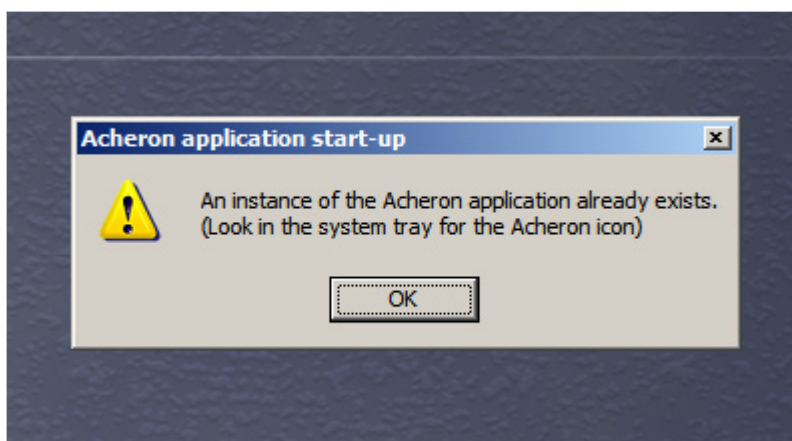


Figura 6-11 - Alert dialog visualizzato al tentativo di esecuzione di più istanze di Acheron

Si tratta comunque di tipici problemi legati al fatto che, dell'intero sistema, i tools grafici presentati sono la parte più “acerba” e meno sviluppata.

Sarebbe sufficiente infatti accentrare il broker ADES in un unico ambiente al quale permettere l'accesso concorrente da parte di più applicazioni; cosa questa realizzabile come “demone” o “servizio”, sia su sistemi Windows che su piattaforme Linux.

Dispiace a questo proposito non aver avuto il tempo di sviluppare un meccanismo del genere, che sarebbe stato certamente utilissimo, ma che presenta purtroppo molte difficoltà di progettazione e realizzazione.

## 7. Panoramica delle tecnologie di brokering presenti sul mercato

È utile concludere introducendo un confronto tra alcune soluzioni presenti sul mercato che offrono funzionalità simili o assimilabili a quelle di ADES.

Si è condotta a questo proposito un'analisi comparata tra queste soluzioni, prendendo in particolare considerazione il livello di usabilità dei tools presentati, la complessità da essi presentata sotto il profilo della gestione e il livello di astrazione al quale il prodotto opera rispetto ai meccanismi che sottostanno a un'attività di brokering.

Le soluzioni software che sono state prese in esame sono:

- la soluzione WebSphere della IBM
- la piattaforma Microsoft Connected Services Framework (CSF), della quale si prenderà in particolare considerazione il prodotto BizTalk Server
- prodotti quali SAP NetWeaver e BEA AquaLogic, i quali sono però stati trattati in maniera meno estensiva

### 7.1. IBM WebSphere Business Integration Message Broker

La suite di prodotti WebSphere comprende soluzioni che intendono portare un'azienda da un contesto di integrazione tramite middleware a una politica service oriented, cercando di garantire un processo il più graduale e “indolore” possibile.

A questo proposito parlare di brokering di servizi in relazione alla soluzione IBM è leggermente discostante dalle reali funzioni del componente che in questa soluzione assolve al ruolo di broker, il

“WebSphere Business Integration **Message Broker**”.



Innanzitutto è necessario inquadrare tale componente nel contesto della suite WebSphere, la quale è essenzialmente composta da:

- WebSphere MQ (Message Queue)
- WebSphere Event Broker
- WebSphere Message Broker

La soluzione è progettata per essere scalabile e adattabile alle esigenze del cliente, in modo tale da offrire solo i componenti dei quali egli abbisogna, senza ulteriori fronzoli e tralasciando di proporre un unico, monolitico prodotto del quale l'utente avrebbe modo di utilizzare sì e no il 30% delle funzionalità.

Inizialmente viene quindi proposto un “nucleo” composto dal solo MQ, un gestore di code di messaggi dotato di un meccanismo “publish/subscribe” e di capacità transazionali.

