UNIVERSITÀ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Un framework per lo sviluppo di sistemi informativi di diagnostica in ambiente IHE

Relatore: prof. Ferrari Carlo
Correlatore: Dott. Cecchin Diego
Correlatore: Paolo Mosca

Laureando: Piva Alberto

7 marzo 2012

Indice

Introduzione

L'informatica è entrata prepotentemente nella Sanità ormai da più di 20 anni, ma l'informatica nella Sanità non è semplicemente il computer in medicina. L'Informatica Medica è la scienza che si occupa della gestione in Sanità dell'informazione e dei programmi basati su calcolatore

Questa scienza è ormai parte integrante di molti processi e procedure anche delle discipline di Radiologia e Medicina Nucleare, appartenenti al dominio più generale della diagnostica per immagini.

In particolare, in queste branche sono da tempo affermati sistemi noti come RIS, ovvero *Radiology Information Sistem*, Sistema Informativo Radiologico. Un RIS è un sistema informatico usato in Radiologia per la gestione del flusso dei dati inerenti i pazienti trattati.

A partire da metà anni novanta, presso il *Dipartimento di Scienze Medico Diagnostiche e terapie speciali* dell'Università di Padova, sezioni di Radiologia e Medicina Nucleare, sono stati sviluppati internamente sistemi informatici adeguati a rispondere alle esigenze specifiche interne e realizzati secondo standard e linee guida internazionali, fino ad ottenere due sistemi noti come MARiS e MN1, quest'ultimo frutto della scissione di MARiS, ormai qualche anno fa.

Col tempo, la manutenzione e lo sviluppo dei due sistemi, simili ma non uguali e realizzati secondo logiche di semplicità e immediatezza ma non di rapida collaudabilità, è diventato meno sostenibile.

In questo lavoro di tesi sono stati analizzati i due sistemi informativi per realizzare un nuovo framework che favorisse l'integrazione e riprogettazione graduale degli stessi.

In particolare, nel capitolo 1 sono introdotti l'informatica sanitaria, le discipline di Radiologia e Medicina Nucleare, i sistemi RIS e PACS in generale, le

organizzazioni inerenti agli standard internazionali usati in questi sistemi.

Nel capitolo 2 sono descritti i due particolari RIS in esame, la loro storia e caratteristiche principali e le motivazioni alla base dell'integrazione.

Nel capitolo 3 sono analizzati le basi di dati e il codice sorgente dei due RIS in esame, e valutati possibili framework esistenti per una loro eventuale adozione, e le motivazioni che hanno condotto a scartarli.

Nel capitolo 4 è illustrato il nuovo framework realizzato, a partire dalla fase di progettazione, passando per la base di dati, i componenti e il flusso di esecuzione.

Nel capitolo 5 è esposta l'analisi del nuovo framework comparativamente ai sistemi RIS precedenti e a noti framework già disponibili, illustrando strumenti, metodologie e tecniche usate per l'analisi.

Nel capitolo finale sono tratte le conclusioni circa il presente lavoro di tesi, i risultati ottenuti e dei possibili ulteriori sviluppi.

Parte I

I due RIS esistenti

Capitolo 1

Sistemi informativi radiologici

Introduzione

L'informatica è entrata prepotentemente nella Sanità ormai da più di 20 anni, e ha ormai assunto un'indiscussa importanza strategica. La progettazione e implementazione di nuovi processi sanitari che non considerino la scienza dell'informazione come propria parte fondante non ha più possibilità di realizzazione.

Tuttavia l'informatica nella Sanità non è semplicemente il computer in medicina, ma racchiude una moltitudine di questioni e problematiche insite nel particolare dominio di applicazione, quello sanitario appunto.

Questo capitolo introduce l'informatica nel dominio medico in generale, quindi entra nel dettaglio delle discipline di Radiologia e Medicina Nucleare, degli enti internazionali e degli gli standard usati nel settore.

1.1 L'informatica medica

L'informatica nella sanità e nella ricerca medica costituisce una rivoluzione epocale realizzatasi negli ultimi 20 anni. In Sanità, le tecnologie informatiche, fino a pochi anni fa destinate solo all'automazione dei processi amministrativi, vengono sempre più utilizzate per:

- migliorare il processo di cura dei pazienti in fase acuta
- migliorare il processo riabilitativo

- migliorare l'assistenza ai malati cronici
- facilitare la prevenzione di malattie
- aumentare l'efficacia e l'efficienza del lavoro quotidiano degli operatori sanitari

Tutte queste attività sono legate in modo diretto al processo di cura, perciò in Sanità l'informatica sta diventando anche uno strumento clinico, e in Medicina si costituisce a naturale strumento di ricerca.

Le esigenze della Sanità generale sono molteplici:

- controllare la spesa attraverso la razionalizzazione dei processi
- gestire i dati dello specifico paziente in modo razionale
- estrarre informazioni cliniche dalle enormi moli di dati disponibili sul paziente
- fornire supporto alla decisione medica

Posto che l'informatica è uno degli strumenti che la tecnologia offre alla pratica medica, dal lato medico chi opera a livello clinico nei sistemi sanitari deve saper sfruttare in modo efficiente ed efficace la tecnologia, mentre chi propone soluzioni tecnologiche deve avere conoscenza significativa di alcune delle problematiche della Sanità dal lato tecnologico. Ne consegue che la distanza fra sapere tecnologico e sapere medico deve essere ridotta.

L'informatica medica è la scienza che si occupa della gestione in Sanità dell'informazione e dei programmi basati su calcolatore, cioè l'applicazione dei principi della Teoria dell'Informazione alla conoscenza medica con l'obiettivo di fornire un supporto alla risoluzione delle problematiche della Scienza Medica attinenti alla diagnosi, alla terapia ed alla prevenzione. Lo scopo ultimo dell'informatica medica è contribuire (direttamente o indirettamente) a migliorare la salute del paziente; essa non è semplicemente il computer in medicina.

I mezzi usati dall'informatica medica sono:

- algoritmi per l'analisi dei dati e la loro elaborazione

- sistemi per la gestione dei dati clinici (include conservazione dei dati consistente, efficiente e sicura)
- sistemi di trasmissione/comunicazione/esportazione di dati clinici basati su metodologie informatiche e telecomunicazioni
- sistemi di supporto alla decisione clinica

1.2 Imaging diagnostico: Radiologia e Medicina Nucleare

Radiologia e Medicina nucleare sono due discipline mediche facenti parte del dominio più generale della diagnostica per immagini. Con i termini di imaging o imaging biomedico o diagnostica per immagini ci si riferisce al generico processo attraverso il quale è possibile esaminare un'area di un organismo non visibile dall'esterno.

Le tecniche di diagnostica per immagini sono numerose, tra le altre si ricordano l'endoscopia, l'ecografia, l'ecodoppler, l'ecografia con mezzo di contrasto, la radiografia, la spettroscopia, la tomografia computerizzata, l'imaging a risonanza magnetica, la fluoroscopia, l'angiografia, la mammografia, la tomografia ad emissione di positroni la tomografia ad emissione di fotone singolo, nonché le varie combinazioni tra queste, come RM-PET[1], piuttosto che eco-TC[2] o altro ancora. Alcune di queste tecniche consentono sia di effettuare diagnosi, sia di effettuare terapie.

1.2.1 Radiologia

La radiologia è la scienza che si occupa della produzione di raggi X e del loro utilizzo, diagnosticare e trattare le malattie attraverso l'utilizzo delle immagini [fig. 1.2] (vere, ricostruite o virtuali) dell'interno del corpo umano. La branca della medicina che si occupa della produzione e della lettura di immagini radiografiche è detta radiodiagnostica. La branca della medicina che si occupa della produzione di raggi X a scopo terapeutico è detta radioterapia.

