Università degli Studi di Padova

Enterprise Architecture e SOA in Provincia di Bolzano

Relazione finale di Corso di Laurea

Laureando: Andrea Cuzzolin
Relatore: Prof. Giorgio Clemente

Dipartimento di Ingegneria dell'Informazione
Anno Accademico 2013-2014

A mia moglie e alla mia famiglia

Indice

Introduzione

La Provincia di Bolzano negli ultimi anni si è trovata ad affrontare una situazione complessa, che vede da un lato una costante riduzione del personale e della possibilità di spesa, dovuta in particolare al patto di stabilità, e dall'altro una richiesta di servizi informatici sempre crescente da parte sia dei collaboratori interni all'ente, sia dei cittadini e delle imprese. Per far fronte a queste circostanze, l'amministrazione provinciale ha intrapreso un significativo percorso di riorganizzazione dell'informatica che ha toccato vari aspetti dell'IT, sia organizzativi che tecnologici.

Il presente documento intende dare una panoramica delle attività intraprese dalla Provincia di Bolzano in questi ambiti, concentrando l'attenzione in particolare sulla scelta di adottare l'Enterprise Architecture, la disciplina che studia la struttura di un'organizzazione, i suoi processi operativi, i suoi obiettivi, i sistemi informativi a supporto, i flussi informativi e le tecnologie utilizzate, fornendo la visione di come tutti questi elementi si legano tra loro.

Nel primo capitolo viene introdotto il contesto di riferimento dell'IT nella Provincia di Bolzano, vengono descritti i problemi incontrati dall'amministrazione e i provvedimenti messi in atto per risolverli.

Nel secondo capitolo viene data una panoramica sulla Service-Oriented Architecture (SOA), lo stile dell'architettura scelto dalla Provincia di Bolzano per aggiornare la propria metodologia di progettazione del software e fornire servizi in modo agile ed efficiente alle unità organizzative della provincia, mettendole in condizione di raggiungere i propri obiettivi sfruttando la spinta innovativa fornita dall'IT.

Sono illustrati i concetti e le proprietà fondamentali della SOA: servizi, messaggi, disaccoppiamento, interoperabilità, astrazione, riusabilità, ecc. Sono inoltre introdotti i fondamenti della tecnologia Web services, che è alla base della maggior parte delle implementazioni SOA.

Nel terzo capitolo è descritta in modo approfondito l'implementazione della SOA adottata in Provincia di Bolzano. Si tratta del framework COSM™ della società

Herzum Software, un framework orientato ai componenti che costituisce un approccio industriale alla gestione dell'informatica e allo sviluppo del software. Sono illustrate in questo capitolo le caratteristiche del framework e la modalità di sviluppo delle applicazioni basate sul modello COSM.

Alla fine viene ripresa la definizione dell'Enterprise Architecture ed esposto in dettaglio il ruolo dell'Enterprise Architect e sono illustrati i pilastri su cui si basa l'Enterprise Architecture in Provincia di Bolzano: TOGAF® e ArchiMate®.

Il framework TOGAF (The Open Group Architecture Framework) è un framework aperto, creato dall'Open Group per supportare enti ed imprese nella progettazione e nella governance di architetture informatiche complesse. TOGAF non prescrive alcun linguaggio per la modellazione di un'architettura. È per questo che viene proposto dall'Open Group l'utilizzo di ArchiMate, un linguaggio aperto ed indipendente in grado di descrivere processi di business, strutture organizzative, flussi informativi, sistemi IT ed infrastruttura tecnologica sottostante, fornendo all'architetto enterprise uno strumento per visualizzare in modo univoco le relazioni tra i diversi domini architetturali con l'obiettivo di soddisfare al meglio i requisiti del business.

L'autore di questo documento è membro del gruppo Enterprise Architects della Provincia di Bolzano. Assieme agli altri componenti del gruppo ha supportato il direttore della ripartizione Informatica nelle scelte di carattere architetturale illustrate in questo documento, collaborando alla realizzazione dei progetti e alla definizione dell'architettura di riferimento.

1 L'IT nella provincia di Bolzano

Il capitolo fornisce una visione di insieme della situazione dell'Information Technology nell'amministrazione della Provincia Autonoma di Bolzano, dei cambiamenti avvenuti negli anni e di come sia stata necessaria una sostanziale e completa riorganizzazione per poter rispondere in modo efficace alle esigenze degli utenti, sia interni che esterni all'amministrazione, in un contesto generale di contenimento dei costi.

Sono sottolineate le circostanze e le motivazioni che hanno guidato le scelte della Provincia Autonoma di Bolzano in termini di metodologie e framework. Argomenti che saranno approfonditi nei capitoli seguenti.

1.1 Il contesto di riferimento

La Provincia di Bolzano è una provincia a statuto speciale, ed ha pertanto competenze paragonabili a quelle di una regione a statuto ordinario.

L'amministrazione ha in organico circa 13.000 dipendenti, compresi i docenti delle scuole provinciali, e gestisce approssimativamente 7.500 postazioni informatiche, situate in circa 350 sedi dislocate sul territorio provinciale (uffici, scuole, stazioni forestali, punti logistici, magazzini, ecc).

Per la realizzazione e la gestione dei sistemi informativi, la Provincia di Bolzano si avvale di un'unità organizzativa interna, la ripartizione Informatica.

Come la Provincia di Bolzano, anche le altre pubbliche amministrazioni locali presenti sul territorio (come p. es. l'Azienda Sanitaria o i Comuni dell'Alto Adige) dispongono di una propria struttura informatica interna.

Con legge provinciale n.33 dell'8 novembre 1982 la Provincia Autonoma di Bolzano ha costituito la società Informatica Alto Adige SpA, una società per azioni a partecipazione interamente pubblica, avente l'obiettivo di fornire agli enti pubblici

locali servizi in ambito informatico con le caratteristiche di flessibilità ed efficienza proprie delle imprese private.

Le strutture informatiche delle pubbliche amministrazioni locali e la società Informatica Alto Adige collaborano da tempo, con l'obiettivo comune di erogare servizi informatici ai dipendenti degli enti stessi, ai cittadini e alle imprese dell'Alto Adige.

1.2 Le sfide dell'informatica nella pubblica amministrazione

Negli ultimi anni i comparti IT delle pubbliche amministrazioni sono stati costretti ad affrontare una situazione complessa, che vede da un lato una costante riduzione del personale e della possibilità di spesa, dovuta in particolare al patto di stabilità, e dall'altro una richiesta di servizi informatici sempre crescente da parte sia dei collaboratori interni agli enti, sia dei cittadini e delle imprese.

Una significativa riduzione del budget dedicato all'informatica si è verificata a livello globale anche nel settore privato, come si deduce dalle cifre riportate da uno studio di KPMG International:

- 67% dei budgets IT vengono ridotti;
- 70% dei budgets IT vengono spesi per la sola manutenzione delle infrastrutture, lasciando poche risorse all'innovazione;
- 50% dei CIOs sostengono di lavorare con organici insufficienti;
- 46% dei CIOs ritengono sempre più difficile e costoso fornire i livelli di servizio richiesti.

I principali compiti della struttura IT della Provincia di Bolzano consistono:

- nella gestione e mantenimento in esercizio dei sistemi informativi esistenti;
- nello sviluppo di nuovi sistemi informativi.

La gestione dei sistemi informativi esistenti, con tutti i sistemi e i servizi infrastrutturali correlati (rete, storage, server, database, application server, mail server, gestione documentale, ecc.), è di per sé piuttosto impegnativa, a causa dell'elevato numero di utenze, sistemi e postazioni di lavoro informatiche da mantenere, ed è resa più complessa dall'elevato numero di applicazioni da gestire, che costituiscono il parco applicativo dell'amministrazione: si tratta di oltre mille software, molti dei quali sviluppati ad hoc per l'amministrazione negli anni passati con tecnologie e linguaggi ormai obsoleti.

Un altro problema è dato dal grande numero di sedi periferiche e scuole, dislocate sul territorio e collegate alla sede centrale con linee di rete inadeguate: i collegamenti lenti costringono ad adottare architetture software ridondate e replicate, rendendo più complesso ed oneroso lo sviluppo e la gestione delle applicazioni, rispetto ad un modello centralizzato dove le applicazioni risiedono in un *Data Center* centrale.

Per i software in gestione dell'amministrazione provinciale è necessario garantire la normale operatività, ma anche le diverse tipologie di manutenzione:

- correttiva, per la correzione di errori e malfunzionamenti;
- adattativa, per l'adeguamento dei programmi al fine di ottemperare p. es. ai requisiti di nuove normative;
- evolutiva, per l'introduzione di nuove funzionalità in modo da rendere più efficiente e produttivo il lavoro degli utenti.

Oltre alla manutenzione dei software e sistemi esistenti, la ripartizione Informatica della provincia autonoma di Bolzano è responsabile dell'evoluzione dei sistemi informativi attraverso lo sviluppo di progetti informatici richiesti dalle ripartizioni provinciali.

Ormai sempre più spesso gli utenti necessitano di sistemi tecnologicamente complessi e interoperabili, che consentano di integrare dati provenienti da diversi ambienti, che siano accessibili sia dalla intranet aziendale che da internet, utilizzando dispositivi con sistemi operativi diversi (PC, Tablet, Smartphones, ecc).

La sfida principale dell'IT è quindi quella di fornire servizi in modo agile ed efficiente alle unità organizzative della provincia, mettendole in condizione di raggiungere i propri obiettivi sfruttando la spinta innovativa fornita dall'IT, in un contesto generale di riduzione delle risorse, sia in termini di budget a disposizione che di personale.

1.3 La riorganizzazione dell'IT della provincia di Bolzano

Al fine di poter soddisfare i requisiti sopra esposti − garantire l'operatività dei servizi attuali ed erogare nuovi servizi mantenendo un elevato livello qualitativo, ma riducendo nel contempo i costi − è stato necessario prevedere e mettere in atto una significativa riorganizzazione delle strutture informatiche esistenti.

Obiettivi principali della riorganizzazione sono:

- garantire una strategia IT unitaria per tutti gli enti pubblici locali;
- garantire una gestione sicura e il più possibile unitaria di tutti i servizi IT necessari alla pubblica amministrazione, con un adeguato livello di qualità;
- ridurre i costi di gestione e manutenzione di hardware e software, rendendo efficiente la gestione del parco applicativo ed eliminando ciò che è obsoleto o ridondante, realizzando economie di scala in virtù dell'accorpamento di servizi comuni, dell'utilizzo di infrastrutture condivise e standardizzazione dei processi;
- gestire la complessità, realizzando una vista panoramica di tutte le applicazioni (mappa applicativa) e garantendo che queste cooperino per garantire i servizi del business, senza ridondanze o inefficienze;

- rispondere in modo rapido ed agile alle esigenze e alle richieste del business;
- trasformare il ruolo dell'IT nell'amministrazione pubblica, da semplice erogatore di servizi IT a consulente strategico per il business in grado di suggerire come ottimizzare i processi e implementarli con soluzioni tecnologiche integrate.

Per raggiungere questi obiettivi la Provincia di Bolzano si è avvalsa della consulenza di imprese specializzate, tra cui il Politecnico di Milano, Dipartimento di ingegneria gestionale − School of Management − ed ha intrapreso diverse iniziative su più fronti.

- Ha avviato un processo di *accorpamento dei reparti informatici* degli enti pubblici altoatesini con la società Informatica Alto Adige, al fine di raggiungere i seguenti obiettivi:
  - unificare il governo delle strategie informatiche per i diversi enti locali;
  - fornire servizi condivisi, comuni tra le amministrazioni coinvolte (p. es. servizi di conservazione sostitutiva, protocollo informatico, ecc);
  - realizzare economie di scala per l'acquisto di beni e servizi;
  - mettere a fattor comune le risorse e le esperienze dei collaboratori degli enti coinvolti;
  - consolidare le infrastrutture tecnologiche presenti sul territorio in un unico Data Center centralizzato.
- Ha predisposto diversi *percorsi formativi* per i dipendenti, al fine di ottenere profili professionali altamente specializzati (Demand Manager, Enterprise Architect, Supply Manager, Project Manager, Service Manager, ecc.).
- Ha individuato e definito *settori organizzativi distinti* (BUILD, RUN, PMO, Supply Management, Demand Management, Enterprise Architecture, Centers of Competence) con competenze ben definite, responsabilità e compiti precisi, stabilendo in modo chiaro i flussi di interazione tra i diversi settori per l'evoluzione e la gestione dei sistemi IT.
- Ha adottato *framework e metodologie* per la standardizzazione dei processi dei diversi settori (p. es. ITIL, TOGAF, PMI).
- Ha fatto proprio un *approccio architetturale orientato ai servizi* (SOA) per l'erogazione di servizi IT ai cittadini e ai propri dipendenti.

1.4 Obiettivi della tesi

Questo documento intende dare una panoramica delle attività intraprese dalla Provincia di Bolzano in questi ambiti, concentrando l'attenzione in particolare sull'Enterprise Architecture e sulla Service-Oriented Architecture con l'impiego del framework TOGAF e del linguaggio di modellazione ArchiMate. Ciò principalmente

per sottolineare il ruolo strategico che esse rivestono nell'obiettivo di allineamento dell'IT alle esigenze del business dell'amministrazione.

Di seguito sono riportate alcune definizioni:

***Enterprise**: Un'organizzazione o un insieme di organizzazioni caratterizzate da un comune obiettivo. [The Open Group]*

***Architecture**: La struttura dei componenti, le loro relazioni, i principi e le linee guida che ne governano la progettazione e l'evoluzione. Una architettura è una descrizione formale di un sistema, o un piano dettagliato di un sistema ad un livello tale da guidarne l'implementazione.[ISO/IEC 42010:2007]*

***Enterprise Architecture**: l'Enterprise Architecture è pertanto la disciplina che studia la struttura di un'organizzazione, i suoi processi operativi, i suoi obiettivi, i sistemi informativi a supporto, i flussi informativi e le tecnologie utilizzate, fornendo la visione di come tutti questi elementi si legano tra loro.*

Il termine "enterprise" nel contesto della "Enterprise Architecture" può rappresentare una impresa - comprendendo l'insieme dei servizi informativi e tecnologici, i processi e l'infrastruttura - o uno specifico dominio interno ad una impresa. In entrambi i casi, l'architettura incrocia diversi sistemi e diversi gruppi funzionali all'interno dell'impresa.

La finalità primaria dell'Enterprise Architect è quella di garantire il maggior allineamento possibile tra business e IT all'interno della propria azienda, progettando i servizi IT al fine di soddisfare gli obiettivi organizzativi desiderati. Uno degli stili architetturali di cui l'Enterprise Architect dispone, e che si presta bene per realizzare questi obiettivi, è la *Service-Oriented Architecture (SOA)*.

Esistono diversi framework architetturali, che definiscono una metodologia standardizzata per la progettazione, attuazione, gestione e governance di un'architettura enterprise. La Provincia di Bolzano ha adottato il framework *TOGAF*.

Infine è fondamentale per l'Enterprise Architect poter disporre di un formalismo grafico di modellazione in grado di descrivere le componenti architetturali dei diversi domini (business, applicazioni, dati, tecnologia) e le relazioni tra di loro in modo standardizzato, univoco e comprensibile agli attori coinvolti nei processi di business. Un linguaggio con tali caratteristiche, che il gruppo degli Enterprise Architects sta adottando, è *ArchiMate*, un linguaggio aperto e indipendente per modellare la Enterprise Architecture, che supporta in modo univoco la descrizione, l'analisi e la visualizzazione delle architetture all'interno dei domini e le relazioni delle architetture tra un dominio e l'altro.

L'approccio adottato dalla Provincia di Bolzano può essere pertanto riassunto nello schema seguente, dove è rappresentata l'impostazione che l'amministrazione provinciale ha dato alla disciplina dell'Enterprise Architecture, basata sui tre "pilastri":

- lo stile architetturale *SOA*
- il framework metodologico *TOGAF*
- il linguaggio di modellazione *ArchiMate*

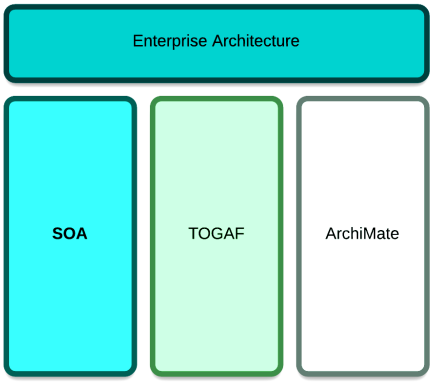


Figura 1.1. Approccio all'Enterprise Architecture in Provincia di Bolzano

2 SOA - Service-Oriented Architecture

Il capitolo fornisce un'introduzione teorica alla Service-Oriented Architecture (SOA). Sono definiti i concetti fondamentali della SOA come i servizi, i registri ed i messaggi. Sono inoltre descritte le proprietà delle piattaforme per realizzare soluzioni orientate ai servizi, quali il disaccoppiamento, l'astrazione, il riuso, la componibilità, ecc., ed infine vengono introdotti i web Services, ovvero la tecnologia che meglio di tutte incorpora i principi alla base della service-orientation.

2.1 Introduzione

Citando la definizione data da Anne Thomas Manes (Director of research at Burton group), *SOA è uno stile architetturale in cui le attività di business sono modellate e implementate come servizi. Un servizio racchiude specifiche funzionalità di business e le rende disponibili ad applicazioni ed altri servizi.*

In altre parole, SOA è un modo di progettare le applicazioni che pone al centro il concetto di "servizio", inteso come un'unità di logica applicativa autonoma ed indipendente che offre una certa funzionalità del business (p. es. la protocollazione di un documento, l'emissione di un certificato, il calcolo di un contributo). I processi di business vengono implementati dall'IT come un insieme di servizi tra loro interoperanti. In questo modo le applicazioni diventano "collaborative", si semplifica l'interoperabilità tra le applicazioni e si assicura una maggiore flessibilità nei confronti dei cambiamenti e delle evoluzioni future.

Nell'ambito di un'architettura SOA è quindi possibile modificare in maniera relativamente semplice la combinazione nella quale i servizi vengono utilizzati nel processo, così come risulta più agevole aggiungere nuovi servizi e modificare i processi per rispondere a nuove richieste del business.

SOA si prefigge l'obiettivo di favorire l'allineamento tra Business e IT[1], creando dei servizi IT che implementano i corrispondenti processi di business e che siano facilmente modificabili ed integrabili seguendo i cambiamenti richiesti dal business.

SOA rappresenta quindi da un lato una importante soluzione basata su standard per garantire l'interoperabilità tra programmi software indipendentemente dai linguaggi e dai sistemi operativi impiegati, dall'altro si prefigge di colmare il gap tra i processi e le attività svolte dall'azienda e gli strumenti ICT che le supportano.

L'architettura orientata ai servizi è particolarmente adatta per le aziende che presentano una discreta complessità di processi e applicazioni, dal momento che agevola l'interazione tra le diverse componenti aziendali.

2.2 La separazione delle responsabilità

L'approccio service-oriented utilizza la tecnica detta "separazione delle responsabilità" (*separation of concerns*[2]), secondo la quale è più facile risolvere un problema se lo si scompone in unità logiche più piccole correlate tra loro. Ciascuna unità logica analizza e risolve una parte del problema.

In questo modo con SOA si ottiene un modello nel quale la logica di programmazione è scomposta in un insieme di unità autonome di logica più piccole.

Suddividere e distribuire la logica di programmazione non è un approccio nuovo. È alla base di molte tecniche di programmazione tradizionali, dalla programmazione strutturata all'object-oriented programming.

L'aspetto più innovativo di SOA, è che la "separation of concerns" viene estesa fino a coinvolgere il business, la sua organizzazione e i suoi processi.

Nella SOA le unità di logica sono definite come *servizi.*

2.3 Servizi

Formalmente possiamo definire un *servizio* nel modo seguente:

---

[1] L'allineamento tra business e IT si raggiunge quando un'organizzazione aziendale è in grado di utilizzare le tecnologie dell'informazione nel modo più efficace per raggiungere i propri obiettivi di business.

[2] nell'IT "separation of concerns" è un principio di progettazione del software che prescrive di suddividere un programma in sezioni distinte, in modo tale che ciascuna sezione affronti uno specifico problema e realizzi una specifica funzionalità. Se gli ambiti sono ben separati, le singole sezioni del programma possono essere sviluppate e modificate indipendentemente l'una dall'altra.

un servizio è una rappresentazione logica di un'attività ripetibile che ha un risultato specifico (p. es. "protocolla documento", "rinnova patente", "rilascia certificato di nascita") con le seguenti proprietà:

- *è autonomo*
- *può essere composto da altri servizi*
- *è una "black box" per chi lo consuma (ovvero chi lo utilizza è inconsapevole di come il servizio funzioni e sia organizzato al suo interno).*

Per essere autonomo, il servizio deve incapsulare completamente la logica di uno specifico contesto, che può essere un'attività del business, un'entità di business o qualche altro raggruppamento logico.

Al fine di garantire che i servizi, pur mantenendo la propria autonomia, siano interoperabili, e quindi impedire che restino isolati, la SOA stabilisce un insieme di principi che consentono ai servizi di evolvere liberamente, mantenendo un livello sufficiente di standardizzazione.

Un servizio può contenere anche logica fornita da altri servizi. In questo caso uno o più servizi si compongono per formare un servizio più ampio (cioè che fornisce più funzionalità).

Quando si progetta una soluzione informatica che comprende servizi, ogni servizio può incapsulare un'attività costituita da un singolo passo del processo di business, oppure un sottoprocesso consistente in una serie di passi del processo. Un servizio può anche incapsulare l'intera logica di processo.

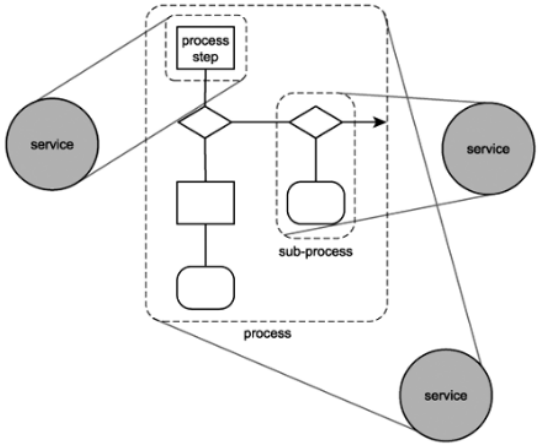


Figura 2.1. Logica incapsulata dai servizi (vedi [1])

Descrizione del servizio

Nella SOA, i servizi vengono normalmente usati da altri servizi o da programmi. Va quindi definita la modalità con cui i servizi si relazionano.

Affinché un servizio possa interagire con altri servizi, deve sapere che essi esistono e quali funzionalità offrono.