Di MQ sono poi previste diverse distribuzioni, ognuna rivolta all'integrazione di una classe particolare di applicazioni: MQ Mobile, MQ Real-time, MQ telemetry, MQ Multicast ed MQ Web Services.

Sul core di funzionalità offerte da MQ può poi essere innestato un “Event Broker” per il trattamento di tipologie specializzate di queues, per la gestione di attività di “transport switching” e per l'adozione di meccanismi di “message replay”.

Infine è poi possibile affiancare a questo già ricco insieme di funzionalità anche l'oggetto della nostra analisi, il “Message Broker”, prodotto disegnato con l'intento di mediare un passaggio da un approccio middleware all'adozione di una piattaforma SOA, prevedendo a tale scopo meccanismi avanzati sia sul versante della gestione dei flows di messaggi sia su quello della validazione e trasformazione dei diversi formati nei quali questi messaggi potrebbero presentarsi.

Si è scelto di offrire del prodotto due tipi diversi di “viste”, una focalizzata sulle funzionalità concettuali che esso offre per il trattamento dei messaggi e una che si sofferma invece sulle funzionalità più tecniche che la piattaforma mette a disposizione dell’utente.

Nella prima vista rientreranno quindi una serie di considerazioni sulle possibilità di intervento sui messaggi prima che essi giungano a destinazione.

Nella seconda prospettiva rientrano invece la descrizione dei tools (e delle astrazioni) che danno all’utente la possibilità di definire i flussi dei messaggi tramite diversi linguaggi, di “proteggerli” e di mapparli secondo le proprie esigenze, il tutto usufruendo di un comodo IDE (“Integrated Development Environment”).

A partire dal primo punto di vista, da un tool così complesso e ricco ci si aspetta che possa, per dirla con parole povere, “fare tutto quello che gli si chiede di fare”. Nel caso di WebSphere Message Broker questo è quasi vero, sempre però che ci si fermi a un contesto di brokering “misto”.

Con il termine “misto” si intende dire che questo software tende a trovarsi più a suo agio in un contesto non nativamente “service oriented”, meglio adattandosi ad ambienti per così dire “misti” (pur assolvendo egregiamente ai suoi compiti anche in un ambiente SOA “puro”, beninteso).

In questo senso Message Broker si indirizza verso una fascia di mercato particolarmente ambita, poiché le aziende che hanno bisogno di passare ad una SOA non hanno di certo la minima intenzione di adottare una politica di “salto” verso di essa.

Hanno al contrario bisogno di un periodo e di strumenti di transizione e proprio in questa direzione si muove IBM, sottolineando con enfasi la sua scelta.

La direzione presa da IBM è deducibile anche dal livello di astrazione che il prodotto presenta, unica pecca questa in una soluzione altrimenti quasi perfetta (si vedrà come su questo campo la soluzione Microsoft guadagni invece terreno).

Si tratta infatti di porre l'utente a stretto contatto con problematiche relative a schemi XML, validazione di documenti nei confronti ("against") di tali schemi, trasformazione da un formato (leggasi "schema") all'altro, mapping di documenti non XML in una qualche struttura adatta alla loro manipolazione e altre, poco piacevoli, attività.

Non si pensi comunque che il livello complessivo del prodotto sia relegato a una sfera troppo tecnica; il Message Broker offre infatti un supporto esteso al routing di messaggi "business-rules based" e prevede un meccanismo di diffusione dei messaggi di tipo "Publish/subscribe", molto più vicino al modo di pensare di un business architect di quanto non lo sarebbero altri (e sicuramente molto meno "developer-oriented" rispetto ad essi).

In fin dei conti gli scopi del prodotto sono, dichiaratamente, i seguenti:

- Integrazione di applicazioni senza che sia necessario modificarle o riadattarle
- Meccanismi avanzati per il routing dei dati
- Funzionalità tese a dare continuità ai dati scambiati tra le applicazioni
- Trasformazione dei dati qualora le applicazioni facciano uso di formati incompatibili tra loro (e in questo il Message Broker non agisce diversamente da un middleware)

Scopi che denotano senza possibilità di fraintendimenti la propensione del prodotto al trattamento dei dati senza eccessivi tecnicismi sì, ma senza neanche agire a un altissimo livello di astrazione (cosa questa che rappresenta un certo vantaggio in determinati contesti ma un handicap quasi insormontabile in altri).