[1] Risonanza Magnetica-Positron Emission Tomography
[2] Ecografia-Tomografia Computerizzata

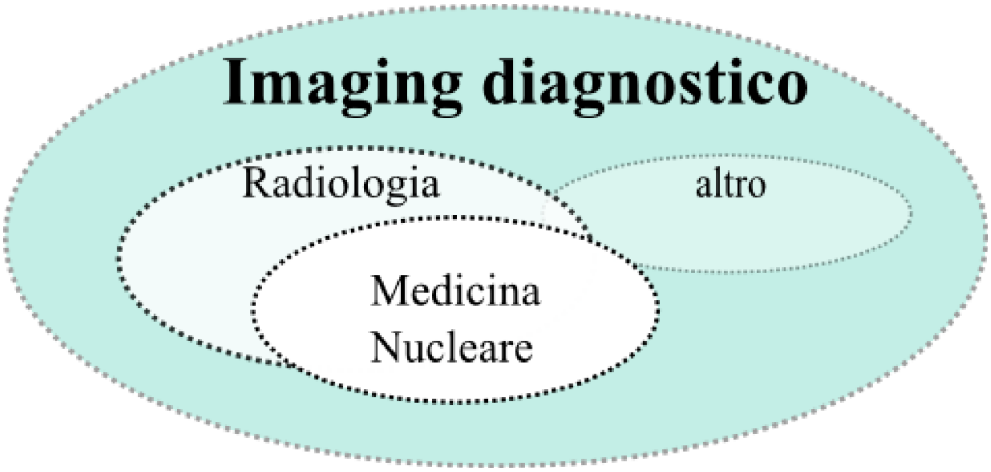

Figura 1.1: Dominio Imaging diagnostico

Negli ultimi decenni, attraverso una sua branca molto specialistica, la radiologia interventistica, essa provvede al trattamento di uno spettro sempre più ampio di patologie vascolari e non. La radiologia ha subito negli ultimi cinquanta anni uno sviluppo travolgente in ambito diagnostico e terapeutico ed occupa oggi un ruolo fondamentale ed imprescindibile nella diagnosi e nel trattamento della stragrande maggioranza delle patologie umane.

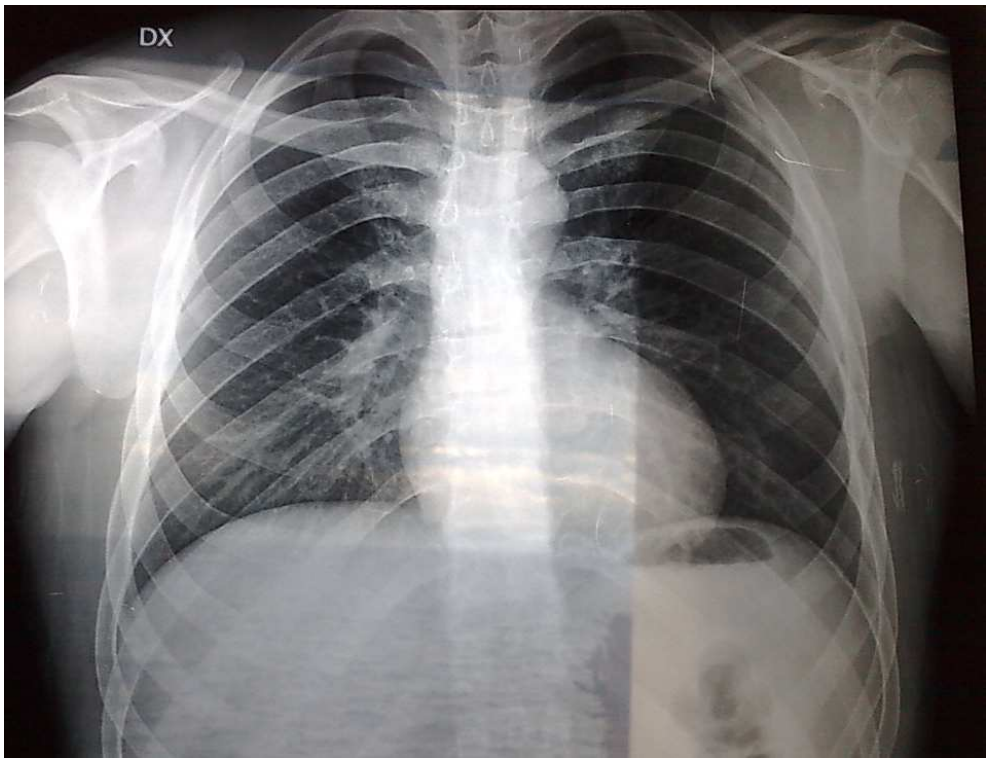

Figura 1.2: Esempio di radiogramma del torace

1.2.2 Medicina Nucleare

La Medicina Nucleare è la specialità che utilizza sorgenti radioattive non sigillate per la diagnosi e il trattamento di alcune patologie. Nelle procedure di Medicina Nucleare, radionuclidi [3] sono combinati con altri elementi per formare composti

[3] Isotopi radioattivi

chimici, oppure sono combinati con esistenti composti farmaceutici per formare radiofarmaci [4].

Questi radiofarmaci possiedono tropismo elettivo, ovvero una volta somministrati al paziente, sono in grado di focalizzarsi su specifici organi o recettori cellulari. Questa loro proprietà consente alla medicina nucleare di visualizzare l'estensione del processo di una malattia nel corpo, basandosi sulla funzione cellulare e fisiologia piuttosto che sui cambiamenti fisici nell'anatomia dei tessuti. In alcune malattie gli studi di medicina nucleare sono in grado anticipare la diagnosi rispetto ad altri test diagnostici.

La disciplina consente anche il trattamento di tessuti malati, basandosi sul metabolismo o sul legame di un particolare ligando [5], similarmente ad altre aree della farmacologia. In futuro, la medicina nucleare potrà anche fornire un nuovo impulso al settore noto come medicina molecolare. Sonde specifiche potranno essere sviluppate per permettere la visualizzazione, caratterizzazione, e la quantificazione di processi biologici a livello cellulare e subcellulare.

1.3 Sistemi RIS e PACS

Il Radiology Information System (RIS), ovvero Sistema Informativo Radiologico, è un sistema informatico usato in Radiologia per la gestione del flusso dei dati inerenti i pazienti trattati.

Le funzionalità del RIS sono molteplici, ma hanno lo scopo di gestire il processo di refertazione, cioè quella serie di azioni od eventi che portano dall'approccio del paziente con la struttura all'espletamento del referto, oggetto dell'indagine radiologica.

Il processo di integrazione informatica di un dipartimento di Radiologia nell'ambito dell'azienda ospedaliera orbita intorno al RIS, dal momento che tipicamente è il riferimento per il dialogo con:

- Sistemi gestionali ospedalieri

[4] Un qualsiasi medicinale che, quando è pronto per l'uso, include uno o più radionuclidi incorporati a scopo sanitario

[5] un atomo, ione o molecola che generalmente dona i suoi elettroni per formare un legame di coordinazione, agendo da base di Lewis

- Sistemi gestionali regionali
- Modalità diagnostiche
- Workstation di visualizzazione

Anche se nell'ottica aziendale è classificato come una delle tante applicazioni (client-server) da reparto, il RIS per ragioni di ordine storico legate al grande numero di prestazioni da gestire ha una lunga storia alle spalle,

In Radiologia quindi il ruolo di un RIS è centrale, un siffatto sistema permette di individuare e di eliminare colli di bottiglia all'interno del processo di refertazione, consente di rendicontare correttamente le attività effettuate, è ausilio indispensabile alla diagnosi grazie alla gestione delle Cartelle Radiologiche informatizzate.

In genearle un RIS è in grado di comunicare con altri sistemi informatici ospedalieri:

- comunicazioni Health Level 7 (HL7) con un sistema informativo ospedaliero,o con un Centro Unico di Prenotazione o con un archivio di referti aziendale
- comunicazione con i sistemi regionali[6]
- comunicazioni Digital Imaging and Communications in Medicine (DICOM) con le diagnostiche o con un Picture Archiving and Comunication System (PACS)
- integrazioni con visualizzatori DICOM

Nel prosieguo del capitolo sono approfonditi gli standard HL7 e DICOM.

PACS è un Sistema di Archiviazione e Trasmissione di Immagini. Un PACS consiste in un sistema hardware e software dedicato all'archiviazione, trasmissione, visualizzazione e stampa delle immagini diagnostiche digitali. Le immagini diagnostiche digitali[fig. 1.3] sono visualizzate su speciali monitor ad altissima risoluzione di client DICOM.

Una parte fondamentale ma non visibile dall'utente finale si occupa dell'interfacciamento con gli altri attori del flusso radiologico. In special modo,

[6] Ad esempio il SISS Veneto e il relativo progetto IESS

è fondamentale la sua integrazione con il sistema informatico radiologico, che rappresenta il software gestionale della Radiologia.