Queste informazioni si ottengono tramite la *descrizione del servizio*, che nella sua forma più semplice comprende il nome del servizio, i dati che richiede e quelli che fornisce. Il modello di relazione che si ottiene in questo modo è definito come "*loosely coupled*" ("disaccoppiato").

Secondo questo modello, il servizio A può comunicare con il servizio B perché conosce la sua descrizione di servizio. Non ci sono però legami diretti tra i due servizi, che possono esistere in modo indipendente l'uno dall'altro.

Messaggi

Oltre alla descrizione di servizio, per comunicare secondo un modello "loosely coupled", c'è bisogno di un apposito framework di comunicazione. Tale framework è costituito dal *messaging*.

Secondo questo framework, quando un servizio ha inviato un messaggio, perde il controllo sul messaggio stesso. Per questo motivo il messaggio deve costituire un'*"unità autonoma e indipendente di comunicazione"*.

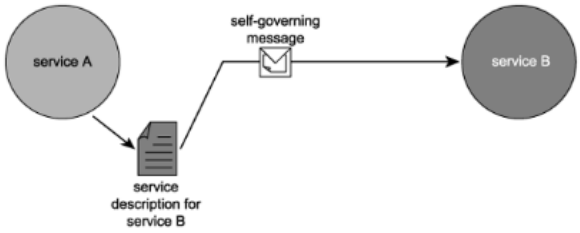


Figura 2.2. Comunicazione tra servizi tramite messaggi (vedi [1])

La comunicazione tra i servizi, implementata secondo la specifica SOAP, viene trattata più in dettaglio nella parte finale del capitolo.

2.3.1 Proprietà di servizi, descrizioni di servizio e messaggi

I concetti introdotti finora non sono nuovi, e appartengono ad architetture distribuite preesistenti che supportano il messaging e la separazione tra interfaccia e logica di elaborazione. Quello che distingue la SOA dalle altre architetture, sono le proprietà di servizi, descrizioni di servizio e messaggi, elencate di seguito:

- *Loose coupling* − il servizio mantiene una relazione con gli altri servizi che minimizza le dipendenze e richiede solo che ciascuno abbia consapevolezza degli altri (cioè sappia che essi esistono e quali funzionalità offrono).
- *Contratto di servizio* − i servizi aderiscono ad un convenzione di comunicazione comune, come definito dalle descrizione del servizio.
- *Autonomia* − i servizi hanno il pieno controllo della logica che incapsulano.
- *Astrazione* − fatta eccezione per quanto descritto nel contratto di servizio, i servizi incapsulano e nascondono logica al mondo esterno.

- *Riusabilità* − i servizi sono progettati in modo da incentivare il riuso di servizi già implementati.
- *Componibilità* − i servizi possono essere coordinati e composti per formare altri servizi con uno scope più ampio.
- *Statelessness* − il protocollo di comunicazione dei servizi è stateless, cioè tratta ogni richiesta come una transazione indipendente, scorrelata con qualsiasi richiesta precedente.
- *Reperibilità* − i servizi sono progettati per essere trovati da altri servizi tramite meccanismi di discovery.

2.4 Piattaforme tecnologiche per soluzioni orientate ai servizi

Una volta definiti gli elementi fondamentali e stabiliti i principi di progettazione di tali elementi, è necessario disporre di una piattaforma tecnologica dove implementare questi elementi per realizzare nella pratica soluzioni orientate ai servizi.

Nonostante la SOA sia uno stile architetturale agnostico rispetto alla tecnologia impiegata per realizzarlo, la tecnologia dei Web services interpreta meglio di qualsiasi altra i principi alla base della service-orientation descritti nel paragrafo precedente e mette a disposizione una piattaforma tecnologica in grado di realizzarli pienamente.

Pertanto possiamo dire che *la SOA contemporanea rappresenta un'architettura che promuove l'orientamento ai servizi attraverso l'uso di Web services.*

Tutte le moderne piattaforme per realizzare soluzioni service-oriented basate su Web services sono fondate sui principi elencati in precedenza, ma si sono sviluppate nel tempo beneficiando delle continue evoluzioni delle specifiche dei Web services.

Uno dei principali obiettivi dell'evoluzione delle moderne piattaforme di Service-Oriented Architecture basate su Web services è stato quello di garantire funzionalità di livello enterprise con lo stesso grado di sicurezza e affidabilità delle architetture tradizionali e consolidate.

Nonostante le differenti implementazioni, si è quindi delineato un *insieme di caratteristiche comuni* alle moderne piattaforme per realizzare soluzioni service-oriented basate su Web services, le principali delle quali verranno descritte nei paragrafi seguenti.

Qualità del servizio (Quality of Service - QoS)

Per garantire adeguati livelli di qualità del servizio sono stati implementati i seguenti requisiti:

- *Sicurezza* − capacità di svolgere compiti in modo sicuro proteggendo il contenuto del messaggio, così come l'accesso ai singoli servizi.

- *Transazionalità* − protezione dell'integrità e della consistenza di specifiche attività di business, con la garanzia che se dovesse fallire il task, viene gestita l'eccezione.
- *Affidabilità* − capacità di svolgere compiti in modo affidabile, in modo che sia garantita la consegna del messaggio o, in caso la consegna non vada a buon fine, la notifica della fallita consegna.
- *Performance* − capacità di garantire che l'overhead imposto dall'elaborazione dei messaggi SOAP e del contenuto XML non faccia fallire l'esecuzione di un compito

Le estensioni WS-* forniscono le indicazioni per implementare piattaforme che soddisfino i requisiti relativi alla qualità del servizio.

Autonomia

Il principio dell'autonomia, definito in precedenza limitatamente ai servizi ("*i servizi hanno il pieno controllo della logica che incapsulano*"), viene ora esteso alle diverse piattaforme, dove applicazioni costituite da servizi autonomi possono essere a loro volta viste come servizi compositi, autonomi e indipendenti all'interno di un ambiente integrato di soluzioni service-oriented.

Il livello di autonomia così ottenuto può superare i limiti imposti da soluzioni tecnologiche proprietarie, consentendone l'interoperabilità.

Standard aperti

Una delle caratteristiche più significative dei Web services è che lo scambio dati è governato da standard aperti. Il messaggio è scambiato tra un Web service l'altro utilizzando un insieme di protocolli globalmente accettati e standardizzati.

Il messaggio stesso è standardizzato, sia nel formato, sia per come è rappresentato il suo *payload* [3].

L'utilizzo di standard come SOAP, WSDL, XML e XML Schema, descritti nella parte finale del capitolo, rende possibile il principio secondo cui i servizi, per comunicare tra loro, devono conoscere solamente la descrizione degli altri servizi.

L'uso di un modello di messaging aperto e standardizzato elimina la necessità di accordarsi di volta in volta sulla logica di comunicazione e di mapping dei dati, come si fa in una modalità di integrazione punto-punto, e supporta il paradigma del disaccoppiamento.

Le moderne piattaforme SOA rafforzano questo modello di comunicazione aperto e riducono la dipendenza dalle specifiche soluzioni dei diversi vendor. Questo modello

[3] con il termine "payload" si intende in genere ciò che viene trasportato da un mezzo o servizio di trasporto. Il termine viene utilizzato per analogia anche nell'informatica per indicare la parte di un flusso di dati che rappresenta il contenuto informativo di un messaggio.

consente quindi di limitare il più possibile il *"vendor lock-in"* [4],favorire l'eterogeneità delle soluzioni tecnologiche e ridurre i vincoli imposti dai diversi produttori di tecnologie.

In questo modo anche se si utilizza un ambiente di sviluppo proprietario, a patto che questo supporti lo standard Web services, può essere realizzata un'interfaccia di servizio non proprietaria, aprendo all'interoperabilità con altre applicazioni che supportano lo standard Web services (p. es. la comunicazione tra soluzioni basate su Microsoft .Net Framework e soluzioni Java JEE avviene in modo ormai consolidato via Web services).

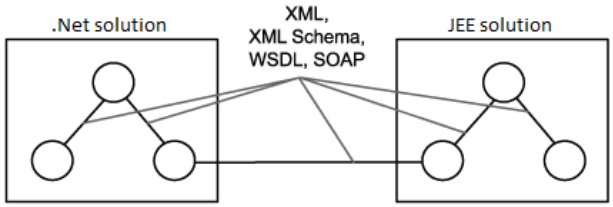


Figura 2.3. Sistemi eterogenei che comunicano attraverso protocolli standard (vedi [1])

Service registry e discovery dei servizi

Le piattaforme SOA prevedono l'impiego di un registro dei servizi (*service registry*) con lo scopo di raccogliere le informazioni sui servizi pubblicati, in modo che possano essere ritrovati e richiamati da altri servizi. Il service registry è pertanto il luogo più idoneo in cui gestire la descrizione del servizio.

In Provincia di Bolzano il service registry è stato realizzato ricorrendo ad un directory LDAP (Microsoft ADAM).

Interoperabilità

Le caratteristiche sopra elencate, relative all'uso di standard aperti, all'indipendenza da soluzioni proprietarie e alla discovery dei servizi costituiscono la nativa e intrinseca *interoperabilità* delle soluzioni SOA.

È ormai un dato di fatto che gran parte delle richieste del business vanno nella direzione di integrare applicazioni, in modo da automatizzare operazioni complesse o ripetitive a vantaggio dell'efficienza e riducendo le possibilità di errori da parte degli utenti.

Se si progetta un'applicazione in modo orientato ai servizi, la si rende da subito interoperabile, e quindi già pronta a soddisfare eventuali requisiti di integrazione futuri, non prevedibili al momento dell'analisi dell'applicazione.

---

[4] con "vendor lock-in" si definisce una pratica di molti produttori di software, che rende il cliente dipendente da uno specifico prodotto. I clienti non riescono a sostituire il prodotto in uso con quello di un altro vendor, senza incorrere in ingenti costi dovuti al cambiamento di tecnologia.

Un'altra importante caratteristica delle piattaforme SOA è la capacità di incapsulare la logica delle applicazioni preesistenti nelle aziende, e di esporla attraverso il protocollo di comunicazione aperto e standardizzato dei Web services.

Per fare questo, molte piattaforme SOA si servono di *adapter* [5] o *wrapper*, cioè di connettori che sono in grado dal un lato di interagire con i protocolli proprietari delle piattaforme legacy più diffuse e dall'altro di trasformare la comunicazione attraverso l'uso di Web services, che possono essere richiamati da altre applicazioni.

La SOA consente quindi di mantenere gli investimenti fatti, evitando che le tecnologie più datate debbano essere sostituite, e facendo in modo che possano esporre i loro servizi in un nuovo ambiente integrato, standardizzato, omogeneo e interoperabile.

Supporto ai processi di business

Se si suddivide la logica di business in modo opportuno e la si incapsula all'interno dei servizi, è possibile fare in modo che i servizi contribuiscano all'esecuzione di un processo di business.

In questo caso ci si può avvalere di strumenti di *Business Process Management* (BPM). Questi strumenti consentono di eseguire processi di business, solitamente rappresentati con linguaggi di modellazione come BPMN (*Business Process Model and Notation*).

I sistemi di BPM si occupano quindi di eseguire i vari passi del processo di business secondo uno schema predefinito, coinvolgendo i servizi quando il processo lo prevede.

2.4.1 Astrazione e disaccoppiamento

Si è sottolineato che in un sistema "loosely coupled", ciascun servizio mantiene con gli altri una relazione che minimizza le dipendenze. Ogni servizio conosce degli altri solo quanto è riportato nella descrizione del servizio.

I servizi pertanto incapsulano e nascondono la propria logica al mondo esterno. È possibile portare questo concetto a livello più alto, separando ed astraendo il dominio di business da quello applicativo all'interno dell'azienda.

Abbiamo già accennato nel paragrafo precedente che è possibile definire uno strato di servizi di business, che incapsulino le specifiche attività dei processi dell'azienda: allo stesso modo è possibile definire uno strato di servizi di tipo applicativo/tecnologico, in modo da nascondere ed incapsulare logica applicativa e funzionalità infrastrutturali (p. es. autenticazione, accesso ai dati, persistenza, ecc.).

---

[5] con il termine "adapter" o anche "wrapper", ci si riferisce ad un design pattern relativo alla programmazione orientata agli oggetti. Il fine dell'adapter è di fornire una soluzione astratta al problema dell'interoperabilità tra interfacce differenti, non perfettamente compatibili con quanto richiesto da applicazioni già esistenti.

Il beneficio che si ottiene è che ciascun dominio evolve in modo indipendente dall'altro, rendendo più semplice la gestione dei cambiamenti *a tutto vantaggio dell'agilità dell'organizzazione*:

- un cambiamento nella logica di business non dovrebbe avere un grande impatto sulle applicazioni che automatizzano quel business;
- allo stesso modo un cambiamento a livello dell'infrastruttura tecnologica non dovrebbe influire in modo troppo pesante sulla logica di business automatizzata da questa tecnologia.

L'agilità organizzativa è forse il maggiore vantaggio che si può ottenere con la SOA.

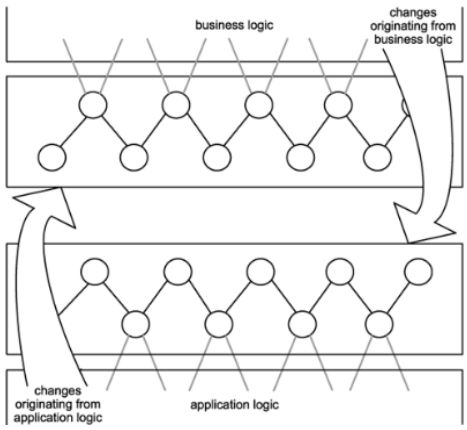


Figura 2.4. Una relazione "loosely coupled" tra lo strato di logica di business e la logica applicativa consente a ciascuno di rispondere più velocemente a cambiamenti nell'altro (vedi [1])

Non ci sono limiti alla dimensione di un servizio. Una singola applicazione service-oriented può essere vista nella sua interezza come un singolo servizio.

Una SOA consiste di servizi contenuti in servizi, a loro volta contenuti in servizi, e così via, fino al punto che una soluzione basata su SOA è essa stessa uno dei molti servizi all'interno di una *Service-Oriented Enterprise* [6] (SOE).

Concludendo, la SOA può stabilire un'astrazione di logica di business e tecnologia, la quale può introdurre cambiamenti alla modellazione dei processi di business e dell'architettura tecnica, con il risultato di un disaccoppiamento tra questi modelli. Questi cambiamenti favoriscono l'orientamento ai servizi a supporto di una service-oriented enterprise. Il vantaggio consiste nella maggiore flessibilità nel rispondere ai costanti cambiamenti imposti dal mercato e della normativa.

[6] una service-oriented enterprise è una organizzazione in cui tutti i processi di business sono composti da − ed esistono come − servizi.

2.5 XML, SOAP e Web services

Il linguaggio XML (*Extensible Markup Language*) è stato creato negli anni '90 dal W3C[7], come derivazione del linguaggio SGML (*Standard Generalized Markup Language*), che esisteva fino dagli anni '60.

XML è un linguaggio a marcatori che definisce un insieme di regole per codificare documenti in un formato comprensibile sia da computer che da esseri umani. Nonostante l'XML fosse inizialmente focalizzato sulla rappresentazione e trasmissione di documenti attraverso internet, oggi è ampiamente utilizzato per la rappresentazione in modo standardizzato di qualsiasi struttura dati.

Questa sua caratteristica si è rivelata fondamentale per consentire l'interoperabilità tra piattaforme diverse, ed è alla base della tecnologia dei Web services.

Nel 2000 è stato sottoposto al W3C il protocollo SOAP (Simple Object Access Protocol), una specifica nata originariamente per unificare e standardizzare comunicazioni proprietarie di tipo RPC (*Remote Procedure Call*). L'idea alla base della specifica era quella che, per trasmettere i dati tra i sistemi attraverso internet, questi venissero serializzati in XML, trasportati e poi deserializzati nel formato nativo.

Con l'unione di questi elementi, ovvero il linguaggio standard e aperto XML per la rappresentazione di strutture dati, e il protocollo SOAP per la trasmissione dei dati in formato XML, si è creato un framework di comunicazione standardizzato, in cui sistemi diversi possono interagire tra loro richiamando i Web services con appositi messaggi: tali messaggi sono inclusi in una "busta" (tipicamente SOAP), e sono formattati secondo lo standard XML, incapsulati e inviati sulla rete tramite i protocolli del Web (solitamente HTTP).

La parte più importante del Web service è la sua interfaccia pubblica. Si tratta di un blocco di informazioni che implementa la descrizione del servizio e rende possibile la sua invocazione: il WSDL (*Web Service Description Language*). La prima specifica del WSDL è stata sottoposta al W3C nel 2001.

Un'altra componente essenziale è il registro dei servizi. Una sua prima implementazione standard è stata inizialmente proposta con la specifica UDDI (*Universal Description Discovery and Integration*), che consentiva la creazione di registri standardizzati per la pubblicazione delle descrizioni dei servizi sia all'interno dell'azienda che in internet. L'UDDI è stato progettato per essere interrogato dai *service requestor* e per fornire il collegamento ai documenti WSDL che contengono

[7] Fondato originariamente da Tim Berners-Lee nel 1994, il W3C (World Wide Web Consortium) è un'organizzazione che si occupa di standard, ed ha avuto un ruolo fondamentale nella diffusione del World Wide Web. Nell'ambito del W3C sono stati proposti, discussi, definiti e ufficializzati oltre 50 standard industriali di vasta portata, tra cui p. es. HTTP, URI, URL, HTML, XML, XML Schema, XSLT, SOAP.

le specifiche necessarie per interagire con i servizi desiderati. Al contrario di altri standard come WSDL e SOAP, UDDI però non è stato accettato dal mercato.

2.5.1 Gli elementi principali dei Web services

Nella figura seguente sono rappresentati gli elementi principali che costituiscono la tecnologia Web services:

- service requestor
- service provider
- service registry

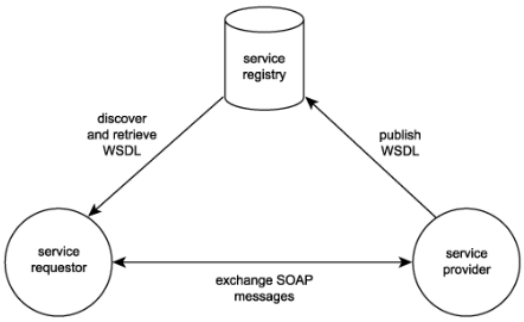


Figura 2.5. Elementi principali della tecnologia Web services (vedi [1])

Il WSDL è pubblicato dal *service provider* e descrive il servizio, il protocollo SOAP specifica il formato da utilizzare nella trasmissione dei dati tra il servizio e il suo richiedente e il registro dei servizi fornisce al *service requestor* il collegamento al WSDL del *service provider*.

Service provider

Un Web service assume il ruolo di *service provider* quando:

– è invocato da un *service requestor*;
– pubblica una descrizione che contiene le informazioni sulle sue caratteristiche e il suo comportamento.

Il ruolo del *service provider* è sinonimo del ruolo del *server* nella classica architettura *client-server*.

Service requestor

È un qualsiasi programma capace di inviare un messaggio di *request* che può essere compreso da un *service provider*. Un Web service di norma è *service provider*, ma nel momento in cui esso stesso chiama un altro servizio agisce come *service requestor*.

Nella figura seguente è illustrato il funzionamento della comunicazione tra Web services.

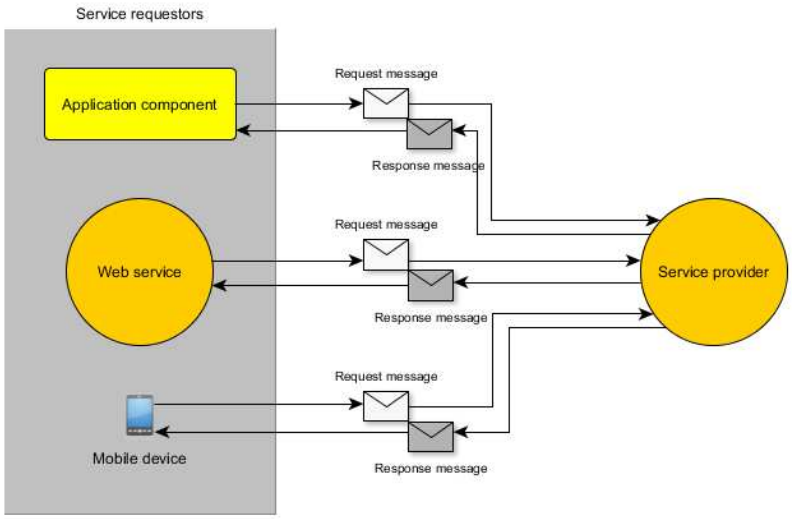


Figura 2.6. Comunicazione tra service requestor e service provider tramite messaggi

Service description (WSDL)

Il WSDL implementa la proprietà di *loose-coupling* enunciata in precedenza, e contiene la descrizione del servizio. Al *service requestor* non serve sapere come è implementato internamente il *service provider*, per interagire è sufficiente conoscere la sua descrizione, ovvero il suo WSDL. Il WSDL definisce in modo formale l'interfaccia del *service provider*, così che ciascun *service requestor* sappia esattamente come strutturare il messaggio di *request*. Inoltre il WSDL indica la locazione fisica (*address*) dove il servizio è raggiungibile.

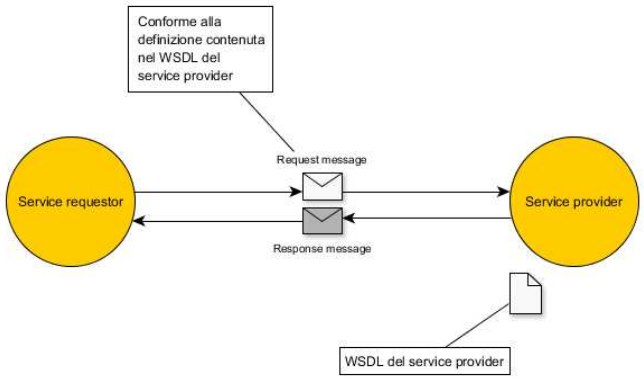


Figura 2.7. Service request conforme alle indicazioni contenute nel WSDL del provider

Come indicato nella figura seguente, la definizione WSDL è suddivisa in due sezioni.