Si garantisce comunque un range di funzionalità (oltre a quelle già citate) potenti e di una comodità non indifferente:

- Relativo real-timing delle procedure di routing
- Validazione e trasformazione di numerosissimi tipi di dati, XML **e non**

- Capacità di “message-enrichment”, vale a dire la possibilità di trattare e “arricchire” i messaggi in maniera trasparente sia al mittente che al destinatario degli stessi
- Un supporto a un gran numero di piattaforme diverse che assicura un’interoperabilità pressoché totale.

Considerando la piattaforma WebSphere non si può poi certo prescindere dal considerare la qualità e la semplicità dei tools che essa mette a disposizione sul versante della gestione e programmazione del sistema al quale viene applicata.

Message Broker non fa eccezione, dotando l’utente che debba gestire le code di messaggi di una gamma completa di strumenti, per la maggior parte visuali e quasi di tipo RAD (“Rapid Application Development”), basati principalmente sulla piattaforma Eclipse.

Adottando questo punto di vista (il secondo dei due che abbiamo eletto a privilegiati nell’analisi di questo sistema) il team di business si trova di fronte a un prodotto che può essere astratto in essenzialmente due parti:

- Un “Development Environment” (ambiente di sviluppo)
- Un “Runtime Environment” (ambiente di runtime)

I due “domini” si diramano poi in una serie di astrazioni e sottosistemi. Nel caso dell’ambiente di sviluppo essi sono:

- “Message flows”, un’astrazione utile a modellare flussi di messaggi che possono prendere diverse “direzioni” e che possono coinvolgere diverse applicazioni
- “Message sets”, con i quali vengono identificati una serie di messaggi dotati della stessa conformazione e composizione

- Un IDE basato sulla tecnologia Eclipse, che permette di “modellare” in maniera quasi completamente visuale flussi di messaggi, regole di routing degli stessi, mappings, gestione degli schemi XML, gestione delle autorizzazioni a livello di applicazione e/o di utenti, definizione delle business-rules che influiscono sui flussi (e/o sui sets) e un’infinità di altre combinazioni che rendono il broker uno strumento estremamente duttile e personalizzabile.

Nel caso dell’ambiente di runtime, il cuore del broker, gli strumenti e le astrazioni messe a disposizione sono i seguenti:

- Il broker stesso o, meglio, i broker, poiché è possibile isolare in diversi di essi diverse tipologie di lavoro da eseguirsi sui messaggi
- Una serie di “Execution groups”, gruppi di esecuzione di più flussi di messaggi indipendenti tra loro, ma per i quali possono essere predisposte politiche comuni
- Un insieme di “Broker domains”, raggruppamenti di diversi broker, qualora la suddivisione del lavoro tra diversi brokers non fosse sufficiente
- Uno “User Name Server”, un agente predisposto all’applicazione di politiche di associazione (con la conseguente gestione dei meccanismi di sicurezza) tra utenti, applicazioni, flussi e sets di messaggi.
- Uno strumento di configurazione centralizzato, il “Configuration Manager”
- Una “prospettiva” (configurazione dell’IDE Eclipse operante su un particolare “aspetto” del sistema) amministrativa del Message Broker dalla quale è possibile gestire lo stesso in modo concorrente rispetto ad altri componenti del team di sviluppo (e questa è una vera manna dal cielo qualora si voglia permettere una suddivisione del lavoro senza dover per forza rinunciare a un controllo centralizzato di tutto l’apparato).