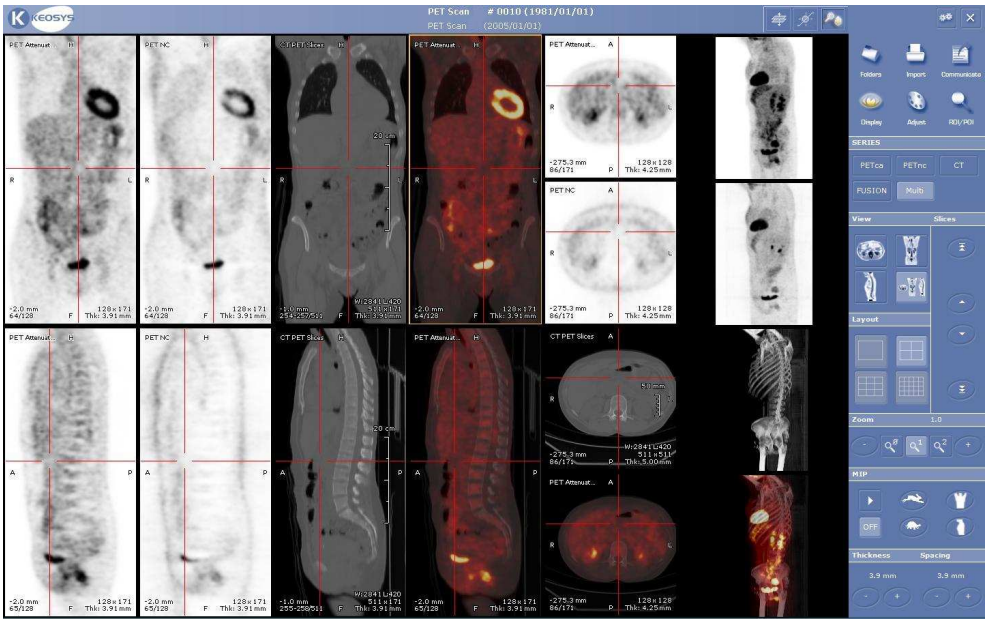

Figura 1.3: Immagine PET in un visualizzatore medicale

1.4 Organizzazioni ed enti

1.4.1 RSNA

Fondata nel 1915, la Radiological Society of North America, o RSNA, è una società di appartenenza professionale nord americana impegnata a favorire l'eccellenza nella cura del paziente attraverso l'istruzione e la ricerca. Più di 48.000 professionisti dell'imaging medico sono membri di RSNA, tra radiologi, oncologi di radiazione, fisici medici e scienziati alleati.

RSNA ospita la più grande riunione annuale di radiologia del mondo, pubblica due riviste altamente rispettate di valutazione tra pari e offre borse di ricerca e formazione per giovani ricercatori.

La mission di RSNA è promuovere e sviluppare i più elevati standard di radiologia e delle scienze correlate. La società mira a fornire a radiologi e scien-

ziati del settore sanitario programmi educativi e materiali di altissima qualità, per migliorare costantemente il contenuto e il valore di queste attività educative.

La Società si propone di promuovere la ricerca in tutti gli aspetti della radiologia e delle scienze correlate, ivi comprese la ricerca clinica di base nella promozione di un'assistenza sanitaria di qualità. La Società si propone di promuovere una più stretta comunione tra tutti i radiologi e una maggiore cooperazione tra radiologi e membri di altre branche della medicina e gli operatori sanitari.

Il successo della Società nel raggiungimento dei suoi obiettivi nel campo dell'istruzione e della ricerca è dovuto all'alto livello di professionalità dei suoi membri e gli altri colleghi che generosamente condividono le loro conoscenze scientifiche e capacità amministrative.

1.4.2 IHE

Integrating the Healthcare Enterprise (IHE) rappresenta un'iniziativa internazionale di produttori ed utenti a supporto dello sviluppo dell'integrazione tra sistemi informativi sanitari.

L'iniziativa IHE, nata negli Stati Uniti nel 1998 ad opera di Radiologica Society of North America (RSNA) e Healthcare Information and Management Systems Society (HIMSS), si propone di definire in maniera chiara come gli standard esistenti (in particolare DICOM e HL7) dovevano essere utilizzati dai diversi sistemi informativi per realizzare un'integrazione tra loro, partendo dall'analisi del reale flusso di lavoro clinico.

Nelle strutture sanitarie esistono numerosi sistemi informativi distinti, che gestiscono i dati anagrafici, clinici e diagnostici del paziente. Questi sistemi necessitano di condividere informazioni, tuttavia, pur utilizzando protocolli standard di comunicazione, spesso non sono in grado di scambiarsi efficientemente dati, in quanto gli standard stessi possono presentare conflitti interpretativi e una scelta di opzioni troppo ampia.

IHE ha stabilito un forum, con comitati volontari di operatori dell'ambito sanitario, esperti HIT e tutte le parti interessate in diverse cliniche e domini operativi, per raggiungere il consenso su soluzioni standard riguardanti gli aspetti critici dell'interoperabilità.

IHE fornisce un processo per gli sviluppatori per il collaudo delle loro imple-

mentazioni dei profili IHE, inclusi regolari eventi di test chiamati Connectathon. Dopo che un comitato ha determinato che un profilo ha superato un adeguato numero di test ed è stato applicato con successo in un ambiente sanitario reale, viene incorporato nell'appropriato Technical Framework IHE. Il Technical Framework è quindi una risorsa unica per lo sviluppo e l'utilizzo di sistemi HIT: un insieme di soluzioni testate e basate su standard per far fronte alle comuni esigenze di interoperabilità e supporto per l'utilizzo sicuro degli Electronic Healthcare Record (EHR).

Gli acquirenti possono specificare la richiesta di conformità con gli appropriati profili IHE come requisito per una proposta di acquisto. I rivenditori che hanno implementato con successo i profili IHE nei loro prodotti, possono pubblicare le dichiarazioni di conformità (chiamate IHE Integration Statements) nell'IHE Product Registry [1].

1.5 Standard e Linee guide

1.5.1 HL7

Per poter scambiare informazioni tra componenti diverse di un Sistema Informativo Sanitario (SIS), o tra applicativi diversi, o tra software diversi di cartella clinica elettronica, o tra strumentazione e calcolatori, e quindi per favorire la collaborazione tra sistemi, si deve utilizzare un linguaggio standard.

Fondata nel 1987, HL7 è un'organizzazione non-profit, dedicata a fornire un quadro globale e dei relativi standard per lo scambio, l'integrazione, la condivisione e il recupero delle informazioni sanitarie elettroniche per il supporto della pratica clinica e la gestione, erogazione e valutazione dei servizi sanitari.

Lo standard ISO/HL7 27931:2009 [2], in rapida diffusione anche in Italia, è concepito per lo scambio di informazioni cliniche tra i diversi possibili attori di un SIS che usano piattaforme hardware e software diverse. HL7 non è un linguaggio di programmazione, ma è un insieme di regole per la comunicazione da applicazione ad applicazione. Il numero 7 si riferisce infatti proprio al livello applicazione dello standard ISO per i protocolli di comunicazione.

I vari tipi di messaggi tra componenti di un SIS previsti in HL7 coprono le varie esigenze cliniche e sono:

- Admsission Dimission Transfer (ADT)
- Finanziari
- Report di dati
- Accesso a campi di un database
- Controllo
- Ordini
- Appuntamenti
- Referti
- Esami di laboratorio

Indipendentemente dal tipo di messaggio, esso è un testo ASCII strutturato nel modo seguente [fig. 1.4]:

- un messaggio è suddiviso in più segmenti
- un segmento è suddiviso in più campi
- un campo può essere suddiviso in più componenti

Ogni segmento contiene informazione correlata dal punto di vista logico, con una propria intestazione di tre lettere (ad esempio in messaggi di tipo ADT ci sono i segmenti con intestazione MSH, EVN, PID,...), ed è demarcato dagli altri segmenti dal separatore ritorno a capo.

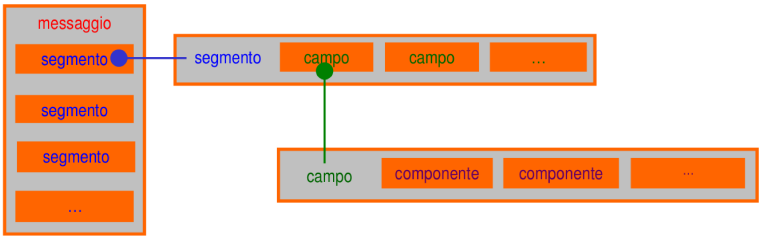

Figura 1.4: Struttura di un messaggio HL7

Ciascun segmento è articolato in più campi, demarcati da separatori pipe (), che contengono i dati veri e propri (nome e cognome del paziente, data di nascita, nome e cognome del medico di riferimento, ...). I campi possono prevedere delle sotto componenti tra loro divise da dei separatori (il simbolo ^).