- *Descrizione astratta* − stabilisce le caratteristiche dell'interfaccia del Web service senza riferimenti alla tecnologia impiegata per abilitare il servizio alla

trasmissione di messaggi. Separando queste informazioni si mantiene l'integrità della descrizione del servizio indipendentemente dai cambiamenti della piattaforma tecnologia sottostante. Di seguito sono descritte le tre sezioni che costituiscono la descrizione astratta:

- *interface*[8] *(portType)*: fornisce una vista di alto livello dell'interfaccia del servizio, classificando i messaggi che il servizio è in grado di elaborare in gruppi di funzioni note come *operations*;
- *operation*: rappresenta un'azione specifica eseguita dal servizio. Un'*operation* del servizio è paragonabile a un metodo pubblico utilizzato dai componenti nelle tradizionali applicazioni distribuite. Come i metodi dei componenti, anche le *operations* hanno parametri di input e di output;
- *message*: dato che i Web services si affidano esclusivamente alla comunicazione basata su messaggi, i parametri sono rappresentati come messaggi. Pertanto, l'*operation* consiste di un insieme di messaggi di input e output.

- *Descrizione concreta* − poiché l'esecuzione della logica applicativa del servizio comporta sempre la comunicazione, l'interfaccia astratta del Web service deve essere collegata ad un protocollo di trasporto. Questa connessione è definita nella sezione concreta della definizione WSDL, che si compone di tre parti correlate:
  - *binding:* rappresenta la tecnologia di trasporto che il servizio può utilizzare per comunicare (SOAP è la più comune forma di binding). Il binding si può specificare per l'intera interfaccia o per singole *operations*;
  - *endpoint*[9] *(port)*: definisce l'indirizzo fisico al quale il servizio può essere richiamato. Tipicamente è un semplice URL HTTP;
  - *service*: contiene una serie di funzioni di sistema che sono esposte ai protocolli Web-based.

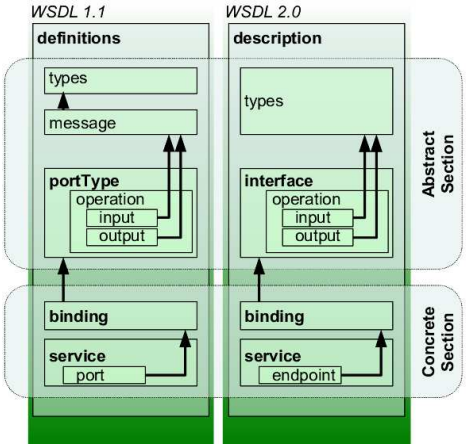


Figura 2.8. Confronto tra le strutture WSDL 1.1 e 2.0 (Wikipedia)

[8] Nella versione 2.0 della specifica WSDL, che è diventata *W3C recommendation* nel 2007, il termine "interface" ha sostituito il termine "portType".

[9] Nella versione 2.0 della specifica WSDL il termine "endpoint" ha sostituito il termine "port".

A titolo di esempio si riportano di seguito alcuni estratti del WSDL del servizio del protocollo informatico della Provincia di Bolzano, relativi alle sezioni viste sopra.

Esempio di *portType* con l'elenco di *operations*, ciascuna con i parametri di input e output:

```
<wsdl:portType name="GestioneDocumenti">
 - <wsdl:operation name="creaProtocollo">
      <wsdl:input name="creaProtocollo" message="tns:creaProtocollo"> </wsdl:input>
      <wsdl:output name="creaProtocolloResponse" message="tns:creaProtocolloResponse"> </wsdl:output>
      <wsdl:fault name="OggettoDocumentoVuoto" message="tns:OggettoDocumentoVuoto"> </wsdl:fault>
      <wsdl:fault name="CodiceFascicoloNonEsistente" message="tns:CodiceFascicoloNonEsistente"> </wsdl:fault>
      <wsdl:fault name="RegistroNonEsistente" message="tns:RegistroNonEsistente"> </wsdl:fault>
      <wsdl:fault name="TipoProtocolloNonValido" message="tns:TipoProtocolloNonValido"> </wsdl:fault>
      <wsdl:fault name="MittenteDestinatarioIncompleto" message="tns:MittenteDestinatarioIncompleto"> </wsdl:fault>
      <wsdl:fault name="AssegnatarioCompetenzaMancante" message="tns:AssegnatarioCompetenzaMancante"> </wsdl:fault>
      <wsdl:fault name="EProcsResponseError" message="tns:EProcsResponseError"> </wsdl:fault>
      <wsdl:fault name="MittenteNonValido" message="tns:MittenteNonValido"> </wsdl:fault>
      <wsdl:fault name="CodiceTitolarioNonEsistente" message="tns:CodiceTitolarioNonEsistente"> </wsdl:fault>
      <wsdl:fault name="CodiceTitolarioObbligatorio" message="tns:CodiceTitolarioObbligatorio"> </wsdl:fault>
      <wsdl:fault name="NumeroProtocolloNonEsistente" message="tns:NumeroProtocolloNonEsistente"> </wsdl:fault>
      <wsdl:fault name="TipoSpedizioneNonEsistente" message="tns:TipoSpedizioneNonEsistente"> </wsdl:fault>
      <wsdl:fault name="DataProtocollazioneNonValida" message="tns:DataProtocollazioneNonValida"> </wsdl:fault>
      <wsdl:fault name="DestinatarioCompetenzaMancante" message="tns:DestinatarioCompetenzaMancante"> </wsdl:fault>
  </wsdl:operation>
```

come si vede, i parametri di input e output sono rappresentati come messaggi.

```
 - <wsdl:message name="creaProtocolloResponse">
      <wsdl:part name="parameters" element="tns:creaProtocolloResponse"> </wsdl:part>
  </wsdl:message>
```

Nell'esempio di *binding* che segue, si vede che è specificato "SOAP document-style" come tecnologia di trasporto, che si applica a tutte le operations.

```
<wsdl:binding name="GestioneDocumentiServiceSoapBinding" type="tns:GestioneDocumenti">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  + <wsdl:operation name="rimuoviDocumentoAllegatoDaProtocollo">
  + <wsdl:operation name="listaAllegati">
  + <wsdl:operation name="chiudiFascicolo">
  + <wsdl:operation name="creaProtocolloDifferito">
  + <wsdl:operation name="downloadDocumentoPrincipale">
  + <wsdl:operation name="associaDocumentoAllegatoAProtocollo">
  + <wsdl:operation name="downloadDocumentoAllegato">
  + <wsdl:operation name="dettaglioProtocollo">
  + <wsdl:operation name="listaOrganizzazioni">
  + <wsdl:operation name="listaProtocolli">
  + <wsdl:operation name="listaTitolari">
  + <wsdl:operation name="creaProtocollo">
  + <wsdl:operation name="associaDocumentoPrincipaleAProtocollo">
  + <wsdl:operation name="annullaProtocollo">
  + <wsdl:operation name="listaFascicoli">
  + <wsdl:operation name="togliProtocolloDaFascicolo">
  + <wsdl:operation name="listaTipoSpedizione">
  + <wsdl:operation name="dettagliOrganizzazione">
  + <wsdl:operation name="modificaProtocollo">
  + <wsdl:operation name="associaProtocolloAFascicolo">
  + <wsdl:operation name="creaFascicolo">
</wsdl:binding>
```

Infine si riporta un esempio di specifica di *service* e *port*:

```
<wsdl:service name="GestioneDocumentiService">
  - <wsdl:port name="GestioneDocumentiPort" binding="tns:GestioneDocumentiServiceSoapBinding">
       <soap:address location="http://egovernment.prov.bz:80/FascicoloInformatico/GestioneDocumenti"/>
    </wsdl:port>
</wsdl:service>
```

2.5.2 Struttura dei messaggi SOAP

Dato che la comunicazione tra servizi è interamente basata su messaggi, il framework di *messaging* scelto deve essere standardizzato, in modo che tutti i servizi, indipendentemente dalla piattaforma utilizzata, usino lo stesso formato e

protocollo di trasporto. L'obiettivo principale della specifica SOAP[10] è quello di definire uno formato di *messagging* standard.

Di seguito viene riportata la struttura di base di un messaggio SOAP:

- *Envelope* − ogni messaggio SOAP è confezionato in un contenitore noto come busta (*envelope*). La busta contiene tutte le parti del messaggio;
- *Header* − ogni messaggio può contenere un'intestazione (*header*), un'area dedicata ad ospitare meta-informazioni. Nonostante sia un'area opzionale, raramente viene omessa, in quanto nella maggior parte delle soluzioni orientate ai servizi questa sezione riveste grande importanza. La sua importanza risiede nell'uso di *header blocks* attraverso i quali possono essere implementate molte estensioni WS-*. Per esempio la specifica WS-Security adottata anche dalla Provincia di Bolzano, prevede che il *token Kerberos* o lo *username token* per l'autenticazione siano inseriti nel *header* del messaggio SOAP.
- *Body* − il contenuto effettivo del messaggio, che consiste tipicamente in dati XML formattati, è custodito nel corpo (*body*) del messaggio. Il contenuto del corpo del messaggio è spesso indicati con il termine"payload".

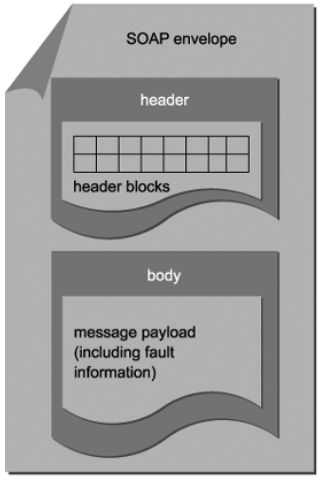


Figura 2.9. Struttura di base di un messaggio SOAP (vedi [1])

[10] A partire dalla versione 1.2 della specifica SOAP, la parola "SOAP" non è più l'acronimo di "Simple Object Access Protocol", ma è considerato un termine a sé stante.

3 SOA nella Provincia di Bolzano

In questo capitolo viene descritta l'implementazione della SOA in Provincia di Bolzano. Vengono illustrati i motivi per cui è stato necessario adottare una nuova modalità di sviluppare le applicazioni, e viene descritto nel dettaglio il framework COSM, che costituisce un approccio industriale alla gestione dell'informatica e allo sviluppo del software. Sono illustrate in questo capitolo le caratteristiche del framework e la modalità di sviluppo delle applicazioni basate sul modello COSM. Particolare attenzione è rivolta al concetto di Business Component, fondamentale nella metodologia COSM. A supporto dei concetti teorici sono riportati alcuni esempi, tratti da progetti implementati nell'amministrazione provinciale.

3.1 La situazione dell'IT antecedente all'e-government

Negli anni intorno al 2007 la Giunta Provinciale della Provincia di Bolzano ha espresso la volontà di introdurre l'e-government nell'amministrazione provinciale, in modo da rendere più moderne ed efficienti le relazioni tra enti, cittadini e imprese.

In quel periodo le architetture in uso all'interno della Provincia di Bolzano erano esclusivamente di tipo client-server (basate principalmente su Oracle Forms, Microsoft Visual Basic, Borland Delphi) o web (Java JSP o Microsoft ASP/ASP.Net), ma in tutti i casi si trattava di architetture *data-centric*, ovvero architetture fortemente incentrate sulle banche dati, con le seguenti caratteristiche:

- accesso diretto alla banca dati con protocolli nativi o driver come p. es. ODBC o JDBC, senza strati di disaccoppiamento, creando una stretta dipendenza tra le applicazioni stesse e la struttura delle banche dati sottostanti;
- implementazione della logica di business nelle maschere delle applicazioni e nelle *stored procedures* eseguite dalla banca dati, invece che astrarre la logica e implementarla a livello dei servizi, come consentono di fare le architetture *multi-tier*;

– uso di banche dati condivise in lettura e scrittura tra più applicazioni per realizzare l'integrazione e il passaggio dei dati tra le applicazioni, invece che realizzare la comunicazione tra servizi attraverso messaggi.

L'architettura di queste applicazioni non prevedeva la possibilità che cittadini ed imprese potessero avviare procedimenti o accedere ai propri dati e documenti in maniera autonoma, senza doversi recare personalmente agli sportelli degli uffici.

I settori IT delle amministrazioni locali hanno pertanto dovuto dotarsi di nuove architetture, idonee a realizzare applicazioni orientate all'e-government.

3.2 L'e-government nella Provincia di Bolzano

La spinta innovativa portata dall'e-government ha indotto il settore IT dell'amministrazione provinciale a rivedere radicalmente il modo con cui realizzare le applicazioni e a dotarsi di uno stile architetturale specifico per la realizzazione di applicazioni predisposte all'e-government.

Tali applicazioni dovevano garantire i seguenti requisiti:

- produrre e gestire dati in modo che possano essere usati da altre applicazioni;
- supportare nativamente un modello di sicurezza e autenticazione comune e standardizzato e funzionalità specifiche, quali per esempio la firma digitale e la protocollazione automatica di documenti;
- avere un'interfaccia utente coerente con un modello standard, comune a varie applicazioni (sia a livello grafico che funzionale);
- gestire processi trasversali a più unità organizzative o intere organizzazioni.

A questo cambiamento radicale a livello tecnologico è stata associata un'evoluzione organizzativa nella direzione dell'*industrializzazione dell'informatica*. Questo percorso ha implicato una progressiva razionalizzazione del modo di sviluppare, integrare, ed acquistare le applicazioni, con l'obiettivo di ridurre i costi informatici, ma anche razionalizzare la gestione dell'informatica e migliorare il servizio al cittadino.

3.3 Obiettivi e principi dell'architettura e-government

Per gettare le basi dell'e-government e predisporre l'architettura IT di riferimento, la Provincia di Bolzano ha adottato i principi dell'approccio *COSM*™ (*Component Oriented Software Manufacturing*) della ditta Herzum Software.

Con l'introduzione di questa metodologia, inizialmente solo nell'ambito e-government, ma estesa in seguito all'intera IT provinciale, la ripartizione Informatica provinciale si è data i seguenti obiettivi:

- ridurre significativamente i costi e i tempi di sviluppo del software

- normalizzare il portafoglio applicativo, riducendo le ridondanze di tecnologie e funzionalità, e introducendo significativi livelli di riuso
- standardizzare l'uso delle tecnologie, per facilitare lo sviluppo, manutenzione, ed evoluzione del software

e ha stabilito i seguenti principi architetturali:

Principio	Descrizione
SOA	Sviluppo delle interfacce delle applicazioni e componenti secondo lo stile architetturale Service-Oriented
Component based	Modularizzazione di applicazioni secondo componenti
Stratificazione (layering funzionale)	Identificazione di chiari strati funzionali, con gestione architetturale delle dipendenze
Applicazioni orientate ai processi aziendali	Allineamento tra la tecnologia e le necessità aziendali
Multi-tier: "user", "workspace" "enterprise"	Chiara identificazione delle responsabilità di ogni tier (p. es. interfaccia utente, tier applicativo, tier di persistenza)
Abbandono dell'approccio *data centric*	Almeno per le applicazioni più importanti, è fortemente sconsigliato l'uso dell'approccio data centric per lo sviluppo di codice applicativo
Separazione aspetti funzionali/tecnologici	Adozione di stili di programmazione al fine di separare in modo chiaro gli aspetti funzionali da quelli tecnologici
Open source	Scelta di tecnologie, ambienti, e prodotti open source quando possibile e vantaggioso
Isolamento/indipendenza da ingredienti tecnologici	Adozione di framework che permettano di massimizzare l'indipendenza dagli ingredienti tecnologici

Tabella 3.1. Principi dell'architettura e-government della Provincia di Bolzano

3.4 Il modello COSM™ e lo sviluppo a componenti

COSM (Component Oriented Software Manufacturing) è un approccio industriale alla gestione dell'informatica e allo sviluppo software che consente di rispettare i principi sopra elencati, ed è stato utilizzato per il rinnovamento tecnologico della Provincia di Bolzano.

Questo approccio non è stato adottato in modo passivo dall'IT dell'amministrazione provinciale, ma è stato progressivamente modificato ed adattato al contesto e alle esigenze specifiche della Provincia di Bolzano. Sulla base di questa metodologia sono state gettate le fondamenta dell'architettura di riferimento per le applicazioni e-government e di back-office, fortemente orientate ai servizi.

Alla base dell'approccio COSM c'è lo sviluppo a componenti.

3.4.1 Caratteristiche dei componenti

Per *componente* si intende un blocco di software autonomo ed indipendente, con un'interfaccia ben definita, che concorre a costituire una struttura più complessa.

Lo sviluppo a componenti ha i seguenti obiettivi:

- gestire la complessità dei sistemi informativi;
- abbassare i costi di realizzazione dei sistemi informativi;
- fornire l'agilità e la flessibilità necessarie per lo sviluppo software.

Per raggiungere queste finalità, si aspira a trasformare il software in una *commodity*[1], che può essere integrata, assemblata e composta con facilità e senza dover far ricorso a risorse specializzate e soluzioni infrastrutturali costose. Un componente deve pertanto essere facilmente combinabile con altri componenti per fornire funzionalità più ampie e di utilità per il business.

Inoltre è necessaria la disponibilità di un'infrastruttura (detta *component socket*) dove i componenti possono essere inseriti ed eseguiti in modo facile e immediato (p. es. come una spina elettrica in una presa di corrente).

I componenti possono essere utilizzati direttamente dagli utenti, tramite interfacce grafiche, oppure da altri sistemi/componenti. Ciò significa che ciascun componente dovrà comportarsi in maniera differente a seconda di chi lo utilizza.

Si possono individuare alcune caratteristiche essenziali di un componente:

1. un componente è un costrutto software autonomo e indipendente, che realizza una funzionalità specifica ben definita e può essere rilasciato in autonomia (senza dipendenze dagli altri componenti) all'interno di uno specifico ambiente di esecuzione (*component socket*);

[1] Commodity è un termine inglese che indica un prodotto o servizio offerto senza differenze qualitative sul mercato. Ciò significa che il prodotto è lo stesso indipendentemente da chi lo produce. La *commodification* accade quando beni o servizi di un determinato mercato perdono la loro differenziazione. Spesso questo avviene quando c'è una tale diffusione della conoscenza da offrire con facilità e convenienza quel determinato prodotto o servizio. Alcuni esempi di commodification possono essere medicinali non più protetti da brevetto o microprocessori la cui tecnologia è diventata pubblica.

2. un component socket è uno strato di software che fornisce un ambiente di esecuzione ben definito all'interno di un'infrastruttura di supporto nella quale il componente viene eseguito (p. es. un application server);
3. un componente è progettato per la composizione e la collaborazione con altri componenti;
4. l'utilizzatore di un componente potrebbe non avere le capacità necessarie per costruirlo, ma solo per utilizzarlo. Questo comporta una separazione di ruoli, in cui sviluppatori specializzati creano i componenti, e sviluppatori funzionali li utilizzano per risolvere problemi specifici.

Siamo interessati ad una particolare tipologia di componenti, determinanti per ridurre drasticamente il costo dello sviluppo software di sistemi distribuiti su larga scala: gli *enterprise components*, cioè quei componenti che sono utilizzati per costruire sistemi che hanno impatto sull'intera azienda.

In aggiunta alle caratteristiche elencate precedentemente, questi particolari componenti hanno ulteriori proprietà specifiche:

a) sono progettati per gestire in modo nativo le problematiche tecniche tipiche dei sistemi distribuiti a livello enterprise (p. es. concorrenza, sicurezza, gestione delle transazioni, accesso ai dati, ecc);
b) hanno un'interfaccia che può essere richiamata attraverso la rete (*network-addressable interface*). Tali componenti sono pensati per essere facilmente usati e riusati a run-time da altri software per creare sistemi software completi;
c) hanno granularità medio - grande: una granularità troppo piccola aumenta la complessità di sviluppo e gestione di grandi applicazioni. Se si vuole dare un ordine di grandezza indicativo, ogni componente corrisponde ad un oggetto del dominio applicativo servito da 10-20 tabelle del database relazionale. Se il componente è implementato con un linguaggio Object Oriented, si può stimare che sia implementato con un numero di classi significative che varia da 5 a 200 classi, che implementano la logica e le regole di business;
d) rappresentano specifici concetti di business nel sistema informativo. Non è un requisito sempre rispettato, ma rappresenta un indicatore molto forte per individuare i tipi di componenti a cui siamo interessati.

3.4.2 Livello di granularità dei componenti

Come accennato alla lettera c) del paragrafo precedente, possiamo classificare le diverse tipologie di componenti in base al loro livello di granularità, da quelli più di dettaglio (più granulari) a quelli meno dettagliati/più di alto livello (meno granulari).

Procedendo da un livello di granularità maggiore ad uno minore possiamo individuare le seguenti tipologie di componenti:

- *Distributed Component* − è considerato il componente più granulare. È di norma costruito utilizzando un linguaggio Object Oriented. Può essere implementato p. es. tramite un Enterprise Java Bean (EJB).

- *Business Component* − è un componente che implementa un singolo autonomo concetto di business. Solitamente consiste di uno o più Distributed Components che cooperano per implementare e risolvere assieme i vari aspetti tecnologici e implementativi delle funzionalità richieste al business component. Il Business Component concettualmente è un singolo costrutto, ma può essere fisicamente suddiviso in più parti, rilasciate su macchine diverse.
- *Business Component System* − è un gruppo di Business Components che cooperano per fornire l'insieme completo di funzionalità richieste da una specifica necessità di business. In altre parole è un sistema interamente costituito da Business Components. Un esempio di Business Component System è un sistema di Gestione Fatture Fornitori — l'insieme di Business Components necessario per gestire le fatture dei fornitori.
- *System-Level Component* − Quando un sistema di Business Components è incapsulato in modo tale che il sistema nel suo complesso possa essere trattato come una black box, allora diventa esso stesso un componente a sé stante, con una minore granularità.

Di seguito è riportato uno schema indicante la tassonomia dei componenti, classificati in base al livello di granularità.

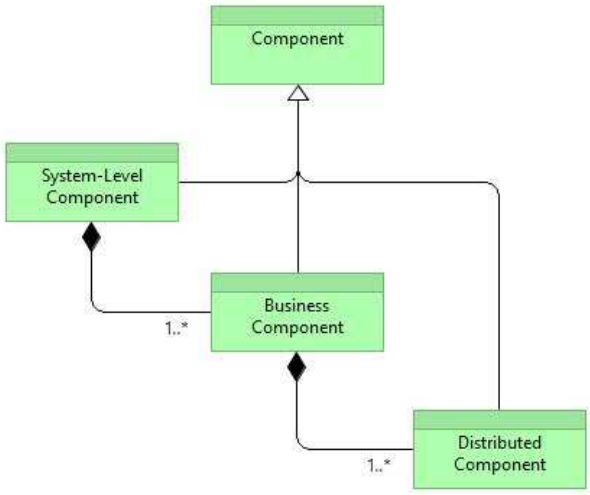


Figura 3.1. Tassonomia dei componenti (vedi [2])

A livello ancora più alto possiamo individuare una *federazione di System-Level Components*: si tratta di un insieme di System-Level Components che cooperano. In questo modo può essere di fatto realizzata l'interoperabilità tra sistemi eterogenei.

Nei paragrafi successivi le diverse categorie di componenti sono esposte in modo più dettagliato ed esaustivo.