Sul versante della programmabilità, su quello dell'estensione e del controllo "fine" dei flussi e della gestione in generale dei messaggi le possibilità offerte sono essenzialmente (ma non solo) due:

- ESQL, una estensione al linguaggio SQL, particolarmente indicata nel caso di interazioni con basi di dati e nel caso si voglia modellare la logica del flusso di messaggi ricorrendo alla sintassi semplice e a molti familiare dell'SQL.
- JAVA, sul quale non è necessario spendere troppe parole, salvo accennare al fatto che i meccanismi interni del broker sono realizzati in questo linguaggio con la conseguente disponibilità di funzioni per l'accesso "a basso livello" al contenuto dei messaggi.

Un'ultima serie di osservazioni va spesa sulla quantità di DBMS supportati: si va da DB2 e Cloudscape (anche noto come "Derby") della IBM a Microsoft SQL Server 2000, dai prodotti della famiglia Oracle a quelli Sybase.

Inoltre il prodotto può essere installato sia su piattaforma Windows che su piattaforma Linux.

Una descrizione approfondita del WebSphere Business Integration Message Broker è [MBRP].

## **7.2. Microsoft Connected Services Framework**

La piattaforma Connected Services Framework (CSF) ben rappresenta la volontà della Microsoft di provvedere alle nuove esigenze che il mercato richiede con l'adozione delle SOAs.

Essa è pressoché completamente dedicata alla gestione di un'infrastruttura IT costruita secondo i dettami della service orientation.

Si tratta di una mossa delicata e azzardata a causa del carattere quasi di “salto” verso una nuova tecnologia che CSF presenta, adottando un approccio meno graduale di quello IBM.

In questo la politica Microsoft perde sicuramente molto nei confronti della soluzione WebSphere.

In un altro campo chiave invece la perdita di terreno è molto più contenuta e si tratta senz'altro di una piacevole sorpresa constatare che è la Microsoft stessa a sottolineare l'interoperabilità della sua piattaforma (BizTalk Server in particolare) con altre tecnologie non strettamente Microsoft.

La piattaforma CSF si compone di essenzialmente tre prodotti: Windows Server 2003, SQL Server e BizTalk Server.

Attorno ai tre principali gravitano una serie di software attraverso i quali avviene l'effettivo utilizzo della soluzione: Visual Studio .NET (o Visual Studio 2005, rilasciato recentemente), Office 2003, Visio 2003, Windows XP Professional e altri prodotti ancora.

Oggetto dettagliato della nostra analisi è il componente della suite che assolve al ruolo di broker. Un ruolo “mascherato”, poiché l'attività di brokering del prodotto viene incapsulata e nascosta nella gestione dei flussi di messaggi associati ai processi di business.

Pertanto se nel seguito si parlerà di processi di business e flussi di messaggi si sottintende che BizTalk Server si fa carico delle attività di brokering tradizionali, evitando di esporre i dettagli tecnici ad esse inevitabilmente correlati.

Analizzandolo nel dettaglio, BizTalk Server (del quale si è presa in considerazione la versione 2004), presenta la solita, tradizionale usabilità dei prodotti Microsoft.

Tornando all'argomento pregi-difetti è quindi sul versante dell'alta astrazione fornita all'utente che il prodotto guadagna sicuramente molto rispetto alla soluzione IBM.

Un guadagno poi incrementato dall'offerta di diversi livelli di astrazione con i quali l'utente può confrontarsi.

Procedendo per gradi è d'obbligo introdurre innanzitutto gli scopi del software:

- Integrazione di applicazioni
- Connessione di applicazioni
- Sviluppo, deploy e gestione di:
  - Processi di business integrati
  - Web Services (basati su XML)

Sostanzialmente però il compito principale al quale si promette di assolvere è quello di integrare applicazioni diverse e di costruire su di esse una serie di processi di business che facciano uso di tali applicazioni come “service providers”.

A tale scopo il prodotto si compone di due parti:

- Una “core engine” (“nucleo operativo”)
- Una serie di servizi, preconfezionati e pronti all'uso, costruiti al di sopra di essa

Le funzioni del nucleo operativo di BizTalk Server sono relative alla modellazione dei processi di business, alla orchestrazione delle applicazioni e alla gestione e monitoring centralizzati delle stesse.