1.5.2 DICOM

Le organizzazioni American College of Radiology (ACR) e National Electrical Manufacturers Association (NEMA) nel 1983 hanno formato un comitato per sviluppare uno standard per:

- promuovere la condivisione di immagini digitali, indipendentemente dai produttori dei dispositivi
- facilitare lo sviluppo ed espansione di sistemi PACS che potessero anche interfacciarsi con altri sistemi informativi sanitari
- permettere la creazione di basi di dati di informazioni diagnostiche che potessero essere interrogate da un ampia varietà di dispositivi distribuiti geograficamente.

Lo standard ISO 12052:2006 DICOM è stato realizzato dal comitato per facilitare l'interoperabilità tra dispositivi medicali di imaging, specificando:

- un insieme di protocolli di comunicazione di rete che i dispositivi devono implementare per essere conformi allo standard
- la sintassi e la semantica dei comandi e delle informazioni associate che possono essere scambiate con i suddetti protocolli di rete
- un insieme di servizi di memorizzazione, di formati di file e strutture di directory, che i dispositivi devono implementare per essere conformi allo standard, per facilitare l'accesso alle immagini e relative informazioni
- le informazioni devono essere fornite da un sistema conforme allo standard

Lo standard DICOM non specifica: i dettagli implementativi, altre caratteristiche che attese da sistemi che affermano l'aderenza allo standard, procedure di collaudo per valutare l'aderenza allo standard.

Lo standard risolve il problema di interscambio di informazioni digitali tra strumenti medicali per immagini e altri sistemi, ma non considera l'interoperabilità con altri tipi di dispositivi medicali.

1.5.3 Technical framework IHE

Il Technical Framework IHE definisce le specifiche implementazioni di standard ben stabiliti per ottenere obiettivi di integrazione che promuovano un'appropriata condivisione delle informazioni medicali per un supporto ottimale della salute del paziente. Viene aggiornato ogni anno, dopo un periodo di valutazione pubblica, e mantenuto regolarmente attraverso l'identificazione e correzione degli errori. L'attuale versione, rev. 10.0, specifica le transazioni IHE definite e implementate fino a ottobre 2010. L'ultima versione del documento è sempre reperibile sul sito web di IHE[3].

Il Technical Framework IHE identifica un sottoinsieme dei componenti funzionali per le aziende sanitarie, chiamati Attori IHE, e specifica le loro interazioni in termini di insiemi di transazioni coordinate standard. Attualmente sono presenti i seguenti Technical Framework:

- IHE Cardiology Technical Framework
- IHE Eye Care Technical Framework
- IHE IT Infrastructure Technical Framework
- IHE Laboratory Technical Framework
- IHE Patient Care Coordination Technical Framework
- IHE Radiology Technical Framework

Il Technical Framework IHE identifica gli attori IHE solamente dal punto di vista delle loro interazioni nell'azienda ospedaliera. Al momento definisce transazioni basate sugli standard HL7 e DICOM.

IHE è quindi un framework implementativo, non uno standard, non è corretto riferirsi a IHE come a uno standard. Tuttavia la conformità dichiarata da un prodotto deve fare diretto riferimento a uno standard specifico. Inoltre sviluppatori che hanno implementato uno o più profili IHE, devono utilizzare un IHE

Integration Statement per descrivere la conformità dei loro prodotti alle specifiche del Technical Framework IHE. Lo scopo dell'IHE Integration Statement è quello di comunicare agli utenti i corrispondenti profili IHE che il prodotto supporta.

Attori IHE

Gli Attori IHE e le transazioni descritte nel Technical Framework IHE sono astrazioni di un sistema informativo sanitario reale. Sebbene alcune delle transazioni vengano solitamente eseguite da categorie di prodotti specifici (per es. HIS, Electronic Patient Record (EPR), RIS, PACS, Clinical Information System, ecc.), il Technical Framework IHE evita intenzionalmente di associare funzioni o attori con tali categorie di prodotti.

Per ogni attore il Technical Framework IHE definisce solo quelle funzioni associate a sistemi informativi integrati. La definizione IHE di un attore non deve quindi essere intesa come una definizione completa del prodotto che possa implementarlo, né il framework stesso deve essere preso come una guida che descriva l'architettura del sistema informativo sanitario.

La ragione dietro la definizione degli attori e delle transazioni è di fornire delle basi per definire le interazioni tra componenti funzionali per i sistemi informativi sanitari.

Profili di integrazione IHE

Ogni profilo di integrazione [fig. 1.5] è la rappresentazione di una risorsa del mondo reale, supportata da un insieme di attori che interagiscono tramite transazioni.

Gli attori sono sistemi o componenti di un sistema informativo che producono, gestiscono o agiscono su categorie di informazioni richieste dalle attività operazionali dell'azienda. Le transazioni sono interazioni tra attori che trasferiscono le informazioni richieste attraverso messaggi basati su standard.

I profili di integrazione IHE offrono un linguaggio comune che professionisti nell'ambito sanitario e rivenditori possono utilizzare per comunicare i requisiti per l'integrazione dei prodotti. I profili di integrazione descrivono scenari reali o specifici insiemi di funzionalità dei sistemi integrati. Ciascuno di essi si applica

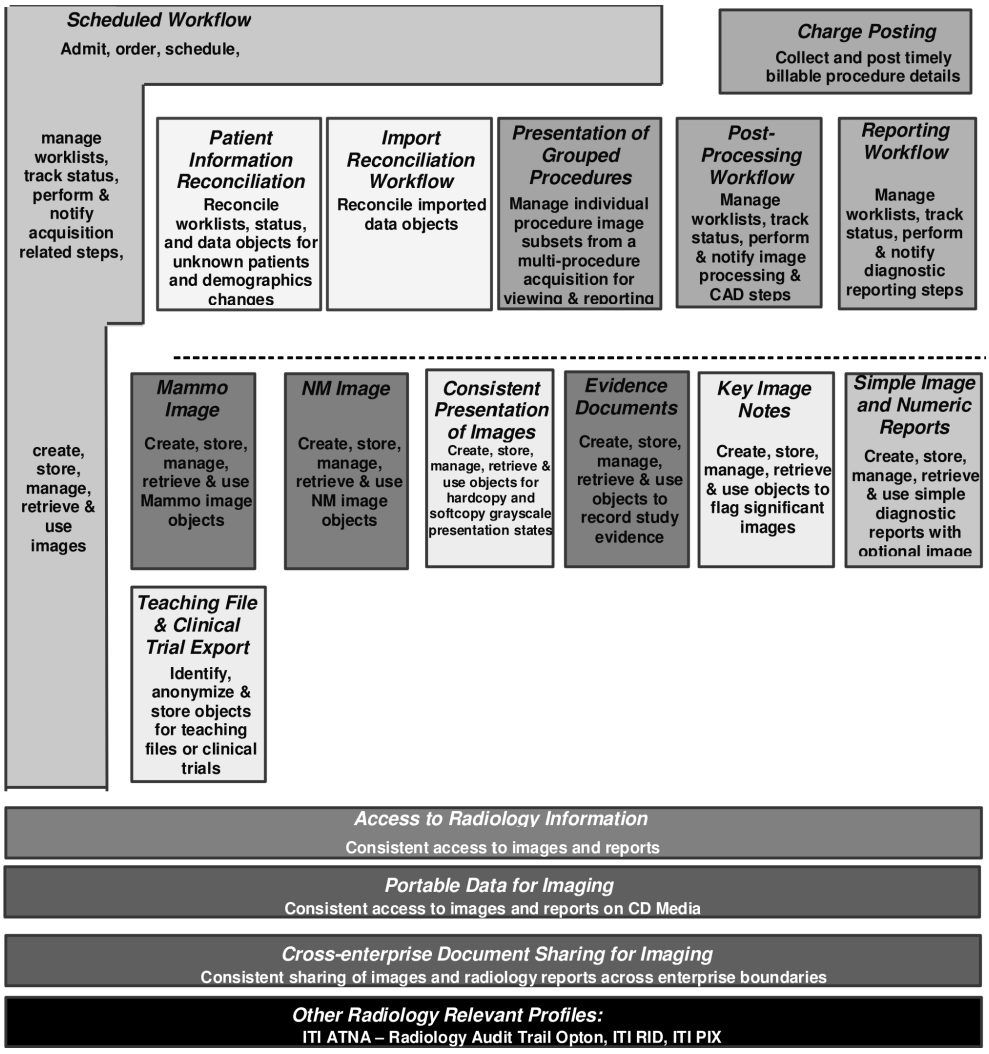

Figura 1.5: Profili di integrazione IHE

ad uno specifico insieme di attori, per ognuno dei quali vengono specificate le transazioni necessarie a supportare tali funzionalità.