3.4.3 Distributed Component

È il componente più di basso livello: si tratta di un artefatto software che ha le seguenti proprietà di implementazione:

- ha un'interfaccia ben definita sia a design-time che a run-time;
- può essere collegato in modo indipendente dagli altri componenti in un ambiente di run-time;
- può essere indirizzato attraverso la rete a run-time.

Si assume che per la costruzione di un Distributed Component vengano utilizzati linguaggi Object-Oriented, con i relativi patterns di architettura, design e implementazione.

Un componente distribuito è di norma composto da un certo numero di classi, come illustrato nell'esempio seguente.

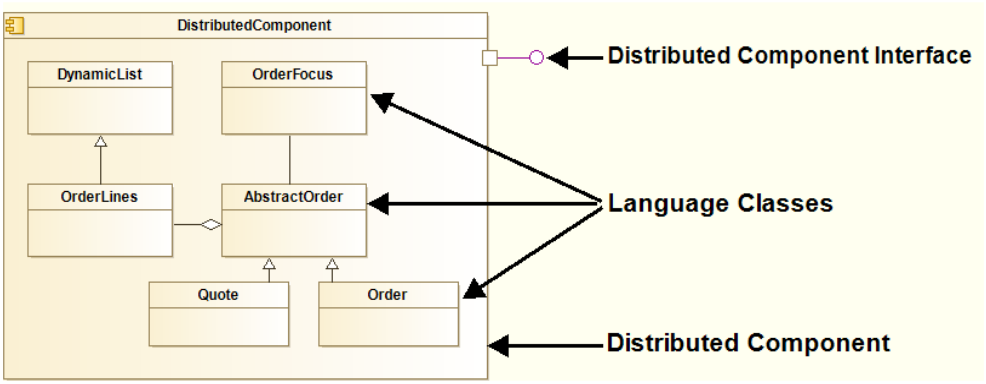


Figura 3.2. Componenti distribuiti e classi (vedi [2])

Solitamente il Distributed Component viene progettato dall'architetto funzionale[2].

Secondo la metodologia COSM l'architetto funzionale, che progetta un'applicazione, non si deve occupare di dettagli tecnici complessi come transazionalità, concorrenza, gestione degli eventi, localizzazione degli oggetti sulla rete, definizione delle specifiche del passaggio dei dati dal client al server, ecc.

---

[2] generalmente Functional Architect è considerato sinonimo di Business Analyst. Nella metodologia COSM i due ruoli invece sono distinti: l'obiettivo del Business Analyst è di produrre un modello del business che sta analizzando, con il fine di identificare le criticità e le opportunità di miglioramento. In questo caso vi è la necessità di approfondire molto l'analisi di business, per chiarire eventuali ambiguità o errate interpretazioni. Il modello di business che produce può non essere in alcun modo correlato con la necessità di implementare un software. Il Functional Architect, invece, durante la modellazione dei processi di business, ha come obiettivo la produzione di un software. L'obiettivo è quindi prevalentemente quello di capire il business per strutturare adeguatamente il sistema informativo. In questo caso l'architetto funzionale può raggiungere il suo scopo anche se il modello di business è poco approfondito, troppo astratto o ambiguo. Un livello maggiore di dettaglio sarà raggiunto nella fase di progettazione.

La definizione del Distributed Component astrae gli aspetti più tecnici, ed è di fatto indipendente dall'implementazione che ne verrà fatta, cioè questa definizione è valida qualsiasi sia la tecnologia di middleware o di esecuzione dei componenti utilizzata.

Ovviamente questa astrazione va poi supportata da un'infrastruttura tecnica che sia in grado di eseguire i componenti fornendo in modo nativo e integrato tutte le caratteristiche di scalabilità, affidabilità, sicurezza, federabilità necessarie a livello enterprise.

Come si vedrà nel seguito, la Provincia di Bolzano ha realizzato due infrastrutture con queste caratteristiche, una in ambito Java e l'altra in ambito Microsoft, utilizzando le funzionalità messe a disposizione dalla piattaforme JBoss e Microsoft .Net

3.4.4 Business Component

Il Business Component è il concetto principale attorno al quale è incentrato l'intero approccio COSM. Peter Herzum (President at Herzum Software), definisce il Business Component come segue:

si tratta dell'implementazione software di un concetto o di un processo di business autonomo e indipendente. Esso è costituito da tutti gli artefatti software necessari per rappresentare, implementare, e rilasciare un certo concetto di business come elemento autonomo e riutilizzabile, parte di un sistema informativo distribuito più ampio.

Nella definizione appena esposta si evidenziano tre aspetti distinti che caratterizzano il Business Component:

1) Rappresenta ed implementa un singolo ed autonomo concetto o processo di business

Un Business Component è adatto a rappresentare quei concetti di business che sono relativamente autonomi. Per esempio il concetto di "persona" oppure "dipendente" o "ditta" potrebbero essere buoni candidati ad essere rappresentati attraverso Business Components.

È implementato utilizzando le attuali tecnologie di sviluppo e componentizzazione del software, e pertanto, in base alla definizione di Distributed Component, un Business Component è una composizione di Distributed Components.

"Autonomo" non significa isolato. Come ciascun concetto di business è correlato agli altri (p. es. un *ordine* è fatto da un certo *cliente* per uno o più *articoli*), così i Business Components forniscono funzionalità utili nella collaborazione con altri Business Components.

2) Si integra in modo nativo in un sistema distribuito

Normalmente un sistema informativo distribuito viene suddiviso in *distribution tiers*.

L'approccio COSM prevede che un Business Component sia composto da quattro tiers: *user*, *workspace*, *enterprise*, *resource*.

Nella rielaborazione di questo modello, la Provincia di Bolzano ha ridotto a tre il numero di tiers: *user*, *workspace* ed *enterprise*.

Dei tre tier, l'unico obbligatorio è l'enterprise tier, mentre i primi due sono opzionali.

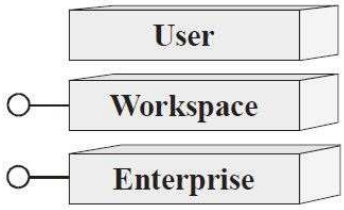


Figura 3.3. I tre tiers di cui si compone un Business Component in Provincia di Bolzano

Lo User tier è costituito dai componenti che realizzano l'interfaccia utente, l'Enterprise tier è costituito dai componenti che racchiudono la logica di business e di accesso ai dati, mentre il Workspace tier è un tier dedicato a funzionalità particolari, come p. es. la gestione eventuali caches locali per realizzare funzionalità off-line sul client, oppure l'elaborazione di logiche di persistenza particolarmente complesse.

Secondo questo approccio, *ciascun tier viene implementato da un Distributed Component* e può essere rilasciato (*deployed*) in modo autonomo e indipendente dagli altri, anche su ambienti diversi.

In termini concreti il Distributed Component è quindi un file .dll, .jar, .war, ecc.

In questo senso il Business Component agisce come elemento unificante che raggruppa i diversi tiers di cui è composto, implementati come Distributed Components e rilasciati in diverse locazioni fisiche.

Si consideri ad esempio uno specifico concetto di business, come il Fornitore:

dove è implementata l'entità Fornitore in un sistema distribuito? Potrebbe corrispondere ad alcune tabelle su una banca dati, alla logica di business su un application server (enterprise tier) e ad alcune maschere di un'interfaccia grafica (workspace e user tier). In questo senso, in un sistema distribuito, ogni concetto di business è realmente implementato in svariati punti del sistema.

Il concetto di Business Component mette in condizione di pensare e sviluppare tutti questi aspetti in modo unitario.

3) Si applica a tutto il ciclo di vita dello sviluppo

Se analizziamo come un certo concetto di business viene rappresentato e implementato in un sistema, osserviamo che appare in una grande varietà di forme durante l'intero ciclo di vita dello sviluppo.

I deliverables che rappresentano il concetto di business sono differenti a seconda che ci si trovi nell'analisi del progetto, a *design-time*, *build-time* o a *deployment-time* (p.

es. diagrammi UML, diagrammi E-R, classi, interfacce, dipendenze, packages, files di configurazione per il build, per il deploy, ecc.).

Tutti assieme, questi deliverables rappresentano quello specifico concetto di business che stiamo trattando.

Ciascuna fase del ciclo di vita dello sviluppo è una particolare vista del Business Component che rappresenta, implementa e diffonde quel concetto di business nel sistema informativo.

Il Business Component cattura e racchiude in un unico concetto software questi molteplici aspetti. Si può pertanto concludere dicendo che *il Business Component è un'implementazione software di un concetto ed è esso stesso una composizione di artefatti software.*

3.4.5 Business Component System

Un Business Component System corrisponde a un sistema informativo, per esempio un sistema di gestione dei pagamenti. In termini di Business Components, lo definiamo come segue:

Un Business Component System è un insieme di Business Components, che cooperano per fornire una soluzione ad un problema del business.

Per illustrare meglio questi concetti possiamo prendere ad esempio il componente "Procedimento", realizzato dalla Provincia di Bolzano, che si occupa di prendere in carico le richieste che i cittadini presentano all'amministrazione tramite il portale e-government e di inoltrarle in modo uniforme e standardizzato agli uffici competenti.

Il componente gestisce anche le comunicazioni in uscita dall'amministrazione provinciale verso i cittadini, sia in risposta ad una loro istanza, sia nel caso di provvedimenti originati dagli uffici (p. es. una sanzione ad un cittadino).

Esempi di domande trattate tramite questo componente sono le richieste di iscrizione a scuola, le richieste di iscrizione agli esami di bilinguismo, le richieste di contributi per l'artigianato, ecc.

Il componente Procedimento può essere visto come un sistema che coinvolge più Business Components che cooperano tra loro, ma che possono esistere indipendentemente l'uno dall'altro.

In questo senso il Procedimento si può considerare un Business Component System, che utilizza i seguenti Business Components:

- *FascicoloInformatico*: mette a disposizione le funzionalità relative al Protocollo Informatico a norma di legge e la relativa fascicolazione.
- *FirmaDigitaleMassiva*: mette a disposizione la funzionalità di firma digitale massiva a norma di legge.
- *ProduzioneDocumento*: consente di generare documenti combinando dati provenienti da applicativi gestionali con diversi template.

- *NotificaAreaPrivataCittadino*: consente di inviare al cittadino una notifica su più canali (sms, e-mail) per informarlo che sono avvenute variazioni nei confronti dell'amministrazione (p. es. è stato depositato nella sua area privata del portale e-government un nuovo documento relativo ad un procedimento in corso).

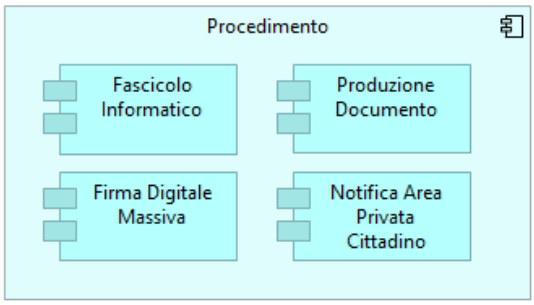


Figura 3.4. Business Component System "Procedimento" composto da più Business Components

Il comportamento del Business Component System "Procedimento" nei casi di comunicazioni rispettivamente in ingresso e in uscita dall'amministrazione è illustrato di seguito, utilizzando *UML sequence diagrams*:

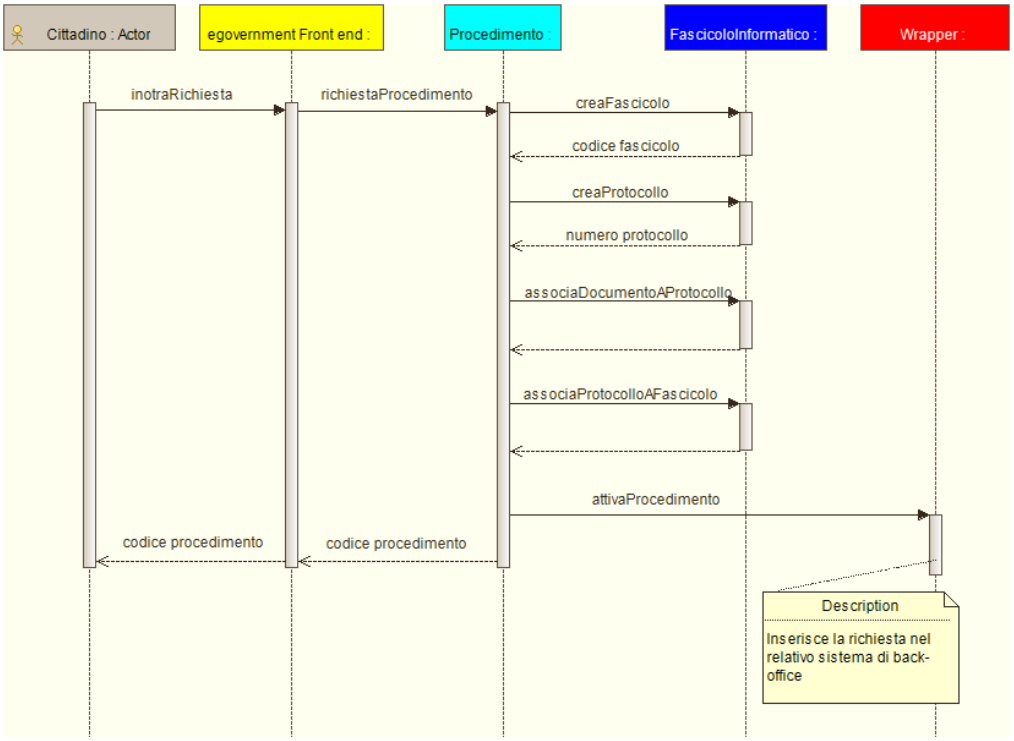


Figura 3.5. Business Component System "Procedimento": comunicazione in ingresso

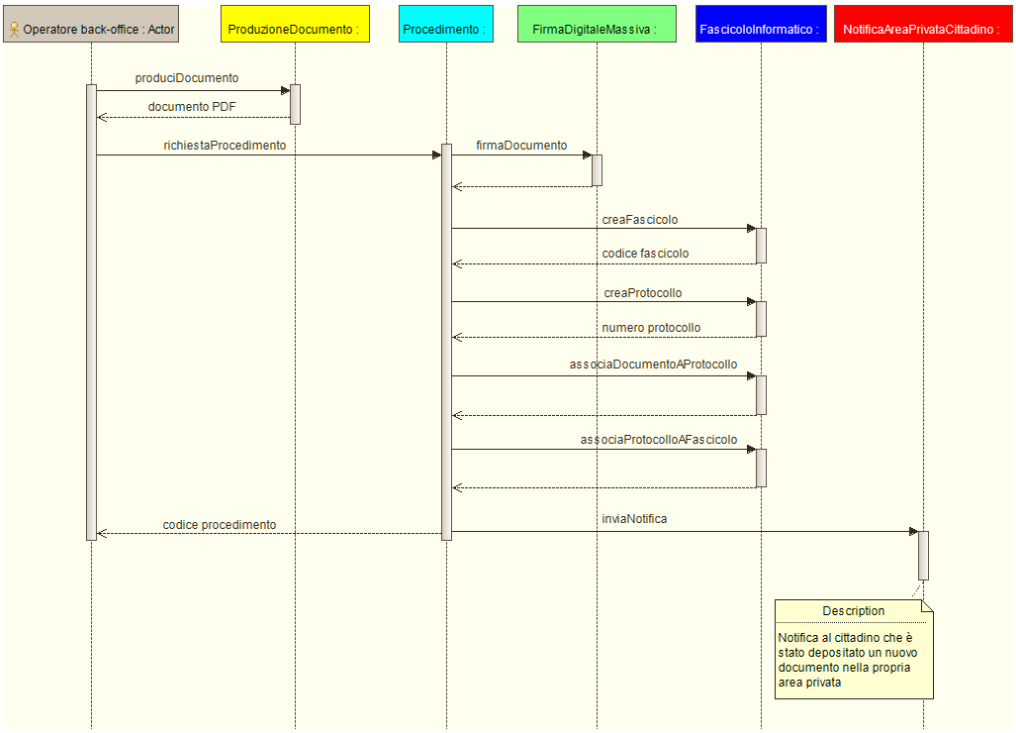


Figura 3.6. Business Component System "Procedimento": comunicazione in uscita

Nel primo caso la richiesta del cittadino, compilata tramite il front-end e-government, viene trasmessa all'amministrazione.

La richiesta viene presa in carico dal componente Procedimento, che provvede a validarla e processarla.

Se la richiesta è soggetta a protocollazione, il componente Procedimento richiede al componente FascicoloInformatico la creazione di un nuovo fascicolo, necessario per raccogliere la documentazione della pratica in corso di apertura. Di seguito richiede la protocollazione del documento elettronico in entrata e associa il documento protocollato al fascicolo corrispondente.

A questo punto il Procedimento inserisce la richiesta del cittadino con i relativi metadati su una coda, identificandola con il codice della tipologia di richiesta ricevuta. Da questa coda la richiesta viene prelevata da un *wrapper*, che provvede a inserirla nel sistema di back-office che tratta quello specifico tipo di istanza.

Questa operazione avviene in modo asincrono affinché il componente del Procedimento non abbia alcuna dipendenza dai sistemi di back-office che trattano i vari tipi di istanze.

Nel secondo caso l'operatore, tramite un sistema di back-office, esegue la generazione del documento da trasmettere al cittadino. Il sistema di back-office produce il documento e lo trasmette al componente Procedimento.

La richiesta con i suoi parametri viene ricevuta dal componente Procedimento, dove viene validata e processata. Il Procedimento richiede al componente FascicoloInformatico di protocollare il documento e associarlo al fascicolo della pratica in corso, qualora già esistente, o in caso contrario di creare un nuovo fascicolo per contenere la documentazione della pratica.

Il documento può essere firmato digitalmente in modo automatico dal componente Procedimento, tramite il componente di firma digitale massiva.

Il Procedimento infine può essere configurato per inviare al cittadino una notifica via e-mail o sms che lo informa che è stato prodotto un nuovo documento relativo ad una delle posizioni aperte presso l'amministrazione. Tramite autenticazione al portale e-government il cittadino potrà accedere alla propria area privata e prendere visione delle pratiche che ha in essere con l'amministrazione, accedendo anche ai documenti che lo riguardano, archiviati nel sistema di protocollo informatico dell'ente.

3.4.6 System-Level Component

Un Business Component System può essere visto a sua volta come un unico componente di più alto livello, chiamato System-Level Component. In questo caso il sistema nel suo complesso viene trattato come una *black box*, che espone le sue operazioni tramite un'interfaccia ben definita.

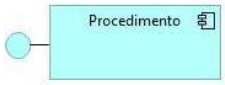


Figura 3.7. Business Component System "Procedimento" visto come System-Level Component

In altri casi, un System-Level Component potrebbe essere anche costituito da un *wrapper* attorno ad un sistema *legacy* [3].

3.5 Organizzazione logica dei componenti

La metodologia COSM prevede la suddivisione rigorosa dei componenti in domini applicativi secondo un approccio stratificato (*layering*).

Gli strati sono definiti in modo tale che, salvo eccezioni, ciascun layer possa accedere esclusivamente ai servizi del layer di livello inferiore.

Nei paragrafi seguenti sono descritti i layer più significativi.

[3] In informatica, un sistema legacy è un sistema IT che utilizza tecnologie obsolete ma che per varie ragioni (tecniche, economiche) è difficile sostituire con uno più attuale.

3.5.1 Layer Foundation

Il dominio applicativo denominato *Foundation* identifica i componenti di base, che sono utilizzati sia dalle applicazioni e-government che dalle applicazioni di back-office per fornire servizi comuni.

Appartengono a questa categoria i componenti visti nei paragrafi precedenti, quali la protocollazione, la firma digitale massiva, l'identity management, la produzione automatica di documenti da template, la gestione del procedimento, ecc.

3.5.2 Layer Core

I componenti presenti nel *Core* rappresentano quelle entità di Business che caratterizzano il domino applicativo della Provincia di Bolzano. In sostanza quelle informazioni o quei processi basilari per realizzare le funzioni tipiche della Provincia.

La caratteristica architetturale del dominio applicativo Core risiede nella indipendenza di questi componenti, cioè nel fatto che ogni elemento di questo dominio non "conosce" (= non ha nessuna dipendenza da) elementi in altri domini, con la sola esclusione di elementi del layer di Foundation. Piuttosto si ha la situazione inversa per cui molti componenti di altri domini dipendono dai componenti Core proprio perché questi ultimi realizzano funzionalità basilari tipiche della maggior parte delle applicazioni della Provincia di Bolzano. Questo garantisce la possibilità di sviluppare ed integrare elementi di questo dominio con facilità.

Esempi di componenti appartenenti a questa area sono: Anagrafica Unica, Contabilità, Gestione Contributo.

3.5.3 Layer Dashboard

La capacità di fornire informazioni in modo rapido e integrato, sia all'interno che all'esterno della Provincia di Bolzano, è un altro elemento importante dell'architettura di riferimento, che si ottiene con i cruscotti (*Dashboard*).

Si tratta quindi di interfacce utente, che possono essere rivolte al cittadino, accessibili via internet, allo scopo p. es. di fornire informazioni sullo stato di una pratica, oppure interne per i dipendenti.

Caratteristica architetturale del dominio applicativo *Dashboard*: ogni elemento di questo dominio può accedere ad elementi in qualsiasi altro dominio. Questo accesso però viene limitato ad interfacce web Services per diminuire le dipendenze da specifici ingredienti tecnologici.

3.5.4 Layer Auxiliary

L'Auxiliary è un dominio applicativo che include servizi, componenti e soluzioni a disposizione dello sviluppo ed integrazione di tutte le applicazioni.

Alcuni elementi di questo dominio sono indicati nella tabella seguente:

Componente	Descrizione
Logging	**Permette di avere una visione centralizzata dei log di tutti i componenti anche su nodi diversi**
Auditing	Permette di fare l'auditing centralizzato su DB di tutte le chiamate ai servizi dei componenti semplicemente dichiarando le interfacce *auditable*
Communication Manager	Gestisce le comunicazioni via e-mail o altri protocolli
Notification Manager	Permette la gestione di code di messaggi per la comunicazione asincrona tra componenti

Tabella 3.2. Componenti del dominio Auxiliary

Di seguito è riportata la gerarchia dei layers con le relative dipendenze: ciascun layer comunica esclusivamente con i layers sottostanti.