Inoltre occorre chiarire fin dall'inizio della nostra trattazione che BizTalk si basa interamente su messaggi XML. Questo comporta un processo di “traduzione” per quei dati scambiati tra applicazioni che di XML non prevedono, invece, l'uso.

In questo senso si può dire che il prodotto si occupi, a basso livello, di incapsulare e nascondere all'utente una natura middleware dalla quale di solito non si può prescindere quando si lavori con applicazioni legacy (e BizTalk Server si rivolge, anche se in misura minore della suite WebSphere, anche a questa particolare categoria d'impiego).

Qui sta inoltre una certa differenza tra BizTalk Server e il prodotto IBM: nel caso del Message Broker si ha una propensione a mettere al corrente il business strategist, o almeno parte del suo team, delle problematiche legate ai differenti formati dei messaggi; nel caso del software Microsoft si tende invece a innalzare il livello di astrazione e a nascondere tali attività all'utente non sviluppatore.

Tale modo di procedere è riscontrabile anche nella presenza di numerosi “adapters” preconfezionati, destinati a mediare il dialogo tra BizTalk e differenti tipi di canali (SOAP, HTTP, FTP, EDI, SMTP e altri).

Il discorso appena fatto potrebbe però trarre in inganno, perché il prodotto permette comunque un'elevata programmabilità, anche nel caso si debba intervenire su aspetti particolarmente “profondi” del trattamento dei messaggi.

A questo proposito BizTalk Server offre un sistema di processing dei messaggi basato su pipelines a diversi stadi, in ciascuno dei quali è possibile intercettare e pre/post-processare il messaggio XML, sia che esso sia in uscita, sia che esso sia in arrivo.

La diffusione e il routing dei messaggi sono poi affidati a un sistema di “subscriptions” simile a quello di IBM Message Broker.

Questa diversa possibilità di intervenire, a diversi livelli di astrazione, sulla definizione e sulla gestione dei processi di business si schematizza in essenzialmente tre possibili punti di vista che è possibile adottare in un approccio a BizTalk Server:

- Il punto di vista del non sviluppatore è supportato da un add-in di Microsoft Visio, l'ODBA (Orchestration Designer for Business Analysts)
- Il punto di vista dello sviluppatore è dispiegato su tre strumenti:
  - BizTalk Editor, per la definizione di schemi XML
  - BizTalk Mapper, per i mappings tra schemi differenti
  - Orchestration Designer, per la definizione dei flussi di dati relativi ai processi di business
- Il punto di vista del cosiddetto “Business-oriented user” è raccolto nello strumento “Business Rule Composer”, piuttosto potente e praticamente assimilabile a un ambiente RAD (“Rapid Application Development”); le regole definite dal composer vengono infine prese in consegna ed eseguite dalla “Business Rule Engine”.

L'output dei tre strumenti rimane comunque lo stesso: assembly standard .NET. Questo perché i vari (rispettivamente all'elenco precedente) Visio, Visual Studio e lo stesso BizTalk Server agiscono nel contesto del .NET Framework e, nonostante si sia provveduto a fornire delle piacevoli astrazioni, qualora la definizione di un processo di business incontrasse un problema che richiedesse un controllo “fine” sui messaggi scambiati non si è chiusa la possibilità di intervenire tramite i linguaggi C# e Visual Basic .NET, che costituiscono pertanto lo strumento di più basso livello attraverso il quale è possibile modellare un processo di business e i messaggi ad esso correlati.

A una rilettura attenta di quanto appena detto si potrebbe pensare che comunque al prodotto manchi qualcosa per essere veramente usabile.

In effetti è impensabile considerare di offrire un sistema simile senza includere come suo “strato” più esterno una serie di servizi preconfezionati che assolvano alle più comuni tasks normalmente richieste da un “information worker” (quindi né un architect, né uno strategist, né un capo-sviluppatore, né tantomeno uno sviluppatore).