I profili di integrazione forniscono un comodo mezzo per utenti e rivenditori di referenziare un sottoinsieme di funzionalità descritte nel Technical Framework IHE. Garantiscono a utenti e rivenditori un maggior dettaglio sulle implementazioni aderenti a IHE, senza dover riscrivere i dettagli relativi agli attori IHE e relative transazioni.

I profili si suddividono in tre classi:

- profili del contenuto, o Content Profiles
- profili del flusso di lavoro, o Workflow Profiles
- profili dell'infrastruttura, o Infrastructure Profiles

I Content Profile si riferiscono alla gestione di un particolare tipo di contenuto: come viene creato, immagazzinato, richiesto e recuperato.

I Workflow Profile affrontano la gestione dei processi dei flussi di lavoro, i quali tipicamente riguardano worklist e il reporting/monitoraggio dei progressi e dei completamente dei workitem. In questo contesto, uno o più oggetti contenuto vengono creati in accordo ai loro Content Profile.

Gli Infrastructure Profiles si occupano delle questioni generali del dipartimento, come per esempio l'audit trail di radiologia o l'accesso al sistema informativo radiologico.

In generale i profili di integrazione IHE non operano indipendentemente. Oggetti che servono come input di un profilo, possono a loro volta essere il risultato di un altro profilo. In alcuni casi questo tipo di dipendenze è strettamente necessario per il corretto funzionamento del profilo. In generale comunque, ogni profilo ha poco valore se ad esso non è associato un relativo profilo di contenuto.

1.6 Conclusioni

Al giorno d'oggi la scienza informatica è parte integrante dei processi sanitari, sia di tipo amministrativo che di tipo sanitario. le tecnologie informatiche sono ormai essenziali in campo clinico, per migliorare la diagnosi e rendere più efficaci le cure, grazie ad apparecchiature con una forte componente informatica Il

valore elevato degli investimenti in questa tecnologia lascia intendere un elevato grado di automazione dei processi di gestione e di erogazione dei servizi, ed è, quindi, un indicatore iniziale del grado di efficienza del sistema sanitario nel suo complesso. Le linee guida e gli standard presentati in questo capitolo rappresentato la risposta più efficace per la migliore interoperabilità possibile tra sistemi informativi, in particolare radiologici, all'avanguardia anche comparativamente tra le discipline cliniche.

Capitolo 2

I due sistemi MARiS e MN1

Introduzione

Tutte le organizzazioni di una complessità non banale hanno visto l'informatica affermarsi come strumento principe per una moderna gestione efficiente della propria macchina di funzionamento. Le aziende ospedaliere rientrano pienamente nella categoria, e la centralità dell'efficienza operativa è ancora maggiore considerando il ruolo vitale nella condizione quotidiana di molti cittadini. Anche per le discipline di Radiologia e Medicina Nucleare l'informatizzazione dell'amministrazione è stata un'evoluzione quanto meno necessaria, ed anzi la diagnostica per immagini patavina si è distinta per essere in prima linea per l'informatizzazione del proprio apparato. Queste due branche si caratterizzano per proprie peculiarità, che nel caso di Medicina Nucleare sono considerate di nicchia già nell'ambito della sanità.

Vedremo in questo capitolo la storia di MARiS e MN1 dalle origini, fondate sui Technical Framework IHE, fino alle emerse esigenze di integrazione in un unico sistema Analizzeremo le principali caratteristiche e l'adesione ai Technical Framework secondo i profili implementati.

2.1 MARiS e MN1

La sezione di Radiologia della facoltà di Medicina di Padova si è sempre distinta per essere all'avanguardia nell'adozione di nuovi strumenti informatici per supe-

rare l'obsoleta gestione amministrativa cartacea di un reparto ospedaliero. Per decenni la gestione è stata implementate usando supporti non elettronici: l'accettazione in forma cartacea, la refertazione su nastro e la conservazione delle immagini su pellicola film.

Radiologia è stata pioniera della realizzazione di una propria rete di comunicazione locale LAN per adattare il processo organizzativo interno al passaggio della refertazione dal sistema analogico a lastre a quello digitale filmless con files DICOM, una mole di dati molto elevata in un periodo in cui non esistevano strumenti telematici a nessun livello aziendale ospedaliero.

2.1.1 Evoluzione di MARiS

Il primo passo è stato l'adozione di un sistema RIS commerciale, ma alla fine del 1997 l'insoddisfazione per la scarsa rispondenza alle esigenze interne era ormai non più trascurabile. La scelta fu quindi quella di creare un'unità Informatica all'interno dell'allora Istituto di Radiologia sviluppando un software adeguato alle esigenze interne. Il progetto aveva l'obiettivo di realizzare un laboratorio di software radiologico disponibile a tutti senza alcuna limitazione.

In quel momento nacque il progetto ESO: le ridotte risorse economiche hanno favorito l'adozione fin dall'inizio di strumenti a basso costo. Il primo passo è stato la creazione di un'applicazione client-server per l'inserimento della refertazione di esami, usando tecnologie proprietarie. Nel frattempo anche l'azienda ospedaliera si dotò di tecnologie informatiche, con cui il sistema sviluppato internamento a Radiologia ha dovuto interagire.

L'arrivo di standard DICOM, HL7 e altri ha spinto ancora di più l'acceleratore verso la gestione totalmente paperless. Un sistema PACS si affiancò per archiviare e distribuire e immagini secondo standard DICOM, usando Central Test Node (CTN), un PACS realizzato dal RSNA. Inoltre, nel 2001 nasce IHE, all'inizio solo per Radiologia, con tutte le sue linee guida per favorire l'adozione di standard esistenti condivisi.

La prima versione di ESO fu sviluppata con architettura client/server in ambiente Microsoft (Windows 95 e Windows NT 4 Server) utilizzando Delphi 3.0 per lo strato lato client e Paradox come server per la base di dati. I risultati furono incoraggianti, ma dopo otto mesi il database raggiunse una dimensione

critica. Il passo seguente fu l'adozione del server database SQL Server, ma dopo pochi mesi l'incremento ulteriore della dimensione del database mise in crisi anche questo strumento. Quindi si decise il grande passo al sistema di gestione di basi di dati Oracle, che però si dimostrò scarsamente compatibile con il sistema operativo server usato Windows NT.

Subito dopo avere iniziato l'avviamento di ESO fu subito avvertita la necessità di distribuire i dati radiologici anche ai reparti richiedenti, sia per snellire le pratiche di invio sia per rispondere all'esigenza dell'Azienda ospedaliera di poter disporre dei referti radiologici nel minor tempo possibile. La necessità fu soddisfatta da un nuovo progetto, WebESO, un'applicazione web sviluppata con tecnologia Microsoft Active Server Pages su sistema Internet Information Server.

Perduranti problemi al sistema client-server, dovuti anche agli strumenti proprietari usati, hanno imposto un ripensamento dello stesso, e sia per una questione di costi che di opportunità di controllo si è giunti l'adozione di tecnologie open source. Il primo passo è stato la riconversione dell'applicazione web per la distribuzione della refertazione in tecnologia PHP, quindi l'adozione del sistema operativo server Linux, quindi l'adozione di software open client-server per la distribuzione e visualizzazione di immagini DICOM.

La sperimentazione di Linux ha indotto a tentare la strada del sistema di gestione di basi di dati open source, mettendo alla prova il sistema Interbase, che per le esigenze interne si è dimostrato equivalente a Oracle, quindi a MySQL, noto sistema open source molto diffuso.

La nuova componente Web si è dimostrata talmente migliorativa della classica client-server da spingere alla totale conversione verso il nuovo ambiente, creando il progetto *MARiS*, internamente noto col nome di *Crono*; come tutte le applicazioni web, il cambio di paradigma ha comportato nuove problematiche sia positive che negative. Da questo momento inizia la massiccia implementazione di attori e transazioni previste dalle linee guida IHE, come ADT, Order Filler e altre ancora.

Usato presso la Sezione di Radiologia del Dipartimento di Scienze Medico-Diagnostiche e Terapie Speciali dell'Università di Padova, il sistema MARiS si è evoluto da semplice RIS a gestore dei dati radiologici, ovvero superando le tradizionali separazioni fra RIS e PACS. Le funzionalità sono state sviluppate

seguendo le priorità interne, ad esempio non esiste una parte di gestione del ticket essendo presente una specifica procedura nell'Azienda Ospedaliera, mentre è stata introdotta sia la refertazione vocale sia la firma digitale.