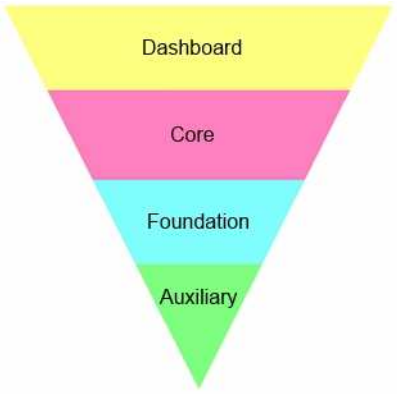


Figura 3.8. Gerarchia dei layers con le relative dipendenze

3.5.5 Caratteristiche delle applicazioni sviluppate con COSM

Le applicazioni sviluppate secondo questa metodologia presentano le seguenti caratteristiche architetturali:

- utilizzano i servizi del layer Foundation
- utilizzano i servizi del layer Core, sviluppando la propria base dati in modo compatibile con i componenti Core e Foundation (p. es. la banca dati deve essere

modellata in modo da tenere conto che il modulo dell'anagrafica è fornito dal componente Core dell'Anagrafica Unificata, e quindi l'applicazione dovrà gestire nella sua banca dati solo i dati specifici di quell'applicazione, che non sono generalizzabili e quindi non sono forniti dall'Anagrafica Unificata).

- consentono l'accesso ai propri dati e funzionalità esclusivamente via il portale unico del cittadino, via interfaccia utente dell'applicazione, o via web Services. Non è consentito di accedere ai dati dell'applicazione con accesso diretto alla banca dati.

A questi requisiti vanno aggiunti quelli standard per lo sviluppo di un'applicazione, e quelli specifici della Provincia di Bolzano (p. es. il pieno supporto al bilinguismo delle maschere e dei dati).

3.6 L'applicazione di COSM in Provincia di Bolzano

Come già più volte accennato, la metodologia COSM è stata recepita a livello concettuale dalla Provincia di Bolzano, ma è stata rielaborata e modificata in più parti per adattarla alle esigenze e necessità dell'amministrazione.

Nei paragrafi seguenti verranno esposti gli elementi distintivi dell'approccio COSM implementato in Provincia di Bolzano.

3.6.1 Component Execution Environment (CEE)

Il *Component Execution Environment* (CEE) è l'ambiente di esecuzione dei componenti (Distributed e Business Components).

Questo ambiente è realizzato in modo da essere pienamente compatibile con il framework architetturale presentato nei paragrafi precedenti e nel contempo fornire strumenti a supporto della progettazione, dello sviluppo e dell'esecuzione dei componenti.

L'obiettivo del prodotto è di esporre interfacce semplici e standardizzate, che nascondano agli sviluppatori funzionali la complessità dell'implementazione sottostante.

L'utilizzo di questa piattaforma ha consentito alla Provincia di Bolzano di:

- utilizzare in modo pervasivo in tutta l'amministrazione un unico stile architetturale orientato ai servizi e basato su componenti;
- focalizzarsi sullo sviluppo delle funzionalità necessarie per il business, piuttosto che di componenti infrastrutturali;
- mantenere nei singoli progetti l'allineamento e la compatibilità con i principi architetturali generali stabiliti dagli Enterprise Architects dell'amministrazione.

In origine il CEE fornito da Herzum Software era un *middleware* java-based costruito seguendo un approccio *best-of-breed*, in cui elementi open source erano

stati integrati e composti in un unico prodotto per la gestione del ciclo di vita dei componenti.

Nel corso degli anni la Provincia ha preferito riportare la metodologia COSM all'interno di infrastrutture più standard e in linea con il mercato.

Attualmente in Provincia di Bolzano sono stati realizzati e messi in produzione due CEE, basati su due piattaforme differenti:

– CEE.Net, basato su Microsoft .Net Framework e Microsoft Internet Information Service (IIS)
– CEE Enterprise Java, basato su JBoss Application Server

Il CEE.Net è un ambiente di esecuzione per Business Components scritti utilizzando la piattaforma Microsoft .Net.

Il CEE Java esegue Business Components realizzati parzialmente secondo le specifiche Enterprise JavaBeans 3.1 (JEE6). I componenti sono sviluppati facendo in modo che sia garantita il più possibile la portabilità su differenti implementazioni di Application Server JEE6. L'infrastruttura si interfaccia con pressoché ogni database: alle tabelle corrispondono oggetti Java e sono accedute secondo standard DAO.

In entrambi i CEE i Business Components sono esposti come Web services, descritti con WSDL standard, mentre la sicurezza è implementata sulla base delle specifiche *WS-Security* (WSS) dell'OASIS[4] utilizzando Kerberos token, username token e SAML token.

Entrambi i CEE supportano anche scenari di comunicazione asincrona.

In ambito Java questi scenari sono realizzati con l'uso di code basate su standard JMS.

3.6.2 Il modello

In base a quanto indicato dalla metodologia COSM, un Business Component viene disegnato attraverso un modello, in accordo con il paradigma *Model-driven Software Development*[5] *(MDSD)*. Il modello è in pratica un file xml, che definisce la struttura di un Business Component, ed è suddiviso in più sezioni. Le sezioni principali sono:

- *Services* − sono i servizi che il Business Component espone, e corrispondono ai servizi di business che una determinata area funzionale del business offre (p. es.

[4] L'OASIS (Organization for the Advancement of Structured Information Standards) è un consorzio globale che guida lo sviluppo, la convergenza e l'adozione di standard informatici aperti. In particolare l'OASIS si è occupato di definire e standardizzare le specifiche relative ai Web services, indicate solitamente come WS-* (dove l'asterisco racchiude un'insieme piuttosto esteso di specifiche, come p. es. WS-Security, WS-Trust, WS-ReliableMessaging, ecc.).

[5] Model-driven Software Development è una metodologia di sviluppo software che prevede la generazione automatica del software eseguibile a partire da modelli formali.

"GestioneSportelloMotorizzazione"), con le relative operazioni (p. es. "RinnovoPatente ").

- *Functional Data Types* − sono gli oggetti di business che vengono gestiti dai servizi.

Di seguito è riportato a titolo esemplificativo il modello del Business Component FascicoloInformatico, già descritto nei paragrafi precedenti.

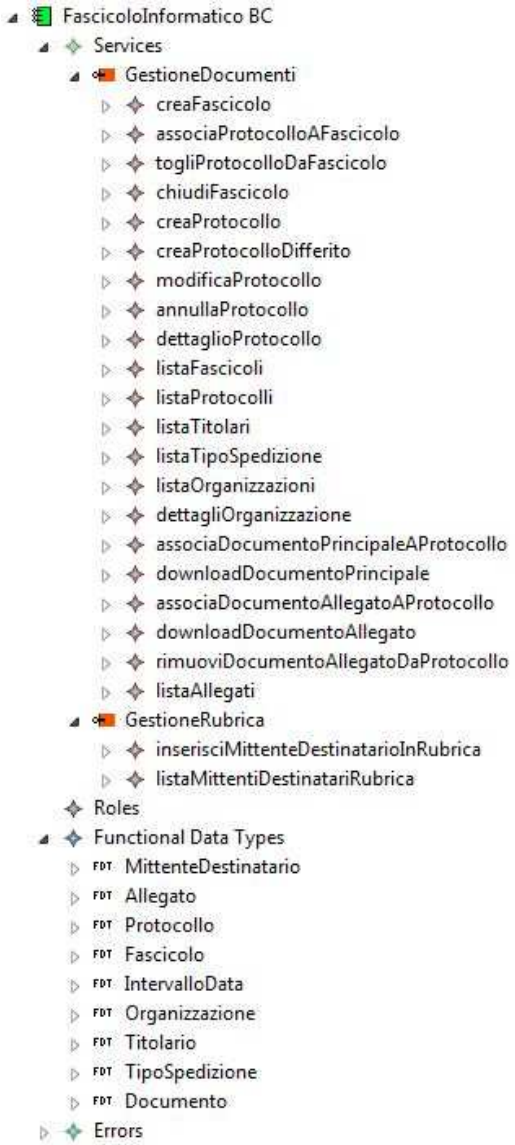


Figura 3.9. Modello del Business Component "FascicoloInformatico"

Come si vede, il modello descrive i servizi che compongono il Business Component e, per ciascun servizio, le relative operations. In questo caso il Business Component preso ad esempio è composto da due servizi, *GestioneDocumenti* e *GestioneRubrica*.

La suddivisione di un Business Component in uno o più servizi fa parte delle scelte progettuali che compie l'architetto funzionale incaricato di disegnare il modello del componente.

A titolo esemplificativo viene riportata di seguito la struttura dell'operation *creaProtocollo* del servizio *GestioneDocumenti*, come definita nel modello:

Figura 3.10. Operation del Business Component "FascicoloInformatico"

Come si vede, l'operation *creaProtocollo* si compone di una serie di parametri di input e di due parametri di output (contraddistinti dai versi delle frecce).

I parametri di input possono essere oggetti elementari (come p. es. il parametro "registro" di tipo stringa) oppure strutturati (come p. es. il parametro "protocollo" di tipo "Protocollo" oppure le liste di assegnatari per competenza e per conoscenza, di tipo "Organizzazione").

L' output funzionale dell'operazione è fornito dagli out parameters (nel nostro esempio il numero di protocollo), mentre per la comunicazione dell'esito irregolare dell'operazione dal punto di vista funzionale è previsto il BusinessResult, una risposta strutturata e multilingue che riporta il problema riscontrato in modo comprensibile per gli utenti.

Sono gestite con *SOAP exceptions* esclusivamente le eccezioni di tipo tecnico-infrastrutturale.

Abbiamo già visto che gli oggetti di business utilizzati dai servizi sono denominati FDT (*Functional Data Types*).

Si riporta a titolo esemplificativo la struttura dell'FDT *Protocollo*, utilizzata dall'operation *creaProtocollo* appena descritta.

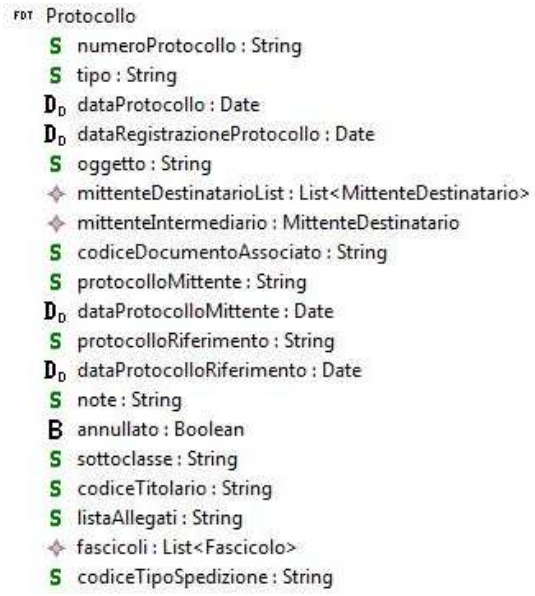


Figura 3.11. FDT del Business Component "FascicoloInformatico"

Ovviamente un FDT può essere a sua volta costituito da altri FDT, come si vede nel caso in esempio: l'FDT *Protocollo* è a sua volta composto da altri FDT come *MittenteDestinatario* o *Fascicolo.*

Allo scopo di garantire il maggiore allineamento possibile tra il business e l'IT, nella modellazione di un Business Component è fortemente raccomandato di mantenere una terminologia il più possibile aderente a quella utilizzata dal business, e di modellare le operazioni nel modo il più possibile corrispondente a quelle che vengono erogate nella pratica dagli operatori dell'amministrazione.

3.6.3 La generazione del codice

Seguendo le specifiche della metodologia COSM, il modello può essere disegnato, con un tool apposito, da un architetto funzionale, che può concentrarsi sulla definizione del Business Component, ignorando completamente l'implementazione che ne verrà fatta.

Per disegnare il modello, in Provincia di Bolzano sono disponibili due tools equivalenti, Studio e BCDesigner:

- Studio è un plug-in dell'IDE Eclipse, realizzato utilizzando l'*Eclipse Modeling Framework* (EMF), ed è più idoneo per chi lavora in Java;
- BCDesigner è un applicazione .Net sganciata da qualsiasi IDE, pertanto è più in linea con la separazione dei ruoli proposta dal modello COSM, in cui l'architetto funzionale definisce il modello del BC in maniera totalmente agnostica rispetto alla tecnologia con cui verrà implementato.

Le figure riportate nel paragrafo precedente sono tratte da un modello disegnato con Studio.

Una volta disegnato il modello, è possibile generare il codice attraverso un'apposita opzione del tool di modellazione.

La generazione del codice produce le interfacce e classi necessarie per implementare il Business Component. Ovviamente il codice che può essere generato è quello infrastrutturale, di mappatura tra le classi e i relativi oggetti persistiti sul database, di security, di logging, ecc., mentre la parte di codice che implementa la business logic relativa allo specifico Business Component va scritta ad hoc di volta in volta.

La filosofia alla base di questo approccio è quella di fare in modo che lo sviluppatore possa concentrarsi sullo sviluppo delle funzionalità di business, piuttosto che sullo sviluppo di componenti infrastrutturali ripetitive.

A partire dal modello del Business Component, per ciascun servizio viene inoltre generato uno specifico Web service con il relativo WSDL, le cui operations sono quelle descritte nel modello stesso.

Il modello è strutturato in modo da garantire l'interoperabilità tra i Business Components implementati su piattaforme tecnologiche diverse, in particolare Java e .Net.

È quindi possibile sviluppare p. es. un Business Component in Java, eseguirlo all'interno dell'infrastruttura di run-time CEE JBoss, e utilizzarlo in un'applicazione client realizzata con il CEE.Net. Le due applicazioni condividono lo stesso modello, quindi di fatto la stessa definizione di Business Component, gli stessi FDT, ecc.

In questo modo, una volta definito il modello con un livello di dettaglio sufficiente, è possibile procedere parallelamente allo sviluppo del componente server, che espone il servizio, e del componente client, che lo consuma. Si parla in questo caso di approccio *contract first*, dove il modello costituisce il contratto che entrambe le parti devono rispettare.

Se durante lo sviluppo si rendono necessarie modifiche al modello, dopo la modifica è sempre possibile rigenerare il codice. Il codice generato viene mantenuto separato dal codice scritto ad-hoc, per cui, se ci si attiene alle linee guida per lo sviluppo, la rigenerazione del codice non ha effetti negativi sull'implementazione del Business Component.

Come già detto nel capitolo precedente, per realizzare l'interoperabilità tra tecnologie differenti (p. es. Java e .Net) sarebbe sufficiente utilizzare i Web services. Il modello però è la rappresentazione del Business Component, che può essere esteso senza vincoli secondo le necessità dell'Ente. L'adozione del modello infatti dà la possibilità di forzare l'applicazione di regole sintattiche.

Al WSDL manca il concetto di componentizzazione dei servizi, è troppo flessibile nella descrizione dei servizi, ed è impossibile forzare l'applicazione di regole (come p. es. quelle sui namespaces e sulla sintassi).

Dal momento che il modello aggrega più Web services utilizzando lo stesso namespace, è possibile condividere gli FDT tra più servizi che condividono lo stesso namespace[6].

Eseguendo il deploy del Business Component nell'application server, è comunque possibile ottenere anche i WSDL dei servizi che il modello descrive, pertanto, quando necessario, è sempre possibile condividere il WSDL di un servizio con sistemi di terze parti, realizzando di fatto l'interoperabilità con qualsiasi piattaforma abilitata alla tecnologia Web services.

3.6.4 Il concetto di applicazione

La Provincia di Bolzano ha adottato la definizione di applicazione suggerita da TOGAF:

un'applicazione è un sistema IT rilasciato e operativo, che supporta funzioni e servizi di business; p. es. un sistema per la gestione delle paghe. Le applicazioni usano dati e sono supportate da molteplici component tecnologici, ma sono distinte dai componenti tecnologici che le supportano.

In accordo con questa definizione, un'applicazione può essere basata su uno stack tecnologico omogeneo oppure no. Un'applicazione può includere anche interfacce utente diverse, destinate p. es. a differenti gruppi di utenti (p. es. cittadini e dipendenti interni all'amministrazione) oppure a consentire l'accesso da dispositivi diversi (p. es. PC, Smartphones, Tablet).

In base a quanto appena esposto, possiamo rappresentare un'applicazione come segue:

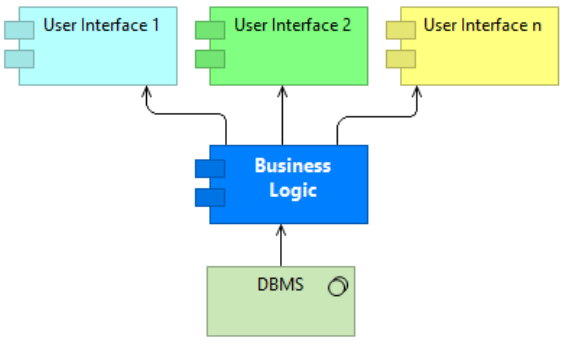


Figure 3.12. Gli elementi che costituiscono un'applicazione

Un'applicazione è quindi composta da un data store (se è necessario memorizzare dei dati), da un'unità che processa la logica di business e da una o più interfacce utente.

[6] In realtà ogni web service ha i propri FDT; se si usa lo stesso FDT in servizi diversi di uno stesso Business Component, vengono create più copie della definizione, che conterranno gli stessi attributi.

La logica di business è costituita da un insieme di regole, definite dal business, che si applicano all'elaborazione, alla consistenza e all'accessibilità dei dati. La business logic include quindi anche il controllo degli accessi ai dati e alle funzionalità: si può accedere ai dati solo attraverso lo strato di business logic.

È inoltre sostenuto con forza il concetto di *isola di dati*: i dati sono di pertinenza dell'applicazione, nessun'altra applicazione accede in modo diretto ai dati che non sono di propria pertinenza. L'accesso ai dati di un'altra applicazione avviene solo attraverso lo strato di business logic dell'applicazione titolare dei dati.

Sono mantenuti i *tiers* previsti dal modello COSM elencati in precedenza: User tier (per gli artefatti software delle interfacce grafiche), Enterprise tier (il componente che racchiude la business logic e gestisce l'accesso ai dati) e, se necessario, il Workspace tier (un tier per funzionalità particolari, come p. es. la gestione di sessioni di lavoro temporanee sul client, prima di contattare un enterprise service - collocato tra la interfaccia utente e il tier della business logic - oppure un tier dedicato tra la business logic e il data store per gestire logica di persistenza particolarmente complessa, come p. es. la trasformazione di entità di business complesse in LDAP schemas).

Sono state rispettate anche le regole di comunicazione tra tiers: ciascun tier può comunicare solo con il tier direttamente sottostante. L'Enterprise tier (che implementa la business logic) è l'unico tier obbligatorio.

3.6.5 La comunicazione tra applicazioni

Il modello COSM prevede che all'interno dello stesso dominio di sicurezza, le comunicazioni tra applicazioni avvengano esclusivamente tramite Business Components che comunicano tra loro per mezzo di Web services SOAP secondo le specifiche WS-Security.

Questo scenario di integrazione tra applicazioni offre indubbiamente un'elevata versatilità: è possibile p. es. sviluppare le componenti server di un'applicazione con una tecnologia, ed utilizzarne un'altra per le componenti client.

Inoltre i servizi sono nativamente riutilizzabili, favorendo in questo modo il riuso dei servizi da parte di più applicazioni.

Nella pratica però ci sono contesti in cui applicazioni tecnologicamente omogenee comunicano tra loro in modo ottimizzato (sia dal punto di vista dei costi che delle performance) utilizzando protocolli nativi. Inoltre in questi casi nella scrittura di codice è possibile sfruttare al massimo le potenzialità offerte nativamente dalle varie piattaforme e utilizzare in maniera spinta i design patterns.

Per questa ragione in Provincia di Bolzano si è deciso di differenziare la modalità di interazione tra le applicazioni a seconda dei contesti in cui si trovano, utilizzando come discriminante il concetto di area funzionale.

Con *area funzionale* si intende *un insieme di attività svolte all'interno dell'azienda, raggruppate in base al criterio di omogeneità delle competenze necessarie per svolgerle.* Un'area funzionale ha uno scopo particolare e funzionalmente specializzato nell'organizzazione, e può essere separata dalle funzioni di altri settori.

Esempi di aree funzionali all'interno della Provincia sono: risorse umane, organizzazione e controlling, finanze, sistemi informativi.

Se un'applicazione deve integrarsi con altre applicazioni tecnologicamente omogenee all'interno della stessa area funzionale, per farlo può effettuare chiamate ai servizi con protocolli nativi, specifici di quella tecnologia (p. es. RMI nel caso di Java).

Se invece un'applicazione deve esporre i suoi servizi al di fuori dell'area funzionale per la quale è stata progettata, p. es. ad altre aree funzionali o all'e-government, la modalità di esposizione dei servizi è quella descritta in precedenza, basata sulla definizione di uno o più Business Components, che comunicano tra loro utilizzando Web services SOAP con WS-Security per l'autenticazione.

In questo modo si garantisce una migliore gestione dei cambiamenti all'interno dell'area funzionale (p. es. nuovi requisiti a fronte di cambiamenti legislativi), che non hanno impatto all'esterno dell'area funzionale, mantenendo stabile l'interfaccia verso le altre aree funzionali e verso l'e-government.

Di seguito si riporta un'immagine che descrive le diverse tipologie di comunicazione tra applicazioni

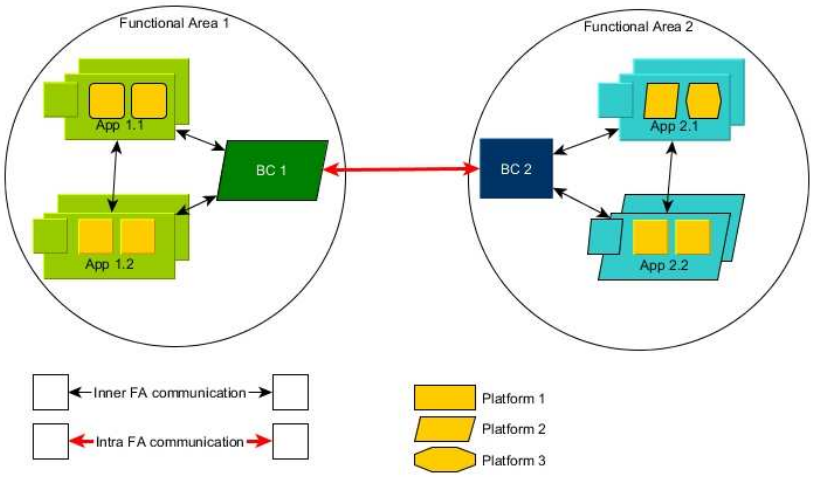


Figura 3.13. Diverse tipologie di comunicazione tra applicazioni (vedi [4])

4 Enterprise Architecture, TOGAF e ArchiMate

Il capitolo approfondisce la definizione dell'Enterprise Architecture data in precedenza ed esamina nel dettaglio il ruolo dell'Enterprise Architect.

In seguito si introducono il framework TOGAF e il linguaggio di modellazione ArchiMate, riportandone le caratteristiche salienti e accennando a come sono stati impiegati in Provincia di Bolzano.