Alla Microsoft non è sfuggito questo particolare e si è deciso di evitare il classico problema della “re-invenzione della ruota” fornendo due classi di servizi:

- Business Activity Services (BAS), dedicati alla gestione di base del business e a loro volta suddivisi in:
  - Trading Partner Management
  - Business Process Configuration
  - Business Process Provisioninge altri ancora.
- Human Workflow Services (HWS), costruiti per la gestione dei flussi di lavoro non informatici e pensati per poter essere utilizzati tramite un semplice programma come Outlook o anche Internet Explorer.

Il funzionamento di BizTalk Server 2004 è relegato purtroppo al solo ambito dei sistemi Windows, la versione Server 2003 soprattutto.

Questo, anche se mitigato dalla promessa di una certa interoperabilità con altri sistemi, rappresenta sicuramente il dazio principale pagato nei confronti di soluzioni come quella IBM.

Rimane però da dire che la scelta di provvedere con una piattaforma dedicata al mondo dei servizi e delle SOA è senz’altro una carta vincente.

Molte compagnie infatti apprezzano la possibilità offerta dal poter lavorare su un insieme di strumenti omogenei per i quali viene garantito un alto rendimento, dovuto proprio al focalizzarsi su un’unica attività comune.

### 7.3. Soluzioni BEA AquaLogic e SAP NetWeaver.

Nel panorama delle piattaforme dedicate alla gestione avanzata di una SOA non si può non menzionare le soluzioni di altri due “competitors” nel mercato dei tools di gestione di una infrastruttura orientata ai servizi.

Si sta parlando di BEA Systems ([www.bea.com](http://www.bea.com)) e di SAP ([www.sap.com](http://www.sap.com)), che fanno parte del nutrito gruppo di compagnie che per prime hanno creduto nelle possibilità di questa nuova tecnologia.

Nel caso di BEA si parla di una compagnia che ha introdotto alcuni tra i concetti fondanti della nuova architettura e che va oltre, proponendone di nuovi (tra i quali quello di “liquid computing”, l’adattarsi del sistema alle sollecitazioni esterne senza che ci sia bisogno del benché minimo intervento umano) e descrivendo gli eventuali scenari futuri possibile evoluzione del paradigma “service oriented”.

Quella proposta è una soluzione completamente dedicata alla gestione di una SOA, basata sul prodotto AquaLogic e su una serie di idee particolarmente innovative.

Una delle più originali presenti in AquaLogic è quella, spesso sottovalutata, di fornire una solida base di accesso ai dati, ovunque essi si trovino, sotto qualunque rappresentazione essi siano disponibili, evitando quanto più possibile allo sviluppatore il compito di dover effettuare trasformazioni tra di esse.

Quantunque i due precedenti prodotti offrano la possibilità di lavorare su dati omogenei, realizzano questa funzionalità dotandosi l’una (IBM) di un linguaggio comune (questo dovrebbe essere il ruolo di ESQL), l’altra (Microsoft) di uno strumento unico, SQL Server.

Nell’ottica adottata da BEA si tratta invece di offrire una “unified view” sui dati, tale che si renda disponibile un’ulteriore livello di astrazione oltre a quello, già piuttosto “alto”, che si avrebbe nel trattare la “fonte” dei dati **come un servizio**.

“AquaLogic Data Services Platform” è il prodotto che BEA dedica a questo proposito, fornendo attraverso esso la funzionalità di “vista unica” sui dati e provvedendo affinché i cambiamenti su di essi effettuati al livello più alto di astrazione si riflettano e si propaghino nel modo corretto sulla loro “reale” rappresentazione.

Si può così evitare di dover sviluppare il meccanismo che fornisce questa visione unificata dei dati, perché esso viene già fornito da Data Services Platform.

Conseguenza diretta da non sottovalutare in questo approccio è l’“omogeneizzazione” dei processi di autenticazione all’atto dell’accesso ai dati, che costituisce spesso un altro problema in sistemi fortemente eterogenei.