Collaborazioni stabili sono state instaurate col Dipartimento dell'Informazione dell'Università di Padova, per realizzare nuove funzionalità generalmente legate ad aspetti di IHE, e con aziende terze per supportare l'adozione del sistema MARiS anche presso altre realtà ospedaliere e fornire supporto.

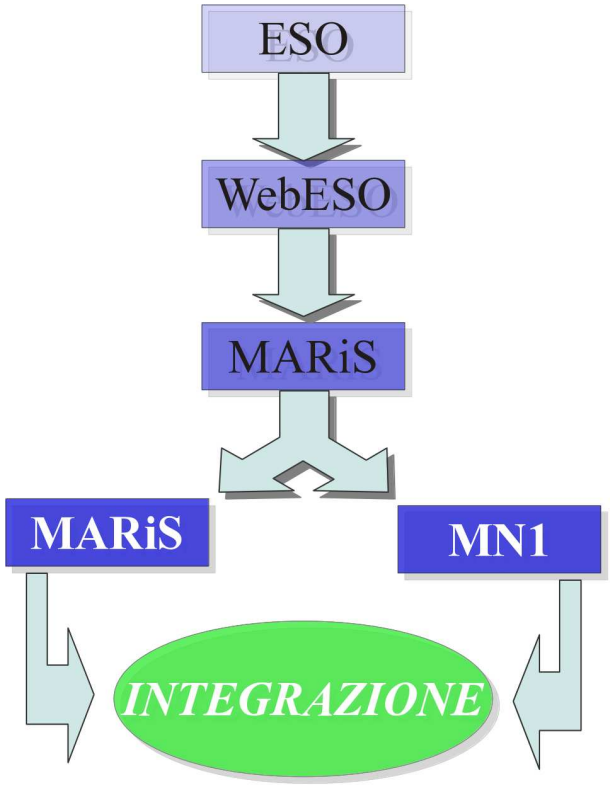

Figura 2.1: Diagramma dell'evoluzione dei RIS interni

2.1.2 La biforcazione

La situazione presso la sezione di Medicina Nucleare era molto diversa: essendo un settore di nicchia per le grandi aziende private non c'era interesse a sviluppare un sistema informatico specifico.

Le differenze rispetto a Radiologia sono poche ma incisive: la più importan-

te è la diversa gestione della calendarizzazione degli esami dovuta a una serie di questioni sollevate dai radiofarmaci, quali la preparazione anticipata dei radiofarmaci, i tempi e modi di somministrazione, i tempi di decadimento, e la disponibilità dei radiofarmaci.

L'esigenza di informatizzare la sezione si fece sempre più pressante, per questo per necessità immediate si realizzò un semplice sistema di refertazione archiviazione di immagini, chiamato *Penelopo* a causa della non eccessiva rapidità di sviluppo. Con la nascita di WebESO si decise di integrare Penelopo con quel sistema informativo, al contempo mutuando e adottando le linee guida **IHE!** (**IHE!**). L'evoluzione di WebESO in MARiS è proseguita di parallelamente anche nel sistema informativo realizzato per Medicina Nucleare.

Altre necessità simili ma non uguali a quelle di un RIS, questioni di tempo e opportunità, generarono la scissione di MARiS per Medicina Nucleare, MN1, il cui sviluppo è proseguito relativamente in parallelo rispetto a quello del progetto progenitore. Realizzato anche da appassionati programmatori ma di formazione medica, il percorso di sviluppo è stato sempre improntato alla massima semplicità.

2.1.3 Riunificazione dei progetti

Dopo alcuni anni in cui i due progetti MARiS e MN1 hanno percorso cammini pressoché paralleli, oggi si è giunti ad un punto in cui l'esistenza di due sistemi separati non è più fattibile ne opportuna.

Grazie ad una costante confronto, le differenze tra MARiS e MN1 sono state gestite per limitare il tasso di difformità tra i due, ma oramai il mantenimento di due sistemi simili comporta uno sforzo non congruo. Lo sviluppo parallelo ha prodotto codice sorgente ridondante e prono alla duplicazione, favorite anche dall'approccio di sviluppo elementare.

Da queste considerazioni è emersa la necessità di riportare le due applicazioni ad un'unica base comune. Il processo di riorganizzazione ha lo scopo di produrre un sistema più flessibile, con un minor numero di duplicazioni e ridondanze ma che permetta di mantenere i tratti caratterizzanti di ciascun sistema attuale.

2.2 Profili e attori di IHE implementati

Come visto nel capitolo 1, è molto importante riferirsi ed usare standard condivisi, e in particolare nell'informatica sanitaria dei sistemi informativi radiologici esistono linee guida uniformi di livello internazionale.

MARiS e MN1 non fanno eccezione, implementando un sottoinsieme dei profili di integrazione IHE ([fig. 1.5]).

In particolare, i principali profili implementati o direttamente coinvolti sono:

- Scheduled Workflow
- Patient Information Reconciliation
- Reporting Workflow
- NM Image
- Audit Trail and Node Authentication (ATNA)
- Audit Trail

2.2.1 Scheduled Workflow

Il profilo di integrazione Scheduled Workflow [fig. 2.2] determina la continuità e l'integrità delle informazioni di imaging dipartimentali di base. Esso specifica una serie di transazioni che consentono di mantenere la coerenza delle informazioni di paziente e ordine, oltre a fornire i passi per le procedure di schedulazione e acquisizione delle immagini. Inoltre questo profilo permette di determinare se immagini o altri oggetti di prova associati con una particolare procedura eseguita sono stato memorizzati e sono disponibili per consentire i passaggi successivi del flusso di lavoro, come la refertazione. Può anche fornire coordinamento per il completamento dell'elaborazione della procedura, oltre a notificare l'Order Placer di eventuali appuntamenti.

Gli attori di questo profilo implementati in MARiS/MN1 sono:

- ADT
- Department System Scheduler/Order Filler

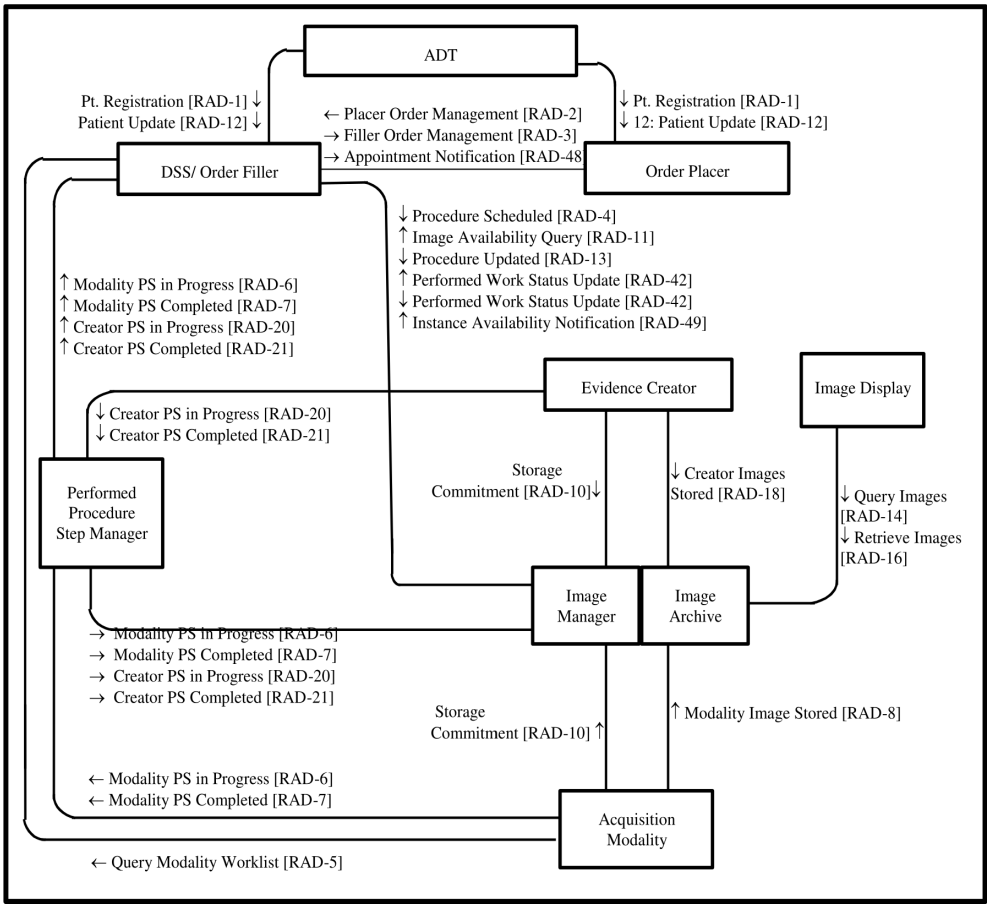

Figura 2.2: Profilo di integrazione Scheduled Workflow

- Order Placer
- Performed Procedure Step Manager
- Image Manager e Image Archive

L'attore ADT è responsabile per l'inserimento o l'aggiornamento delle anagrafiche dei pazienti. In particolare registra un nuovo paziente presso l'Order Placer e l'Order Filler.