4.1 Enterprise Architecture

Man mano che la complessità dell'IT aumenta, sempre più organizzazioni realizzano di avere bisogno di un'architettura.

Anche all'interno della Provincia di Bolzano, nel contesto della riorganizzazione dell'IT di cui abbiamo parlato in precedenza, è stato istituito il gruppo degli Enterprise Architects.

Riportiamo la definizione di Enterprise Architecture già data nel primo capitolo:

l'Enterprise Architecture è la disciplina che studia la struttura di un'organizzazione, i suoi processi operativi, i suoi obiettivi, i sistemi informativi a supporto, i flussi informativi e le tecnologie utilizzate, fornendo la visione di come tutti questi elementi si legano tra loro.

La definizione di che cosa sia un'architettura IT, di quali titoli e competenze debbano avere gli architetti e di quale ruolo ricoprano all'interno dell'azienda, varia però da organizzazione ad organizzazione.

Nei seguenti paragrafi si cercherà di individuare un'insieme di attività che caratterizzano l'Enterprise Architect e di dare indicazioni concrete sul suo ruolo nell'organizzazione.

4.1.1 Enterprise Architect vs. Solution Architect

Innanzitutto è necessario fare una distinzione tra due figure simili, che nella pratica spesso si sovrappongono, ma che in realtà hanno compiti ben distinti: l'*Enterprise Architect* e il *Solution Architect*.

Solitamente ci si riferisce al Solution Architect come alla figura responsabile della progettazione dell'architettura di una specifica applicazione.

Il Solution Architect si occupa di definire la struttura logica e il comportamento del software in risposta ai requisiti funzionali analizzati, e inoltre di specificare l'infrastruttura e la piattaforma più adatte per soddisfare i requisiti di qualità del servizio (p. es. i Service Level Agreement[1]) stabiliti per il progetto specifico.

Questo ruolo è spesso chiamato anche Software Architect o Application Architect.

La differenza tra il Solution Architect e l'Enterprise Architect è nello *scope*, cioè nel diverso ambito in cui queste due figure operano: il Solution Architect agisce prevalentemente a livello del singolo progetto, mentre l'Enterprise Architect opera a livello dell'intera organizzazione.

Nella figura seguente sono riportati gli *scope* dei diversi ruoli coinvolti nella definizione delle architetture.

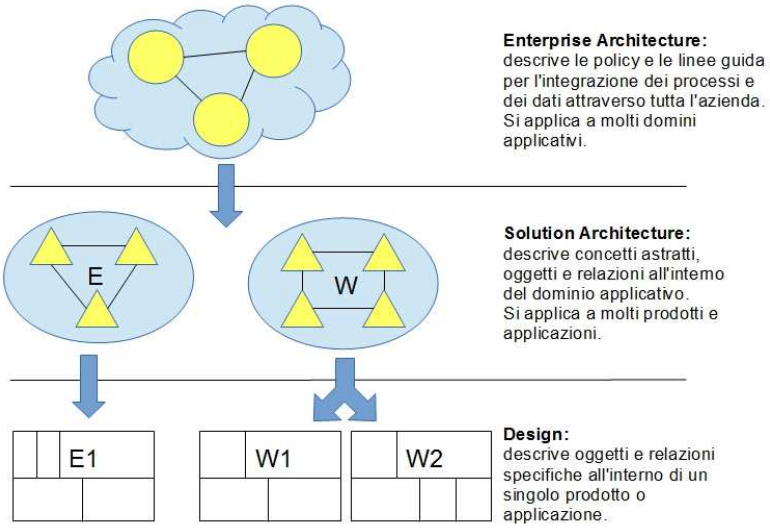


Figura 4.1. Differenti ruoli coinvolti nella definizione di architetture (vedi [5])

Nell'esempio in figura si vede come da un insieme omogeneo di specifiche architetturali possano derivare più architetture applicative (nell'esempio sono

[1] I Service Level Agreement (SLA) sono strumenti contrattuali attraverso i quali si definiscono le metriche di servizio che devono essere rispettate da un fornitore di servizi nei confronti dei propri clienti. Gli SLA possono riguardare p. es. le performance di rete, la disponibilità delle componenti di sistema o del servizio *end to end*, i tempi di risposta delle transazioni Web.

riportate un'architettura Web "W" e una basata su Enterprise Application Integration "E"). Per ciascuna architettura ci sono potenzialmente più implementazioni possibili, che rispettano le linee guida e le specifiche architetturali di dettaglio (le applicazioni E1, W1, W2 indicate in figura).

Come è indicato in figura, l'Enterprise Architect si occupa, tra le altre cose, di descrivere le regole e le linee guida per l'integrazione dei processi e dei dati tra i diversi sistemi presenti nell'organizzazione concentrandosi sulle parti comuni tra i sistemi, mentre il Solution Architect si occupa di calare queste specifiche a livello di progetto e di disegnare la struttura e il comportamento di un specifico sistema.

Per esemplificare il concetto, facendo un'analogia con il mondo dell'edilizia, l'Enterprise Architect rappresenta l'urbanista, che redige il Piano Urbanistico, mentre il Solution Architect corrisponde all'architetto che progetta l'edificio attenendosi alle norme stabilite dal Piano Urbanistico.

Obiettivo dell'Enterprise Architect è di individuare cosa debba essere messo a fattor comune e gestito a livello aziendale. I criteri per identificare cosa deve essere oggetto dell'Enterprise Architecture sono dettati dagli obiettivi e dai requisiti di business. Se i requisiti non sono chiari, il lavoro iniziale dell'architettura è di determinare e chiarire gli obiettivi e la strategia aziendale.

A titolo di esempio, possiamo considerare l'obiettivo di business di avere un'unica visione dell'entità "cittadino", comune a tutti gli uffici e le ripartizioni dell'ente.

L'architettura in questo caso deve individuare una definizione dell'entità "cittadino" con tutti i relativi attributi, condivisa da tutta l'amministrazione. Questo richiede un'analisi approfondita: definire un'unica visione di "cittadino" coinvolgerà nuovi processi di business, l'aggregazione e normalizzazione di dati dei cittadini da molteplici fonti, l'integrazione di molte applicazioni, nonché l'infrastruttura per supportare tutto questo. L'Enterprise Architect deve determinare come risolvere tutti questi aspetti, riferiti ad ambiti diversi (del business, dei dati, applicativo e tecnologico) per soddisfare il requisito di business di avere un'unica visione dell'entità "cittadino".

È lo scope architetturale pertanto che determina il ruolo e la responsabilità di un architetto:

- l'Enterprise Architect è responsabile della definizione di standard e la progettazione di soluzioni per ciò che è comune a tutti i settori dell'azienda. Inoltre stabilisce il contesto tecnologico all'interno del quale le applicazioni devono essere progettate.
- il Solution Architect garantisce che le singole applicazioni vengano progettate rispettando le specifiche e il contesto tecnologico definito dall'Enterprise Architect.

L'obiettivo comune ad entrambi i ruoli è di evitare scelte tecnologiche e applicative contingenti, che comportano successivamente elevati costi di manutenzione, integrazione ed evoluzione.

Gli Enterprise Architects si occupano di tematiche come p. es. la definizione di specifiche per gli stili e i principi architetturali, i protocolli di autenticazione, la federazione tra domini, le regole per la progettazione di applicazioni, le SOA policy, la SOA Governance, l'integrazione tra sistemi, ecc.

4.1.2 I domini architetturali

I framework architetturali forniscono un modello concettuale che possiamo usare per inquadrare i differenti domini dell'Enterprise Architecture e le relazioni che li legano. Esistono diversi framework per l'Enterprise Architecture, come p. es. Zachman, TOGAF, FEAF, e DoDAF; quasi tutti però prendono in considerazione i quattro domini architetturali illustrati nella figura seguente: Business, Information (o Data), Application, Technology.

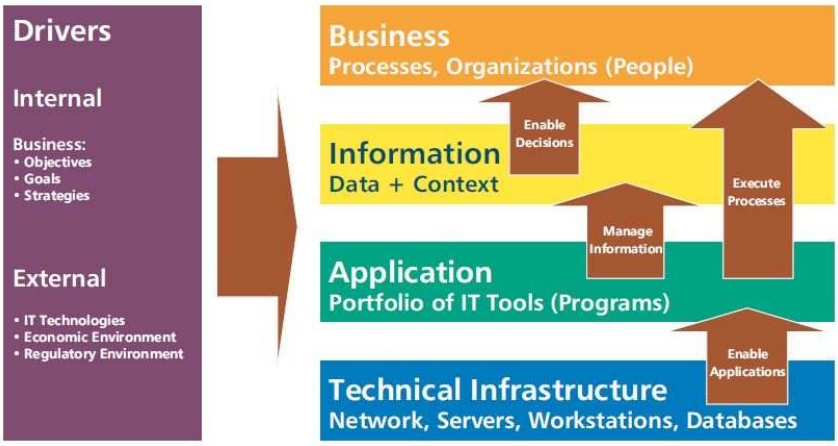


Figura 4.2. I domini architetturali (vedi [5])

Ciascun dominio descrive un aspetto particolare dell'architettura dell'intera azienda ed ha una relazione ben definita con gli altri domini.

Per esempio vediamo che le informazioni mettono i manager in condizione di prendere decisioni, mentre le applicazioni permettono di gestire i processi di business e le informazioni. Infine l'infrastruttura tecnologica rende possibile l'esecuzione delle applicazioni, le quali supportano il business nelle sue attività.

Di seguito esaminiamo brevemente ciascuno dominio e il ruolo che l'architetto svolge all'interno del dominio in esame, sia a livello Enterprise che del singolo progetto.

Business

L'architetto di business si occupa di descrivere il business, la sua organizzazione e i suoi processi, in modo tale che i sistemi IT possano effettivamente supportarli.

L'obiettivo primario è di ottenere l'allineamento dei requisiti di business con l'implementazione dei sistemi.

Un metodo per raggiungere questo obiettivo è rappresentare i processi di business con digrammi specifici, realizzati con formalismi come p. es. BPMN (*Business Process Model and Notation*) e UML (*Unified Modeling Language*) *activity diagram*. È compito degli architetti suddividere l'intero processo di business in sottoprocessi, compiti e decisioni.

In Provincia di Bolzano il compito di modellare i processi di business è stato assegnato al ruolo del Demand Manager.

Information

L'Information Architect si occupa di fornire una rappresentazione delle entità[2] di business coinvolte sia a livello dello specifico progetto che a livello enterprise.

L'obiettivo è quello di evitare ridondanze e duplicazioni, definendo in modo univoco le entità di business generiche (p. es. "Persona", "Ditta", ecc.) con i relativi attributi, e specializzandole per ottenere le entità di business dei singoli progetti (p. es. l'entità di business "Dipendente" eredita da "Persona", cioè ha tutti gli attributi dell'entità "Persona" - nome, cognome, codice fiscale, ecc - ed aggiunge gli attributi specifici del dipendente - tipo di contratto, numero di matricola, ecc.).

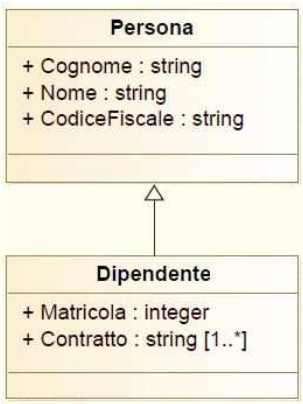


Figura 4.3. Ereditarietà tra entità di business

In questo modo è possibile ottenere una vista completa dei dati a livello enterprise e razionalizzare l'accesso da più applicazioni alle informazioni comuni, minimizzando le duplicazioni e le ridondanze.

Esempi tipici di informazioni comuni a più applicazioni sono le anagrafiche dei dipendenti, dei cittadini, delle imprese, gli stradari, ecc.

[2] con il termine *entità* in informatica si intendono classi di oggetti (fatti, cose, persone, ecc.) che hanno proprietà comuni ed esistenza autonoma ai fini dell'applicazione di interesse.

I modelli informativi sono tipicamente creati con diagrammi di tipo *UML class diagram* o *entity relationship diagrams*, che rappresentano le entità coinvolte nel progetto specifico e le relazioni tra di esse.

Application

L'Application Architect si occupa di ciò che è comune a livello applicativo.

Questo significa individuare modelli di riferimento, standard e stili architetturali che promuovano l'utilizzo di servizi e piattaforme comuni, evitando soluzioni applicative specifiche per ciascun progetto.

L'obiettivo non è di limitare la creatività degli sviluppatori, ma di migliorare l'integrazione tra le applicazioni, consentendo la condivisione di informazioni e la riduzione del costo e della complessità nella manutenzione delle applicazioni. Per raggiungere questi obiettivi, l'Application Architect deve innanzitutto definire lo stile architetturale da utilizzare.

Un altro obiettivo degli Application Architects è quello di creare standard e linee guida che descrivano specifici aspetti dello sviluppo, p. es. il servizio di *logging*, la classificazione di *error, warning, information*, ecc.

Anche gli aspetti tecnologici quali le prestazioni, la scalabilità, l'affidabilità, la sicurezza, dovrebbero essere inclusi nell'architettura di riferimento, e non nei singoli progetti.

I modelli architetturali sono tipicamente creati utilizzando diagrammi UML (p. es. *component diagram, sequence diagram, collaboration diagram*).

A livello di progetto, l'Application Architect si occupa di applicare l'architettura di riferimento allo specifico progetto.

Technology

Nel dominio tecnologico, il Technology Architect è responsabile di fornire piattaforme comuni a supporto dello stile architetturale scelto nell'architettura di riferimento, con il livello di qualità del servizio adeguato.

La Technology Architecture comprende i sistemi, lo storage, la sicurezza, le reti, i data centers, il capacity planning, il monitoraggio e l'analisi delle performance, la pianificazione del disaster recovery e della business continuity, ecc.

A livello di progetto, il Technology Architect si occupa di mettere a disposizione le piattaforme necessarie, e di integrarle nell'infrastruttura esistente, in conformità con l'architettura tecnologica di riferimento.

Questi aspetti in Provincia di Bolzano sono demandati al settore *RUN*, che si occupa di garantire l'operatività dei sistemi e la continuità dei servizi.

4.1.3 La mappa di processo dell'Enterprise Architecture

Possiamo individuare tre categorie di compiti che l'Enterprise Architect svolge in un'azienda.

- compiti strategici (*strategic tasks*)
- governance dell'architettura (*architecture governance*)
- compiti trasversali (*cross cutting tasks*)

Nella figura seguente sono illustrate le attività che fanno capo ad ogni singolo compito dell'Enterprise Architect.

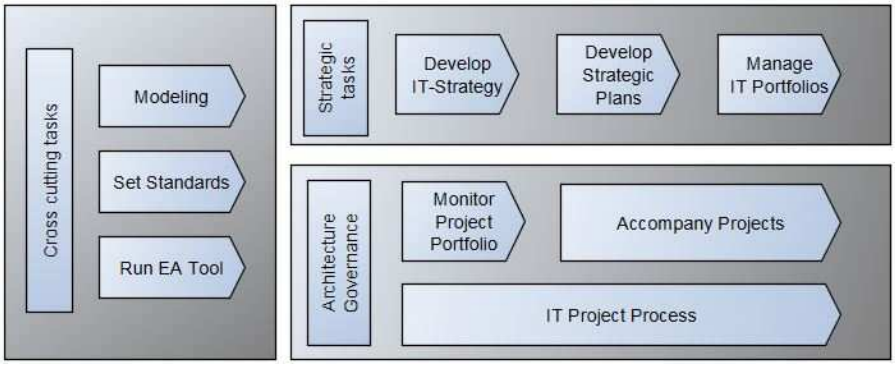


Figura 4.4. Il processo dell'Enterprise Architecture (vedi [11])

Compiti strategici

Con il termine *strategia* intendiamo un piano di azione pensato per ottenere un particolare obiettivo. Pertanto è necessario innanzitutto fissare un obiettivo, e poi definire una o più vie per raggiungerlo. Definire una strategia significa scegliere quale via percorrere per raggiungere l'obiettivo e trasformarla in un piano.

Gli Enterprise Architects assistono il CIO nella definizione della *strategia IT*. L'*IT Portfolio Management* fornisce i dati di base necessari per la *pianificazione strategica*.

Nella figura seguente sono riportati gli elementi che tipicamente compongono la strategia IT.

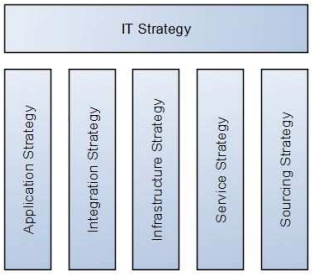


Figura 4.5. Gli elementi della strategia IT (vedi [11])

Governance dell'architettura

Una volta definite le strategie e derivate da queste le regole e le linee guida architetturali, è necessario divulgarle e comunicarle, e fare in modo che vengano applicate e implementate. La migliore opportunità è quella di applicarle a progetti critici, progetti che hanno il potenziale per cambiare l'architettura. Per individuare questi progetti è necessario tenere sotto controllo il *project portfolio.*

Una volta individuati questi progetti strategici, un team di architetti deve accompagnarli nel loro svolgimento, in modo da assicurare che le regole e linee guida vengano applicate e rispettate.

Compiti trasversali

Vi sono infine attività che vanno svolte in modo continuativo, quali p. es. disegnare e manutenere una mappa di tutte le applicazioni distribuite nell'azienda (utilizzando modelli UML come il *collaboration diagram*, il *component diagram* e il *deployment diagram*) ed individuare e definire standard di riferimento validi per l'azienda. Queste attività consentono di sviluppare e mantenere aggiornata l'architettura IT nell'azienda.

A conclusione di questa panoramica sui compiti dell'architetto nell'azienda, desideriamo ribadire l'importanza del coinvolgimento dell'Enterprise Architect nei progetti strategici con una frase del consulente del Cutter Consortium Jeroen van Tyn, "All architecture is local". Questo significa che, qualsiasi sia l'architettura stabilita, essa è efficace solo quando viene effettivamente applicata localmente a livello di progetto. In altre parole, il valore non viene dalla creazione dell'architettura. La migliore architettura non fornisce alcun valore se non viene utilizzata, e in generale le possibilità di applicare l'architettura si presentano con i progetti.

4.1.4 L'Enterprise Architecture a supporto della SOA

L'Enterprise Architecture fornisce framework, strumenti e tecniche per supportare le organizzazioni nello sviluppo e nel mantenimento delle proprie architetture SOA.

L'Enterprise Architecture è particolarmente utile per incentivare l'adozione dell'architettura SOA in un'organizzazione, perché fa in modo di mantenere uno stretto legame tra il contesto del business e l'IT, assicurando che le esigenze degli stakeholder siano soddisfatte. Questo legame è il fondamento per l'interoperabilità e il riuso dei servizi.

Con il supporto dell'Enterprise Architecture è possibile:

- mostrare come le soluzioni SOA possono essere efficacemente progettate per supportare il business nel raggiungimento dei suoi obiettivi;
- individuare i servizi che dovrebbero essere creati ad hoc e quelli che possono essere riutilizzati;

– presentare il modo corretto di progettare i servizi affinché siano riutilizzabili e di valore per il business.

Se non si adotta l'Enterprise Architecture è possibile andare incontro ai seguenti rischi:

– limitata agilità dell'azienda (l'IT è così veloce nel rispondere alle esigenze del business)
– difficoltà ad identificare e orchestrare i servizi SOA
– proliferazione dei servizi eccessiva e incontrollata
– interoperabilità limitata dei servizi SOA
– riuso limitato dei servizi SOA
– più SOAs isolate nella stessa azienda
– difficoltà nell'evoluzione dell'implementazione della SOA

4.2 Il framework TOGAF®

Un framework architetturale consiste in una metodologia dettagliata, con un insieme di strumenti a supporto, per sviluppare un'architettura Enterprise. È un modo per pensare l'Enterprise Architecture, per categorizzare e organizzare i concetti architetturali e gli artefatti, suddividendo l'azienda in viewpoints e agevolando quindi l'approccio "*separation of concerns*".

Come accennato nel paragrafo precedente, esistono diversi framework architetturali. La Provincia di Bolzano ha adottato il framework TOGAF (*The Open Group*[3] *Architecture Framework*).

4.2.1 La struttura del framework TOGAF

TOGAF è stato sviluppato dall'*Open Group Architecture Forum*, composto da oltre 200 imprese del settore IT, sulla base delle esperienze raccolte da ciascuna impresa nello sviluppo di architetture. L'Open Group Architecture Forum si occupa di mantenere il framework e ne pubblica periodicamente le versioni successive. La versione attuale di TOGAF è la 9.1.[4]

Come definito dall'Open Group "*TOGAF fornisce i metodi e gli strumenti per favorire l'accettazione, lo sviluppo, l'uso e la manutenzione di un'architettura*

[3] L'*Open Group* è un consorzio indipendente costituito da più di 400 imprese del settore IT, orientato allo sviluppo di standard informatici aperti e *vendor-neutral*. Tra le imprese più importanti citiamo Hewlett-Packard, IBM, Oracle, Philips e Capgemini.

[4] Si tratta di un *maintenance update* di TOGAF 9, che corregge errori e imprecisioni e garantisce un uso più coerente della terminologia rispetto a TOGAF 9.

Enterprise. Si basa su un modello di processo iterativo, supportato da best practices e da un insieme riutilizzabile di asset architetturali".

Non si tratta quindi di un'architettura "*one-size-fits-all*", ma di un framework generico per sviluppare architetture che vanno incontro alle diverse esigenze del business.

TOGAF è composto da diversi elementi:

- *The Architecture Development Method (ADM)* − descrive un processo standardizzato per sviluppare un'architettura su misura per la propria organizzazione, che soddisfi i requisiti del business.
- *ADM Guidelines and Techniques* − è un insieme di linee guida e tecniche per applicare l'ADM ad un'organizzazione.
- *Architecture Content Framework* − fornisce un metamodello strutturato per i prodotti del lavoro architetturale, e una panoramica degli elaborati architetturali che tipicamente vengono forniti.
- *Enterprise Continuum and Tools* − propone tassonomie e strumenti appropriati per categorizzare e memorizzare i prodotti del lavoro architetturale nell'azienda, in modo da gestirne l'evoluzione e il riuso.
- *TOGAF Reference Models* − fornisce due modelli di riferimento architetturali:
  - *TRM (TOGAF Reference Model)*
  - *III-RM (Integrated Information Infrastructure Reference Model)*
- *Architecture Capability Framework* − è un insieme di risorse, linee guida, modelli e informazioni fornite per assistere l'architetto nel suo lavoro.

TOGAF suddivide l'architettura complessiva in quattro sezioni specifiche, relative ai quattro domini architetturali, come mostrato nella tabella seguente.