Si noti comunque come si rischi di incorrere nel considerare come dati i messaggi scambiati tra i servizi. Si commetterebbe un errore piuttosto grave, poiché i dati con i quali si ha a che fare sono quelli forniti dai svariati “data providers” alle elaborazioni del processo di business. I messaggi non rientrano pertanto nel modello della “Unified View” di AquaLogic Data Services Platform.

Da questo punto di vista la soluzione offerta da BEA propone senz’altro qualcosa in più delle corrispettive tecnologie IBM e Microsoft.

D’altronde si ha a che fare con una delle aziende che più delle altre spingono per l’adozione dell’architettura orientata ai servizi, uno dei primi sostenitori di questa tecnologia e uno dei più attivi innovatori di essa.

\*\*\*

Un’altra compagnia attivamente coinvolta nel settore delle architetture orientate ai servizi di livello enterprise è SAP ([www.sap.com](http://www.sap.com)).

L’offerta di prodotti di SAP si basa su una visione dei servizi definita “Enterprise Services Architecture” che è al contempo un “blue-print” (un piano di sviluppo, un programma) e un’architettura applicativa, basata innanzitutto sull’assunzione che la transizione da vecchi paradigmi a nuove tecnologie debba essere graduale.

Cardine delle attuali soluzioni rivolte allo sviluppo service oriented è SAP NetWeaver, una suite software che, nelle parole della stessa SAP, diventerà il fulcro di tutte le soluzioni offerte in futuro dall'azienda.

NetWeaver copre l'intero ciclo di vita di una SOA, a partire dal suo sviluppo, comprendendo il deployment dei servizi e l'amministrazione del sistema.

L'ordine di grandezza al quale la soluzione SAP si rivolge è quello degli Enterprise Services e lo scopo dichiarato del prodotto è quello di estendere all'ambito enterprise i benefici dei Web Services.

Non avrebbe senso, come giustamente osserva SAP, dotarsi di uno strumento potente capace di realizzare una enterprise network orientata ai servizi se poi non si fornisce la funzionalità di poter costruire applicazioni che di tale rete facciano uso.

NetWeaver non trascurava di coprire questo aspetto, provvedendo affinché anche lo sviluppo delle applicazioni sia agevole, a partire dalla realizzazione di applicazioni composite (che traggono beneficio cioè dalla composizione e dall'orchestrazione di più servizi), fino alla (apparentemente) banale realizzazione delle interfacce utente, passando per l'applicazione di patterns e metodologie collaudate a qualsiasi fase dello sviluppo di tali applicazioni.

Si noti come la soluzione SAP sia agevolmente integrabile con Microsoft .NET e IBM WebSphere tramite l'utilizzo di "adapters". Non si pregiudica così l'interoperabilità del prodotto, tenendo in stretta considerazione la possibilità per una compagnia di realizzare dinamiche di B2B senza dover affrontare il benché minimo problema di integrazione presente o futuro (si stima che WebSphere e .NET da soli copriranno una fetta consistente del mercato dei sistemi IT service-oriented).

#### 7.4. Tavole riepilogative di confronto

Vengono di seguito presentate alcune tavole riepilogative utili per effettuare un confronto veloce e schematico fra “Microsoft BizTalk Server” e la suite “WebSphere Business Integration”.

Restano “esclusi” dal confronto i prodotti SAP e BEA, non certo perché non li si ritenga altrettanto validi, ma esclusivamente perché una trattazione completa anche di essi avrebbe appesantito non poco questo lavoro.

	Microsoft BizTalk Server 2004	WebSphere BI (Business Integration)
Formati dei messaggi	Pesantemente XML-based	XML, non-XML
Processing dei messaggi	Pipelines	Message flows, message sets
Sistema di routing dei messaggi	Publishing / Subscribe	Publishing / Subscribe
Integrazione “esterna” (B2B e gestione dei Trading Partners)	Attraverso standards (Web Services soprattutto)	Mediante l’utilizzo di Connectors
Interoperabilità	Limitata (è previsto l’uso di adapters per la sola interoperabilità con diversi canali)	Vasto range di piattaforme, architetture e tecnologie supportate