L'attore Order Filler è un sistema dipartimentale (per esempio relativamente a tutta la sezione di Radiologia) che fornisce funzioni relative alla gestione degli ordini ricevuti da sistemi esterni o dipartimentali tramite opportune interfacce utente.

L'attore Order Placer è un sistema di livello aziendale ospedaliero che genera gli ordini per vari dipartimenti e distribuisce questi ordini al dipartimento corretto.

L'attore Performed Procedure Step Manager è un sistema che svolge il ruolo di intermediario tra gli attori Evidence Creator, Order Filler, Image Manager e Report Manager.

Gli attori Image Manager e Image Archive sono sistemi che forniscono funzionalità di memorizzazione e gestione relativamente a oggetti di prova. La memorizzazione è fornita a lungo termine e in modo sicuro, fornendo al sistema di schedulazione dipartimentale informazioni sugli oggetti memorizzati.

Questi attori non sono implementati direttamente dai progetti MARiS e MN1, ma sono realizzati utilizzando il software open source dcm4che[4] presso Radiologia e il software CTN presso Medicina Nucleare [1].

2.2.2 Patient Information Reconciliation

Il profilo di integrazione Patient Information Reconciliation (PIR)[fig. 2.3] estende i profili Scheduled Workflow e Reporting Workflow, offrendo le modalità per combinare le immagini, i report diagnostici e altri oggetti di prova acquisiti per un paziente non identificato o identificato in modo errato con il fascicolo del paziente.

[1] Medicina Nucleare è anch'essa in procinto di impiegare a dcm4che

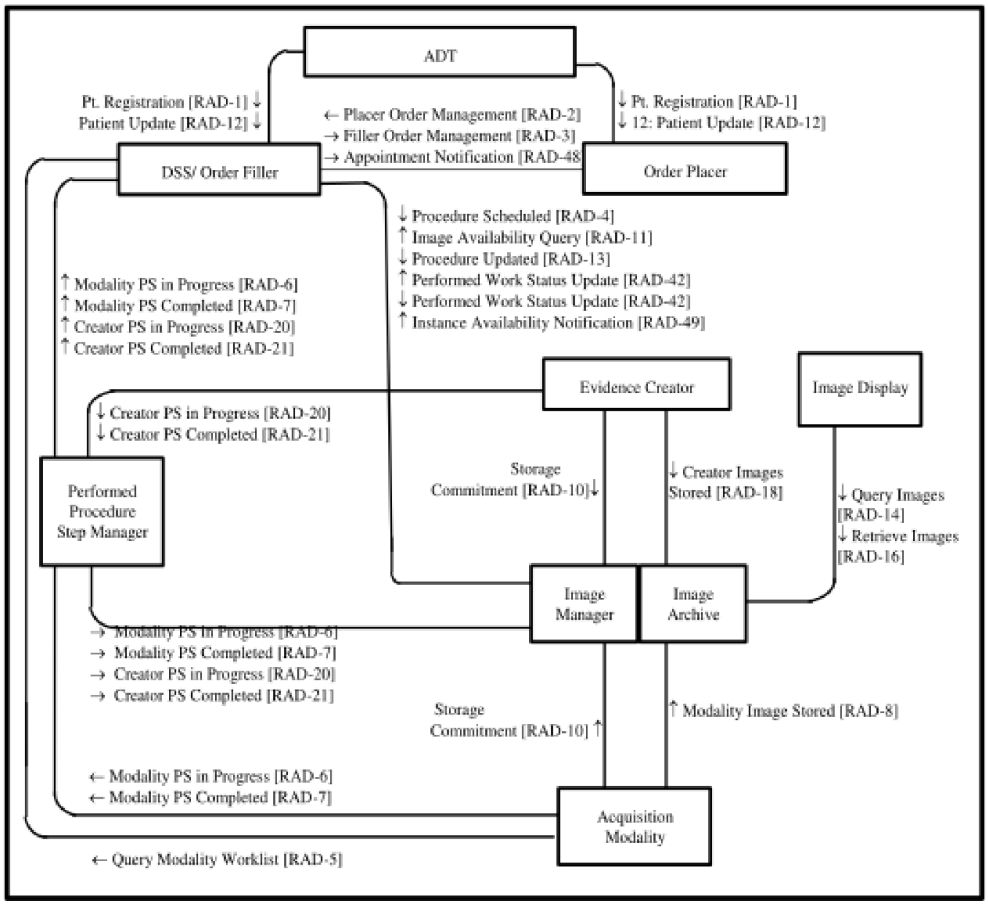

Figura 2.3: Profilo Patient Information Reconciliation

Ad esempio, nel caso di un paziente traumatizzato questo profilo consente la riconciliazione di immagini acquisite prima della determinazione dell'identità del paziente, anche senza previa registrazione o con una registrazione generica. Queste informazioni potranno quindi essere associate correttamente anche al momento dell'inserimento degli attori ADT, Order Placer e Order Filler. Inoltre questo profilo consente agli attori Image Manager e Report Manager di ricevere messaggi di aggiornamento dei dati dei pazienti.

Questo profilo aggiunge il solo attore Report Manager.

L'attore Report Manager provvede alla gestione e alla memorizzazione a breve termine di oggetti DICOM durante il processo di refertazione, distribuisce report testuali o strutturati ai centri di memorizzazione, gestisce la lista di lavoro e lo stato della refertazione stessa.

2.2.3 Reporting Workflow

Il profilo Reporting Workflow [fig. 2.4] è una continuazione del profilo Scheduled Workflow, e risolve la necessità di programmare e tenere traccia dello stato di vari incarichi di refertazione. Questi includono le fasi di interpretazione, dettatura, trascrizione, verifica, confronto, revisione e codifica. Il sistema che gestisce la procedura rende disponibile lo stato attuale anche ad altri sistemi interessati. L'output del profilo è costituito da informazioni codificate secondo standard DICOM. I dettagli per la creazione, memorizzazione, interrogazione, recupero e codifica sono descritti nel profilo Simple Image and Numeric Report (SINR).

Questo profilo aggiunge gli attori Report Creator e Report Reader.

L'attore Report Creator è un sistema che genera e trasmette bozze, e facoltativamente revisioni definitive, di referti diagnostici presentandoli come oggetti strutturati DICOM. Inoltre può recuperare la lista di lavoro per le fasi di refertazione dal Report Manager, fornendo notifiche del completamento di ciascuna fase, e permettendo il tracciamento dello stato di un referto.

L'attore Report Reader è parte di un sistema che può accedere ai referti tramite interrogazioni via rete o tramite lettura di supporti interscambiabili, permettendo all'utente di visualizzare referti presentati come oggetti strutturati DICOM.