Architettura	Descrizione
Business Architecture	Include le strategie di business, la governance, l'organizzazione e i processi di business principali.
Data (Information) Architecture	Comprende l'insieme dei dati di un'organizzazione, le entità di business con le relative gerarchie.
Application Architecture	Fornisce un modello di architettura applicativa per le singole applicazioni, le loro interazioni e le loro relazioni con i processi principali dell'organizzazione.
Technology Architecture	Racchiude il software e l'hardware necessario per eseguire le applicazioni ed i servizi, include l'infrastruttura IT, il middleware, la rete, ecc.

Tabella 4.1. I quattro tipi di architettura che compongono l'architettura complessiva

Il processo centrale di TOGAF è denominato *Architecture Development Method* (ADM).

Si tratta di un processo iterativo, che ha lo scopo di fornire una metodologia per ottenere un'architettura Enterprise personalizzata per l'organizzazione che soddisfi i requisiti del business.

L'ADM descrive:

- un metodo sicuro ed affidabile per sviluppare un'architettura Enterprise;
- un modo per sviluppare architetture nei differenti domini (Business, Data, Application e Technology) che soddisfino requisiti anche complessi;
- linee guida e strumenti a supporto dello sviluppo di un'architettura.

4.2.2 Il processo ADM

Il processo ADM, rappresentato nella figura seguente, è strutturato in fasi, in modo da suddividere il processo complesso dello sviluppo di un'architettura in passi più semplici (secondo il principio di *"separation of concerns"*), nei quali l'Enterprise Architect considera aspetti differenti del problema generale.

Si tratta, come già anticipato, di un metodo iterativo, in cui è consentito percorrere più volte tutto il ciclo, iterare tra due fasi adiacenti o ripercorrere più volte la stessa fase.

Ogni fase è suddivisa in vari passi, che costituiscono una vera e propria check-list per verificare il completamento della fase. TOGAF indica anche quali documenti dovrebbero essere prodotti a conclusione di ciascun passo.

Sono previste frequenti validazioni dei risultati in raffronto ai requisiti (sia dell'intero ciclo ADM che di ogni singola fase). Queste valutazioni dovrebbero riconsiderare lo scope, i dettagli, le milestones e gli asset prodotti da ogni fase.

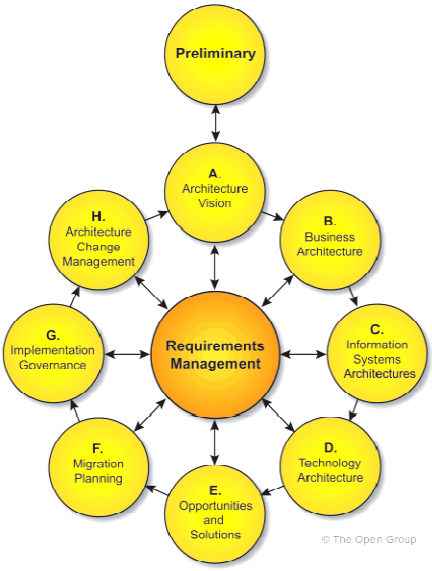


Figura 4.6. Il processo ADM e le sue fasi (vedi [6])

Nel seguito analizziamo nel dettaglio le singole fasi.

Preliminary

In questa fase si pongono le basi per affrontare con successo il ciclo ADM.

Tra gli obiettivi di questa fase riportiamo: l'individuazione degli stakeholder, i loro requisiti e le loro priorità, la conferma del mandato dagli stakeholder, l'identificazione delle organizzazioni coinvolte, la definizione dei vincoli e degli assunti. Questo è particolarmente importante nelle grandi organizzazioni, dove possono essere presenti ambiente architetturali federati.

Nella fase preliminare inoltre si costituisce il gruppo degli Enterprise Architects e si definiscono il framework e le metodologie dettagliate che verranno usate per sviluppare l'Enterprise Architecture (questo passo costituisce tipicamente un adattamento del ciclo ADM alla specifica organizzazione).

Infine si definiscono i principi architetturali vincolanti, che devono essere ricavati dai requisiti del business.

Se l'azienda ha scelto di adottare lo stile architetturale SOA, in questa fase è importante stabilire il principio architetturale di *Service-Orientation.*

Requirements Management

Il processo di gestione dei requisiti si applica a tutte le fasi del ciclo ADM. Ogni passo del processo ADM di TOGAF è basato sulla raccolta e sulla validazione dei requisiti di business. I requisiti sono identificati e memorizzati, e viene data loro una priorità. Ogni fase del ciclo ADM deve soddisfare i relativi requisiti. È particolarmente importante la capacità di gestire il cambiamento dei requisiti, per costruire un'architettura flessibile in grado di fare da ponte tra le aspirazioni degli stakeholders e ciò che può essere realmente realizzato.

A. Architecture Vision

Durante questa fase vengono impostati lo scope, i vincoli e le aspettative del processo TOGAF di definizione dell'architettura. Si crea la visione architetturale (cioè si definisce l'architettura che si vuole ottenere), si definiscono gli stakeholders, si validano i principi di business e si crea lo statement di lavoro architetturale. Infine si ottiene l'autorizzazione formale a procedere con il progetto.

B. Business Architecture

Durante questa fase viene analizzata la *Baseline Business Architecture*, cioè l'attuale architettura di business, che comprende il disegno dei processi di business con appositi strumenti di modellazione e la definizione delle entità di business coinvolte, e viene sviluppata la *Target Business Architecture*, cioè l'architettura di business che si vuole ottenere, nella quale i processi sono rivisti e ottimizzati in base ai requisiti raccolti. Viene inoltre eseguita la *gap analysis*, cioè vengono individuati i punti di intervento per portare il business dalla *Baseline Architecture* alla *Target*

Architecture, e viene individuata la *roadmap*, cioè l'insieme di attività e la loro sequenza, per consentire al business di raggiungere la situazione finale desiderata.

Vengono selezionati gli *architecture viewpoints*[5] adatti per mostrare come i requisiti degli stakeholders siano soddisfatti dalla Target Business Architecture.

Viene infine analizzato l'impatto di questi cambiamenti nell'architettura complessiva.

C. Information Systems Architecture

In questa fase viene analizzato il sistema IT dell'organizzazione, comprese le informazioni (dati) più rilevanti che vengono gestite e le applicazioni utilizzate per gestirle.

Questa fase si compone di due passi, che possono essere svolti parallelamente oppure in sequenza:

- *Data Architecture* − definisce i tipi di dati necessari per supportare il business, li struttura e rappresenta in un modo che sia comprensibile agli stakeholders. Anche in questo caso si procede individuando la *Baseline Data Architecture*, che comprende la definizione delle principali entità dati utilizzate dal business, e si sviluppa la *Target Data Architecture*, in cui vengono introdotte nuove entità oppure vengono riorganizzate le entità esistenti, vengono ridefiniti i rapporti tra di esse, ecc.
- *Application Architecture* − definisce quali applicazioni gestiscono le entità dati, come sono utilizzati i dati e le applicazioni dai processi di business, che livello di integrazione è richiesto e quali dipendenze esistono tra le applicazioni.

In entrambi i casi viene eseguita la *gap analysis*, cioè vengono individuati i punti di intervento per portare il business dalla *Baseline Architecture* alla *Target Architecture*, e viene individuata la *roadmap*, cioè la sequenza di attività necessarie per raggiungere la situazione finale desiderata. Viene infine analizzato l'impatto di questi cambiamenti nell'architettura complessiva.

D. Technology Architecture

L'obiettivo di questa fase è lo sviluppo di una *Target Technology Architecture* che getti le basi tecnologiche per la successiva fase di implementazione e migrazione.

In questa fase vengono definite le piattaforme tecnologiche, le specifiche e gli standard di comunicazione, il dimensionamento dell'infrastruttura, ecc.

Anche in questo caso si esegue la *gap analysis* che evidenzia il divario tra la *Baseline Architecture* e la *Target Architecture*, si definisce la *roadmap* e si analizza l'impatto di questi cambiamenti nell'architettura complessiva.

---

[5] L'architettura software è comunemente descritta tramite *views* (viste), che esprimono l'architettura secondo una certa visuale (p. es. dal punto di vista del business o da quello tecnologico). Un *viewpoint* costituisce la specifica di come costruire e utilizzare la vista, descrive le notazioni, le tecniche di modellazione e di analisi, le convenzioni, ecc. da utilizzare in una vista, per mantenere la coerenza con le altre viste.

E. Opportunities and Solutions

Questa è la prima fase legata direttamente con l'implementazione. In questa fase si produce un piano iniziale di implementazione, vengono consolidati i requisiti emersi nelle fasi precedenti del ciclo ADM, vengono definite le soluzioni da implementare, si valuta l'opportunità di soluzioni acquisite dall'esterno, sono definiti i principali pacchetti di lavoro e sono identificati i passi della trasformazione verso la nuova architettura.

F. Migration Planning

Si definisce la transizione dalla situazione attuale (*baseline*) a quella futura (*target*) definendo nel dettaglio il piano di implementazione e migrazione (*roadmap*). Per ottenere questo, viene attribuito il valore per il business di ciascun progetto previsto nella *roadmap*, vengono stimate le risorse necessarie per realizzarlo e ne viene data la priorità tramite una valutazione costi/benefici e un'analisi dei rischi.

G. Implementation Governance

Questa fase fornisce la sorveglianza architetturale alla implementazione. Assicura che i progetti di implementazione siano conformi all'architettura di riferimento. Durante l'implementazione e al termine dei lavori, sono previste revisioni da parte degli Enterprise Architects per assicurare la compatibilità con l'architettura di riferimento. In questa fase sono previste anche altre attività come p. es. la verifica del corretto avanzamento dei progetti, del raggiungimento degli obiettivi in maniera conforme alle aspettative, della corretta installazione e messa in esercizio dei sistemi, nonché la documentazione della nuova architettura.

H. Architecture Change Management

Fornisce il continuo monitoraggio dell'architettura risultante dal ciclo ADM, verifica che sia allineata alle esigenze del business e ne controlla le performance, individua punti di miglioramento ed eventualmente fa proposte per modificare elementi non adeguati. Assicura che l'architettura risponda alle necessità dell'azienda e massimizza il valore dell'architettura per il business.

4.2.3 TOGAF in Provincia di Bolzano

Il framework TOGAF è generico e richiede di essere adattato e specializzato in base alle esigenze della propria impresa. In Provincia di Bolzano, per esempio, le diverse fasi del processo ADM di TOGAF sono state attribuite a ruoli differenti della struttura IT.

Nella figura seguente è illustrato il processo ADM in Provincia di Bolzano, e sono evidenziati i ruoli coinvolti nelle varie fasi del processo di evoluzione architetturale, in occasione della realizzazione di una nuova soluzione IT.

Sono previste tre fasi:

- una fase concettuale, nella quale vengono raccolti ed elaborati i requisiti di business e l'architettura IT nei rispettivi domini architetturali. Questa fase è fondamentale per garantire che la nuova soluzione soddisfi i requisiti degli utenti portando valore aggiunto all'azienda, e nel contempo sia integrabile senza soluzione di continuità con l'architettura IT esistente; in questa fase sono coinvolti prevalentemente gli Enterprise Architects, i Demand Manager e i Supply Manager.
- Una fase implementativa, nella quale viene realizzato quanto definito a livello concettuale nella prima fase; in questa fase sono coinvolti prevalentemente i Solution Architects, gli analisti, gli sviluppatori e i Project Manager.
- Una fase operativa, nella quale la soluzione IT realizzata è mantenuta in esercizio e offerta agli utenti come servizio; in questa fase sono coinvolti prevalentemente i tecnici delle reti, i sistemisti, i database administrators, i tecnici del supporto e gli amministratori degli application server.

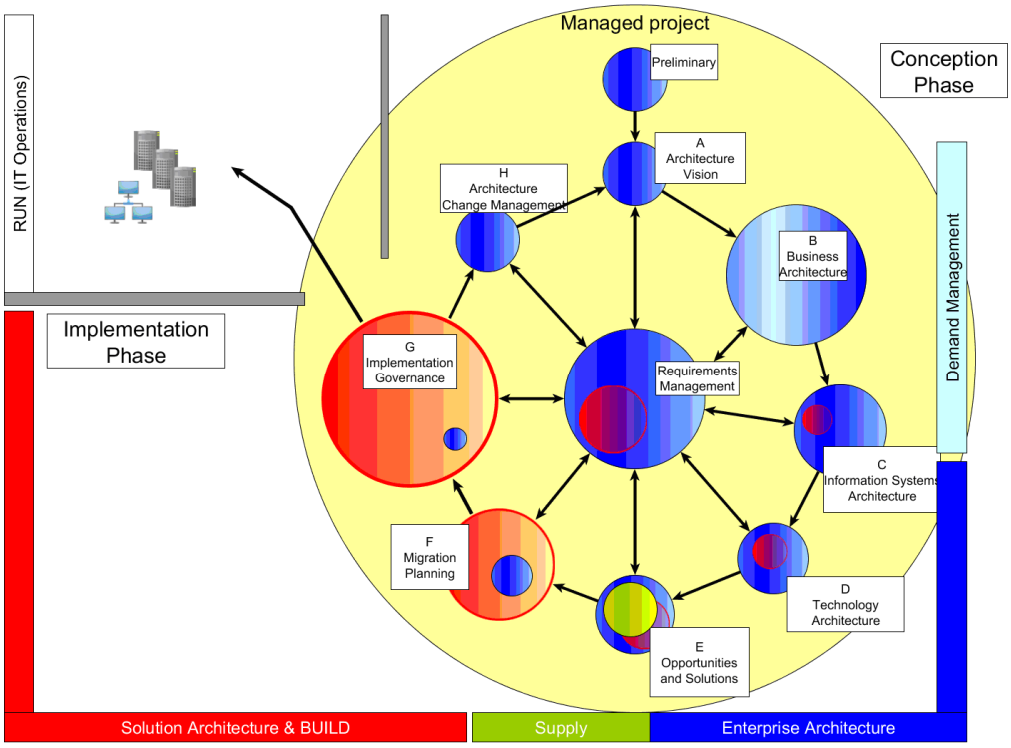


Figura 4.7. L'evoluzione dell'architettura IT nei progetti in Provincia di Bolzano

In conclusione, ricordiamo che il framework TOGAF è pensato per creare architetture, non applicazioni, ed ha il grande pregio di definire un processo ed un vocabolario comune. Per contro però, è un framework molto vasto e generico, che va di volta in volta personalizzato e "ritagliato su misura" per le proprie esigenze (*tailoring*).

Il consulente del Cutter Consortium Michael Rosen raccomanda di conoscere TOGAF, usarne il vocabolario, incorporare nella propria realtà le idee del flusso di processo architetturale e il modo standard di pensare e organizzare le architetture, ma di non applicarlo alla propria impresa *out of the box*, senza averlo prima adattato e personalizzato.

4.3 ArchiMate®

ArchiMate è un linguaggio di modellazione aperto ed indipendente per l'Enterprise Architecture, che fornisce gli strumenti per mettere in condizione gli Enterprise Architects di descrivere, analizzare e visualizzare le relazioni tra i domini architetturali in modo chiaro ed evidente.

Proprio come un disegno architettonico nell'architettura classica descrive i vari aspetti della costruzione e dell'uso di un edificio, ArchiMate offre un linguaggio comune per descrivere la costruzione e la gestione di processi aziendali, strutture organizzative, flussi informativi, sistemi IT e infrastruttura tecnica. Questa rappresentazione visiva aiuta i soggetti interessati comunicare, progettare le architetture e a valutare gli effetti delle decisioni sui domini aziendali.

ArchiMate è stato sviluppato nei Paesi Bassi da un team di progetto del Telematica Instituut[6], in collaborazione con diversi partner della pubblica amministrazione, dell'industria e del mondo accademico.

Lo sviluppo del linguaggio è durato dal 2002 al 2004 ed ha richiesto circa 35 anni/uomo e 4 milioni di Euro (finanziati in parte dal governo dei Paesi Bassi e in parte da partner privati).

Nel 2008 la proprietà e la gestione di ArchiMate sono state trasferite all'Open Group. Attualmente è gestito dall'ArchiMate Forum interno all'Open Group e le specifiche più recenti sono relative alla versione 2.1.

Il linguaggio ArchiMate è supportato da diversi produttori di tools per la modellazione, tra cui ricordiamo Archi (archimatetool.com) ed Enterprise Architect (sparxsystems.com.au), entrambi in uso nell'amministrazione provinciale.

4.3.1 La modellazione dell'Enterprise Architecture

Abbiamo visto che l'Enterprise Architecture consente un approccio completo e integrato ai problemi, che coinvolge tutti i domini dell'azienda, da quello di business a quello tecnologico. Purtroppo però ogni dominio parla il proprio linguaggio, disegna i propri modelli, usa tecniche e strumenti propri. Questo ha un effetto

[6] Telematica Instituut è un istituto olandese di ricerca nel settore IT fondato nel 1997. Nel 2009 il Telematica Instituut è stato riorganizzato e ha cambiato nome in Novay.

negativo sulla comunicazione all'interno dell'azienda e sui processi decisionali. La maggior parte dei linguaggi di modellazione inoltre è particolarmente adatta a modellare domini specifici, p. es. processi di business (BPMN) o architetture software (UML), ma la possibilità di modellare le relazioni di alto livello tra domini differenti è quasi completamente assente.

È necessario quindi un linguaggio di modellazione di alto livello, con il quale i differenti domini concettuali e le relazioni tra di essi possano essere descritti ad un livello sufficientemente astratto. Il linguaggio ArchiMate ha queste caratteristiche.

Si tratta pertanto di un linguaggio adatto a modellare ad alto livello l'architettura dell'intera azienda, cioè l'Enterprise Architecture.

Per la modellazione dettagliata delle architetture dei singoli domini possono essere utilizzati linguaggi di modellazione specifici, standard o proprietari (p. es. UML o BPMN).

Attualmente ArchiMate copre i domini di Business, Applications, Technology, Motivation, Implementation and Migration. Un ruolo importante nella definizione delle relazioni tra i diversi livelli è dato dai servizi che ciascun livello offre agli altri.

4.3.2 La grammatica di ArchiMate

ArchiMate è stato creato con l'obiettivo di essere facile da imparare e capire anche per gli utenti meno tecnici, pertanto il suo insieme di concetti è limitato.

Il linguaggio è costituito da tre tipi di oggetti:

- oggetti che agiscono (*strutturali attivi*)
- oggetti che rappresentano il comportamento degli oggetti attivi (*oggetti comportamentali*)
- oggetti che subiscono l'azione degli oggetti attivi (*strutturali passivi*)

Facendo una similitudine con il linguaggio naturale, l'elemento strutturale attivo è il soggetto, che compie un'azione (comportamento) nei confronti di un oggetto (elemento strutturale passivo).

In secondo luogo, il linguaggio distingue tra *vista esterna* e *vista interna* del sistema.

Questo rispecchia i principi di *service-orientation* visti in precedenza: il concetto di servizio rappresenta un'unità funzionale autonoma ed indipendente, che un sistema espone verso l'esterno. Per gli utenti esterni al sistema sono rilevanti solo le funzionalità, offerte come servizi, unitamente ad aspetti non funzionali come la qualità del servizio, i costi ecc. I servizi sono accessibili attraverso interfacce, che costituiscono la vista dall'esterno sugli aspetti strutturali.

Nella figura sottostante è mostrato un esempio di quanto esposto sopra: gli oggetti in azzurro rappresentano gli elementi attivi mentre gli oggetti gialli rappresentano l'aspetto comportamentale. Infine l'oggetto verde rappresenta l'elemento passivo.

Nel caso specifico riportato in figura, è rappresentata una generica applicazione che accede in scrittura ad un data store. La parte inferiore del diagramma rappresenta la parte interna dell'applicazione e i dati. La parte superiore rappresenta come l'applicazione è vista e utilizzata dall'esterno.

L'applicazione è composta da 4 elementi, anche se ovviamente è possibile semplificare la rappresentazione. Il numero di elementi impiegati per rappresentare un concetto dipende sempre dal livello di dettaglio che si vuole ottenere. La rappresentazione di un'applicazione riportata in figura è molto dettagliata e mostra tutti gli elementi che la costituiscono, ma, come si vedrà in seguito, solitamente l'obiettivo è di mostrare i legami tra tutti i layer dell'Enterprise, e pertanto raramente è necessario modellare un'applicazione utilizzando questo livello di dettaglio.

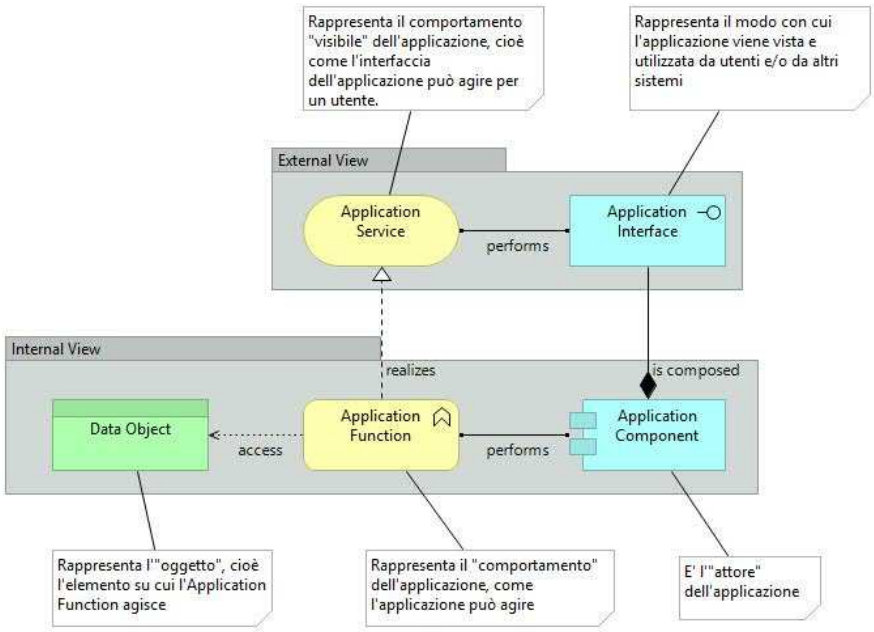


Figura 4.8. Gli elementi fondamentali del linguaggio ArchiMate (vedi [9])

4.3.3 I layer del linguaggio ArchiMate

ArchiMate definisce tre layer architetturali principali:

1. *Business layer* − contiene gli elementi necessari per offrire a clienti esterni prodotti e servizi, realizzati da processi di business dell'organizzazione, eseguiti da attori e ruoli di business.
2. *Application layer* − contiene gli elementi necessari per supportare lo strato di business con i servizi applicativi, che sono realizzati da componenti applicativi.
3. *Technology layer* − contiene gli elementi da utilizzare per offrire servizi infrastrutturali (p. es. storage, sistemi, reti) necessari per eseguire le applicazioni.