Tabella 7-1 - Aspetti tecnici di confronto fra i due prodotti

	Microsoft BizTalk Server 2004	WebSphere BI (Business Integration)
Servizi built-in (per l'information worker)	Si	No
ESB (Enterprise Service Bus)	No	Si
Supporto a BPEL	Si	Si

Tabella 7-2 - Aspetti e scelte architetturali

	Microsoft BizTalk Server 2004	WebSphere BI (Business Integration)
IDE	Visual Studio, Visio	Eclipse (e altri tools complementari)
Livello di conoscenze tecniche richieste	Medio	Alto
Business-user orientation	Maggiore	Minore
Information-worker orientation	Maggiore	Minore
Livello di integrazione dei tools di gestione e di progettazione	Maggiore	Minore
Enterprise-level orientation	Minore	Maggiore
Vocazione all'integrazione di sistemi eterogenei	Minore	Maggiore

Tabella 7-3 - Confronto logico-concettuale e aspetti relativi all'astrazione fornita all'utente



## 8. Conclusioni

Non resta che esprimere alcune considerazioni finali sul sistema ADES e su questo lavoro.

La prima e sicuramente più importante è che ADES è un applicazione ancora troppo immatura per poter essere considerata “commercializzabile”, ma chi scrive è convinto del fatto che alla base essa sia costruita su idee solide e innovative.

Il modello che descrive ADES e che ne rappresenta il progetto è, sempre nella personale opinione dell'autore, piuttosto grezzo ma contiene alcune soluzioni eleganti e portabili, che potrebbero essere ulteriormente sviluppate e messe a frutto.

Inoltre le possibilità di personalizzazione e sviluppo del protocollo ADES sono pressoché infinite e, di sicuro, sfruttabili proficuamente.

Acheron e Limbo sono poi la direzione che si dovrebbe seguire con ben impresse in mente due parole: semplicità e modularità.

Detto questo non si può non ignorare che è sicuramente una buona idea offrire un insieme di funzionalità tali da favorire l'adozione di un'architettura orientata ai servizi nascondendo alla vista dell'utente tutta una serie di dettagli eccessivamente tecnici e di procedure laboriose.

In quest'ultimo periodo è raccolto il vincolo più importante che l'autore si è imposto durante lo sviluppo del progetto: la semplicità.

Si è cercato di rendere le cose semplici a tutti i livelli del sistema, ma soprattutto a quello più esterno, quello a diretto contatto con l'utente.

Infine non resta che ringraziare chi fosse giunto a questo punto della lettura per lo sforzo che certi argomenti, ci si rende conto, comportano.

Cagliari,

14/02/06

## Bibliografia

[SMRW]: “*SOA meets the real world*”, (serie di articoli) “InfoWorld.com” (www.inforld.com)

[WSA]: “*WSA (Web Services Architecture)*”, W3C Working Draft (8 Agosto 2003) (www.w3.org)

[CAWP]: “*Gestire un’architettura orientata ai servizi (SOA) in un contesto di Computing on-demand*” di Dmitri Tcherevik, Computer Associates Italia (CA<sup>®</sup>) (www.ca.com)

[BPE]: “*Business Process Execution: la nuova frontiera dei Web Services*” di Camilla Tamburini; “Oracle Developer’s Point - Programmazione.it” (19 Maggio 2005) (www.programmazione.it)

[CSZN]: “*The Hartford: A case study in real-world UDDI adoption*” di Jason Bloomberg, ZapThink LLC (30 Marzo 2004) (www.zapthink.com)

[TANB]: “*Reti di calcolatori*”, Andrew S. Tanembaum, Prentice Hall International (2000), paragrafo 5.5

[MBRP]: “*WebSphere Message Broker basics*”, Saida Davies, Laura Cowen, Cerys Giddings, Hannah Parker, IBM<sup>®</sup> (www.ibm.com/redbooks)

[BEA1]: “*The ABCs of SOA*”, BEA<sup>®</sup> Systems (www.bea.com)

[BEA2]: “*Service Infrastrucure*”, BEA<sup>®</sup> Systems (www.bea.com)