Oltre a questi layer, ArchiMate introduce anche le estensioni *Motivation Layer* e *Implementation and Migration Layer*, così da rendere ancora più evidenti le analogie tra TOGAF ed ArchiMate.

Nella figura seguente sono mostrate le corrispondenze tra i layer di ArchiMate e le fasi del ciclo ADM di TOGAF.

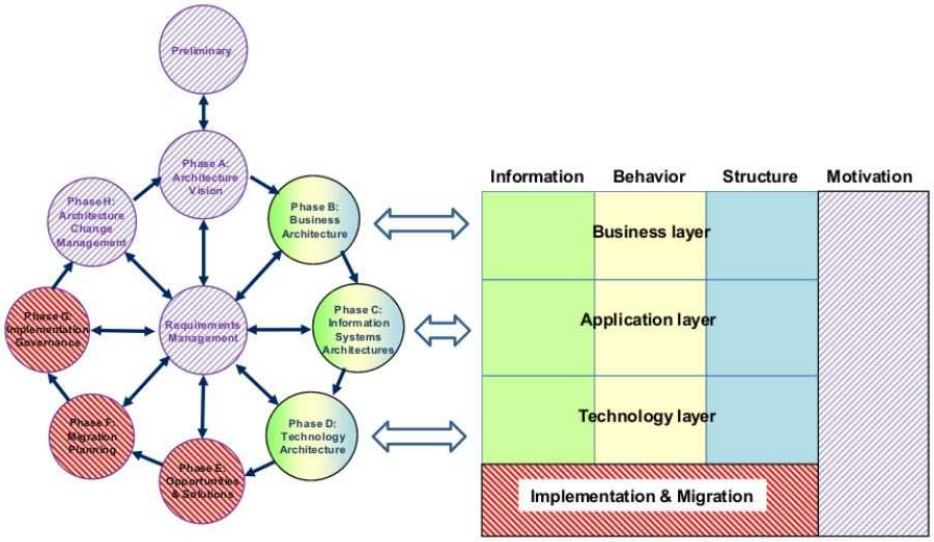


Figura 4.9. Corrispondenze tra ArchiMate e TOGAF (vedi [10])

Ciascun layer comprende un insieme di elementi che lo caratterizzano (come p. es. quelli visti nella figura 4.8, che appartengono tutti all'Application Layer).

ArchiMate non ha un codice standardizzato per i colori. Storicamente il blu è stato usato per gli elementi attivi, il giallo per quelli comportamentali e il verde per quelli passivi. In seguito però è stato introdotto uno schema di colori giallo/blu/verde per distinguere i layer di business, application e infrastructure.

È quindi possibile utilizzare schemi di colori personalizzati: quello utilizzato nella figura precedente, prodotto da Gerben Wierda [9], è uno schema a 9 colori che migliora la rappresentazione delle viste costituite da un singolo layer, che altrimenti sarebbero monocromatiche, consentendo di distinguere a colpo d'occhio attori e comportamenti; le lievi differenze di intensità del colore rendono più leggibili anche le viste multi-layer.

Di seguito è rappresentata una panoramica degli elementi presenti nei diversi layer di ArchiMate.

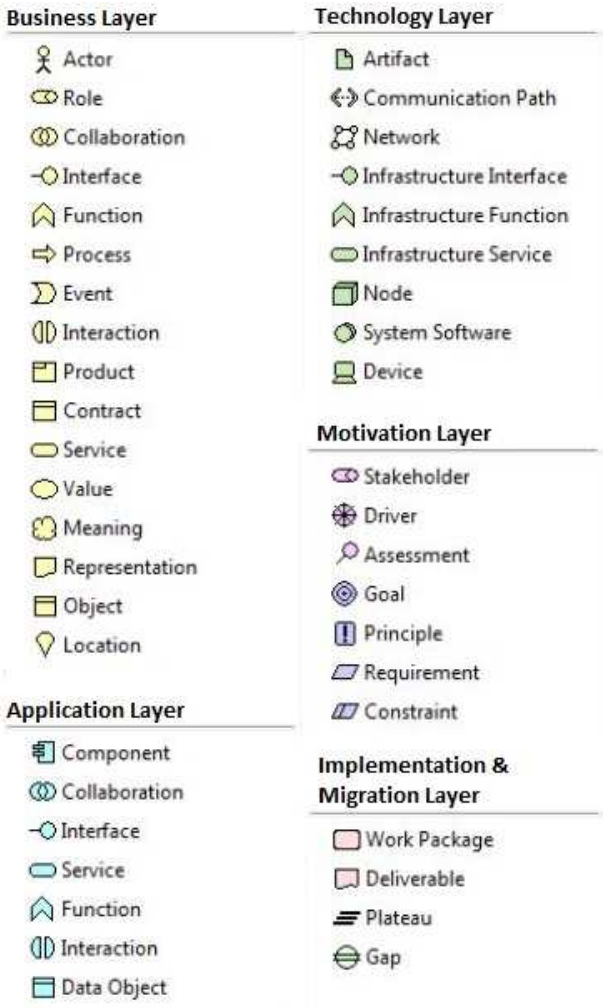


Figura 4.10. Gli elementi di ArchiMate

Analizziamo ora più nel dettaglio i tre layer principali:

Business Layer:

i principali concetti attivi a questo livello sono l'attore e il ruolo, i quali eseguono comportamenti come le funzioni e i processi.

Le funzioni di business rappresentano le capacità di alto livello di un'organizzazione, e offrono funzionalità che possono essere utilizzate in processi di business per realizzare i prodotti e servizi. I processi di business solitamente vengono innescati da eventi e possono produrre oggetti di business.

Il comportamento di un processo aziendale visibile dall'esterno è modellato tramite il concetto di servizio di business, che rappresenta un'unità di funzionalità significativa.

I servizi di business possono essere a loro volta raggruppati per formare prodotti (p. es. un prodotto bancario è un insieme di servizi bancari), insieme ad un contratto che ne specifica le caratteristiche associate.

Nella figura seguente è riportato un esempio di un processo di business relativo ad una compagnia di assicurazioni, che utilizza gli elementi sopra descritti, appartenenti al Business Layer.

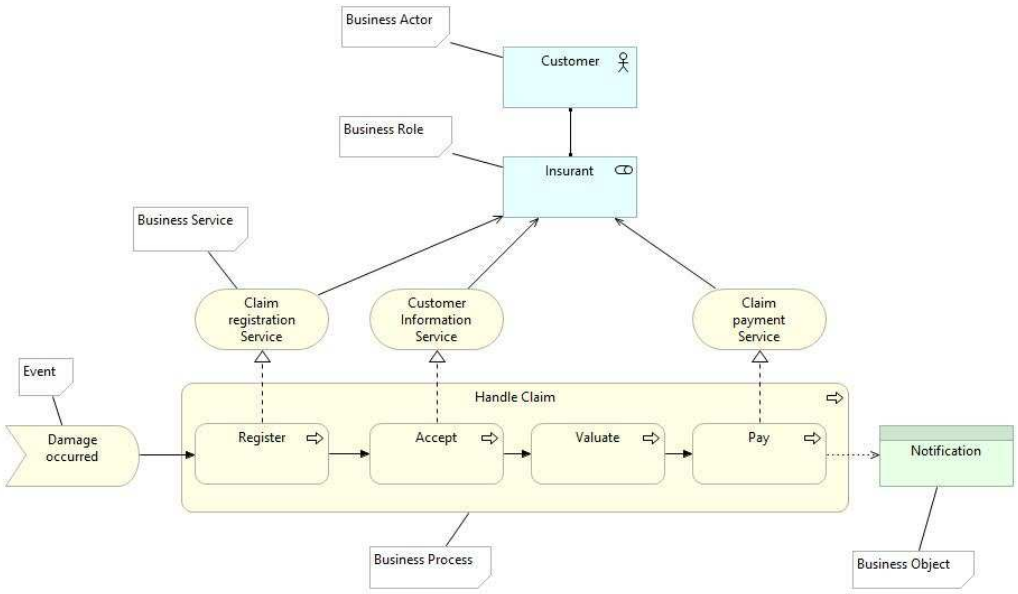


Figura 4.11. Esempio di processo di business con ArchiMate (vedi [12])

Application Layer:

l'elemento attivo principale dell'Application Layer è l'Application Component, che può essere usato per modellare qualsiasi componente software, applicazione o sistema informativo. Il comportamento, in modo molto simile al layer di business, è dato dall'Application Service nella vista esterna e dall'Application Function che realizza il servizio, nella vista interna. I servizi sono offerti tramite interfacce applicative. Per un esempio di modello applicativo si può fare riferimento alla figura 4.8.

Technology Layer:

L'elemento attivo più importante è il nodo, che può essere un dispositivo o un *system software* (quello che in UML è chiamato *execution environment*, p. es. un sistema operativo, un application server, ecc.).

L'elemento comportamentale, in modo molto simile all'Application Layer, è dato dall'Infrastructure Service nella vista esterna e dall'Infrastructure Function che realizza il servizio, nella vista interna. L'Infrastructure Interface è il punto attraverso il quale i servizi infrastrutturali offerti dal nodo possono essere messi a disposizione di altri nodi o di componenti applicativi dell'Application Layer.

L'*artifact* è un blocco di informazione usato o prodotto nel processo di sviluppo software.

Il *network* modella la connessione fisica tra due o più dispositivi.

Nella figura seguente è riportato un esempio di un modello tecnologico che utilizza gli elementi sopra descritti.

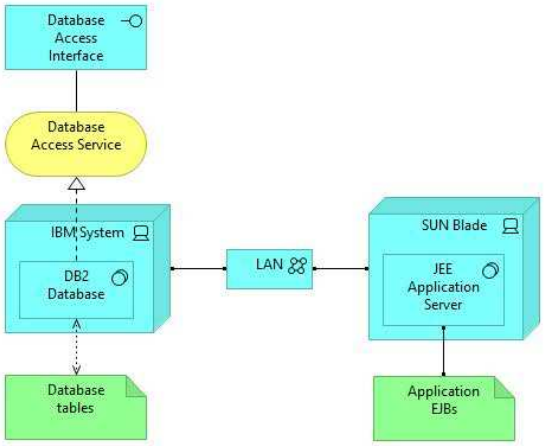


Figura 4.12. Esempio di tecnology model con ArchiMate (vedi [12])

4.3.4 Le relazioni tra i layer

Come in UML, anche in ArchiMate le relazioni tra elementi hanno un significato specifico e una tassonomia ben precisa.

Le relazioni si dividono in:

- strutturali
- dinamiche
- altre relazioni

L'associazione tra elementi e relazioni non è arbitraria, ma esiste una matrice che specifica le possibili associazioni tra elementi e relazioni.

Di seguito sono elencate le relazioni previste da ArchiMate, ciascuna con il suo significato.

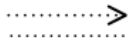
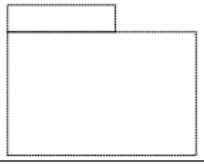
Structural Relationships		**Notation**
Association	Association models a relationship between objects that is not covered by another, more specific relationship.	
Access	The access relationship models the access of behavioral concepts to business or data objects.	
Used by	The used by relationship models the use of services by processes, functions, or interactions and the access to interfaces by roles, components, or collaborations.	
Realization	The realization relationship links a logical entity with a more concrete entity that realizes it.	
Assignment	The assignment relationship links units of behavior with active elements (e.g., roles, components) that perform them, or roles with actors that fulfill them.	
Aggregation	The aggregation relationship indicates that an object groups a number of other objects.	
Composition	The composition relationship indicates that an object is composed of one or more other objects.	
Dynamic Relationships		**Notation**
Flow	The flow relationship describes the exchange or transfer of, for example, information or value between processes, function, interactions, and events.	
Triggering	The triggering relationship describes the temporal or causal relationships between processes, functions, interactions, and events.	
Other Relationships		**Notation**
Grouping	The grouping relationship indicates that objects, of the same type or different types, belong together based on some common characteristic.	
Junction	A junction is used to connect relationships of the same type.	
Specialization	The specialization relationship indicates that an object is a specialization of another object.	

Tabella 4.2. Relazioni previste da ArchiMate (vedi [10])

Riportiamo di seguito un semplice esempio di come con ArchiMate possono essere rappresentati in un'unica vista tutti i domini architetturali con le relative relazioni.

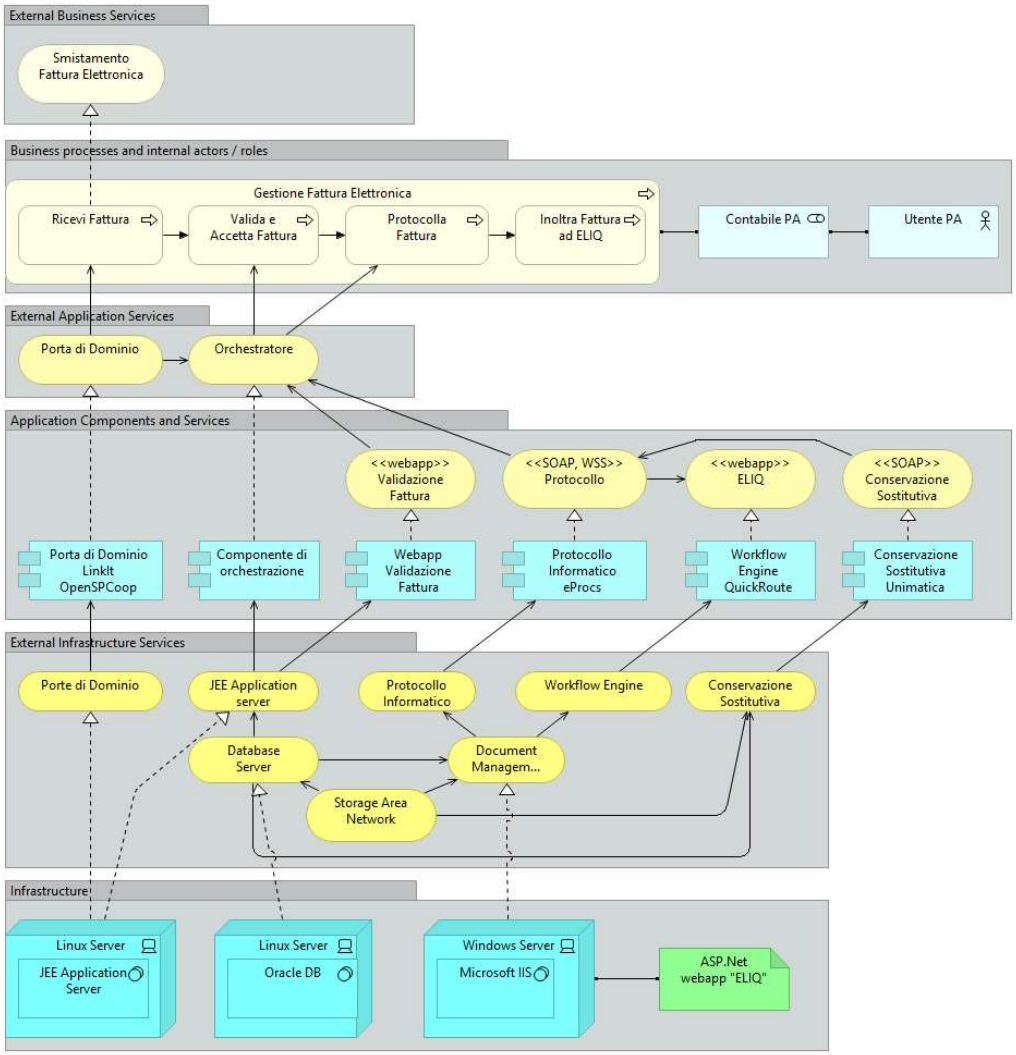


Figura 4.13. Architettura Enterprise per la gestione della fattura elettronica con ArchiMate

ArchiMate fornisce quindi una visione Enterprise del modello architetturale, integrata e orientata ai servizi, ottenuta connettendo i layer differenti tramite i servizi esposti da ciascun layer. Il principale vantaggio di questa visione unica è quello di poter valutare con immediatezza l'impatto complessivo di un cambiamento in un dominio, e di riuscire a far comprendere in modo semplice le conseguenze del cambiamento agli stakeholder, anche non tecnici.

4.4 Conclusioni

Il presente lavoro di tesi si è posto l'obiettivo di analizzare e documentare il significativo percorso di riorganizzazione dell'IT, fatto dalla Provincia autonoma di Bolzano per poter offrire servizi qualitativamente elevati ai propri collaboratori, ai cittadini e alle imprese in un contesto di costante riduzione delle risorse, e rispondere in modo rapido ed efficace alle richieste del business.

L'analisi si è concentrata in particolar modo sulla scelta di uno stile architetturale orientato ai servizi (SOA), sull'impiego di un framework per la realizzazione di soluzioni IT orientate ai servizi (COSM), e sull'adozione dell'Enterprise Architecture, una disciplina adatta ad affrontare in modo completo e integrato le problematiche architetturali dell'azienda a tutti i livelli, dal business alla tecnologia.

A supporto del lavoro del gruppo Enterprise Architects della Provincia, del quale il laureando fa parte, sono stati inoltre individuati il framework TOGAF e il linguaggio di modellazione ArchiMate.

L'architettura SOA della Provincia è operativa ormai da alcuni anni, seppur in costante evoluzione, e sono stati realizzati molti progetti che ne fanno uso, sia in ambito e-Government che di back office. A titolo di esempio si possono citare tra gli altri: il sistema informativo agricolo e forestale LAFIS, il sistema di gestione contributi per l'industria, il commercio e l'artigianato GC35, l'iscrizione on-line a scuola (IOLE), all'esame di bilinguismo e nelle liste di disoccupazione (EJOB-OLE), la gestione dei corsi di formazione, il fascicolo personale digitale dei dipendenti provinciali. Questa architettura si è rivelata inoltre adatta a supportare il sistema informativo scolastico LASIS, che coinvolge le circa 150 scuole dislocate sul territorio e collegate alla sede centrale con linee di rete inadeguate, e che richiede pertanto architetture software distribuite, ridondate e replicate.

Un ulteriore esempio del lavoro svolto dalla Provincia in direzione della Service-Orientation, al quale il laureando ha partecipato attivamente, è il servizio di Protocollo Informatico a norma di legge, attualmente utilizzato da circa 20 applicazioni basate su tecnologie differenti (Java, Delphi, .Net, ma anche Oracle Forms e Microsoft Access attraverso un wrapper COM).

L'adozione di un'architettura orientata ai servizi ha dimostrato pertanto nel corso del tempo la sua validità, visto anche che questa filosofia è stata largamente seguita dalle architetture web che hanno dominato il mercato nell'ultimo decennio.

In futuro la richiesta di servizi online per i cittadini e per le imprese, accessibili da qualsiasi dispositivo, sarà sempre maggiore, e l'impiego di un'architettura orientata ai servizi consentirà di sviluppare applicazioni già predisposte per questi requisiti, garantendo un risparmio di tempi e costi di sviluppo e la possibilità di rispondere velocemente ed efficacemente alle richieste del business con applicazioni nativamente integrate nell'architettura di riferimento.

Anche la **standardizzazione** garantita dalla generazione automatica del codice permette di aumentare la qualità del software e di facilitare enormemente la

manutenzione e la risoluzione degli errori, e consente agli sviluppatori di tralasciare lo sviluppo di codice infrastrutturale e concentrarsi sulle funzionalità che danno valore al business.

La quantità di applicazioni che entrano a far parte della mappa applicativa della Provincia di Bolzano, ciascuna con propri requisiti di business e con diverse esigenze tecnologiche, è sostenuta dall'architettura orientata ai servizi, che è in grado di supportare in maniera agile e flessibile queste richieste.

Un ultimo importante fattore è dato dalla **riduzione dei costi**: l'industrializzazione del processo di sviluppo e la separazione delle responsabilità sono elementi che portano ad una diminuzione degli *skill* richiesti agli sviluppatori, con conseguente riduzione dei costi di sviluppo.

Un ulteriore valore aggiunto di questa architettura è il fatto di essere realizzata con componenti open source, e quindi è potenzialmente riutilizzabile da parte di qualsiasi pubblica amministrazione interessata, nell'ottica del riuso del software fortemente incentivato dal Codice dell'Amministrazione Digitale.

In qualità di Enterprise Architects, il nostro obiettivo nel prossimo futuro è quello di estendere ad un numero sempre crescente di progetti l'impiego dell'architettura di riferimento, per creare un ampio insieme di servizi riutilizzabili. È fondamentale a questo proposito che gli Enterprise Architects siano coinvolti nei progetti strategici, in modo che possano introdurre e mettere alla prova sul campo l'architettura di riferimento.

Il mondo dell'IT è però in costante evoluzione. Nascono continuamente nuove tecnologie, nuovi linguaggi di sviluppo e nuovi framework. È fondamentale quindi che l'Enterprise Architect sia sempre aggiornato sulle nuove tecnologie e sia in grado di valutare se e quando è opportuno integrarle nell'architettura.

Altrettanto importante è il costante monitoraggio dell'architettura di riferimento da parte degli architetti, al fine di verificare che sia sempre allineata con i requisiti del business, in conformità con quanto previsto nella fase H (*Architecture Change Management*) del ciclo ADM di TOGAF.

È infine di fondamentale importanza, per validare e diffondere l'architettura di riferimento, misurare il grado di **riutilizzo dei servizi** (uno dei fattori di successo di una SOA), e definire degli indicatori che consentano di dimostrare l'efficacia del contributo dell'Enterprise Architect nei progetti di successo e la validità delle scelte fatte.

Bibliografia

[1] Erl T., Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall, 2005.

[2] Herzum P., Sims O., Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise, Wiley Computer Publishing, 2000.

[3] Herzum P., E-Government Reference Architecture, PAB Internal documentation, 2007.

[4] PAB EA Group, Application Design Policies, PAB Internal documentation, 2013.

[5] Rosen M., 10 Key Skills Architects Must Have to Deliver Value, Cutter Consortium, 2008.

[6] The Open Group, TOGAF™ Version 9, Van Haren Publishing, 2010.

[7] Josey A., TOGAF™ Version 9.1 Enterprise Edition, An Introduction, The Open Group, 2011.

[8] The Open Group, Using TOGAF to Define and Govern Service-Oriented Architectures, The Open Group, 2011.

[9] Wierda G., Mastering ArchiMate, R&A, 2014.

[10] The Open Group, ArchiMate® 2.0 Specification, The Open Group, 2012.

[11] Keller W., TOGAF 9.1 Quick Start Guide for IT Enterprise Architects, 2009, *available from: www.objectarchitects.biz/TOGAF9/TOGAF_9_1_Quickstart_(V0_9).pdf*

[12] Lankhorst M., Van Drunen H., Enterprise Architecture Development and Modelling, 2007, *available from: vianovaarchitectura.nl/page/enterprise-architecture-development-and-modelling*