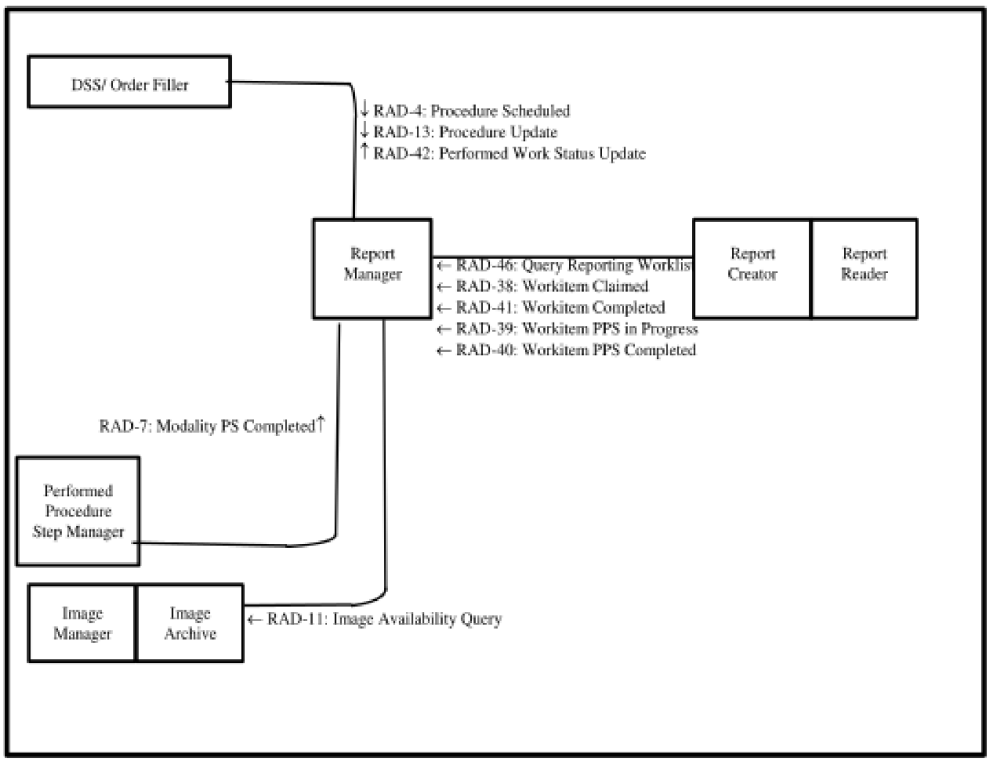

Figura 2.4: Profilo Reporting workflow

2.2.4 NM Image

Reporting Workflow

Il profilo NM Image [fig. 2.5] specifica come le immagini di Medicina Nucleare siano memorizzate dalle stazioni di Acquisizione e Creazione, e come i visualizzatori di queste immagini dovrebbero recuperarle e usarle. Definisce le capacità di base che i visualizzatori di immagini devono possedere, ma non specifica caratteristiche più avanzate. Questo profilo può essere valorizzato combinandolo con i profili del flusso di lavoro, come Scheduled Workflow, Post-Processing Workflow e Reporting Workflow, che risolvono il problema di come programmare e gestire gli stati delle fasi di creazione degli oggetti NM Image.

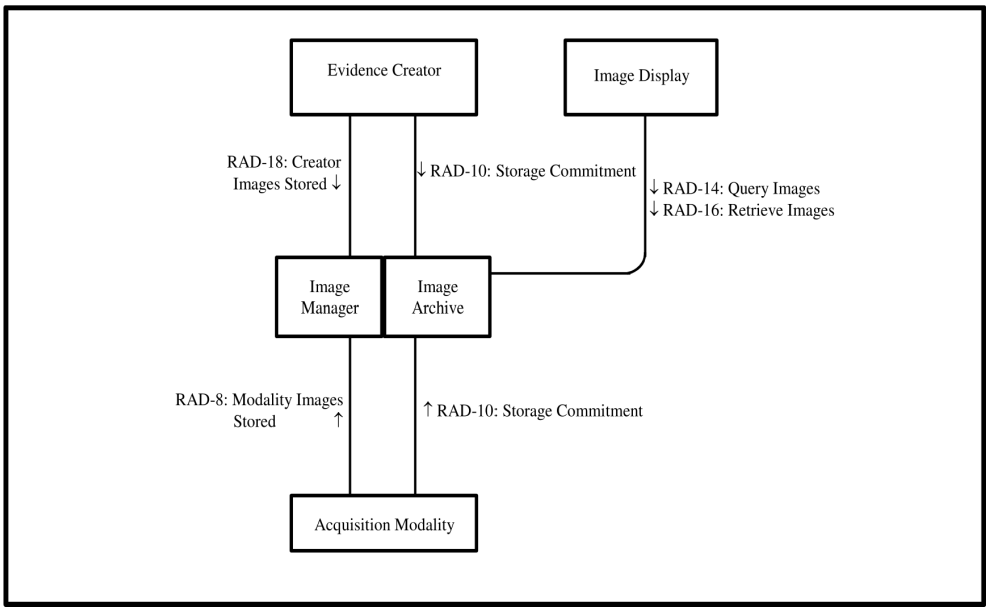

Figura 2.5: Diagramma profilo NM Image

Questo profilo non è implementato direttamente da MARiS e MN1, ma lo è da tutti i dispositivi di visualizzazione di immagini usati nei reparti e che collaborano con i sistemi informativi radiologici.

2.2.5 ITI-ATNA

Le transazioni IHE spesso contengono informazioni che devono essere protette in conformità con le leggi sulla privacy e le normative vigenti nelle diverse regioni. IHE include alcuni profili incentrati sulla sicurezza e la privacy, mentre la maggior parte non specifica alcun tipo di protezione, ma vanno raggruppate e integrate adeguatamente con i precedenti.

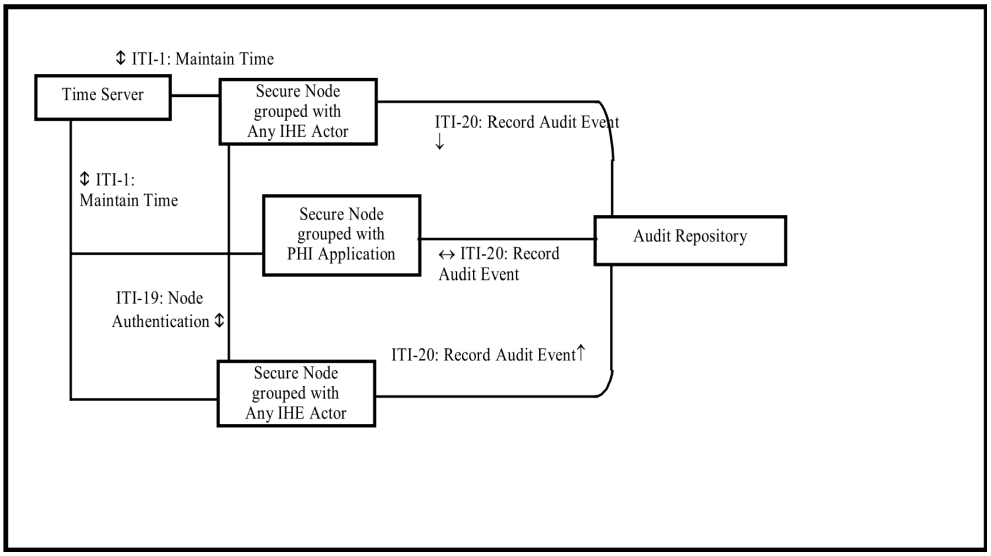

Figura 2.6: Diagramma profilo ATNA

Il profilo di integrazione ITI-ATNA, IT Infrastructure Audit Trail and Node Authentication , stabilisce le misure di sicurezza che, assieme alle procedure e politiche di sicurezza, garantiscono la confidenzialità dei dati sensibili, l'integrità delle informazioni e la responsabilità utente. L'ambiente così definito è detto dominio di sicurezza e può spaziare da un singolo dipartimento, fino all'intera azienda ospedaliera.

Il profilo Radiology Audit Trail è un opzione del profilo ITI-ATNA, ed è un sistema per garantire la responsabilità dell'utente. L'Audit Trail deve permettere a un ufficiale di sicurezza, autorizzato alla verifica delle attività, di valutare la conformità alle politiche del dominio di sicurezza, individuare istanze di comportamenti non conformi e facilitare l'individuazione di attività di creazione, accesso, modifica e cancellazione di informazioni sanitarie protette. Queste in-

formazioni possono essere richieste dagli utenti o scambiate tra sistemi. Ciò include le informazioni esportate e importate da e verso ogni nodo sicuro del dominio di sicurezza.

La responsabilità utente è ulteriormente assicurata da un deposito di Audit Record centralizzato (Centralized Audit Record Repository) altamente sicuro. Il trasferimento di un Audit Record , da un qualunque attore IHE all'Audit Record Repository, riduce le possibilità di manomissione e rende più semplice monitorare il dipartimento. Nodi disconnessi dalla rete dovrebbero salvare i propri Audit Record localmente e trasferirli all'Audit Record Repository non appena viene ripristinata la connessione al dominio di sicurezza.

L'Audit Trail and Node Authentication fornisce gli strumenti necessari a rendere l'azienda conforme alle norme di sicurezza e privacy (HIPAA, European, Japanese, ecc.), ma non la rende automaticamente conforme.

2.3 Funzionalità di MARiS e MN1

MARiS e MN1 realizzano una moltitudine di funzionalità coerentemente ai profili di integrazione, la maggior parte sono comuni a entrambe.

I moduli principali sono:

- gestione di anagrafiche dei pazienti
- gestione delle cartelle cliniche
- la refertazione
- l'agenda
- gestione della lista di lavoro
- modulo di amministrazione

Ogni modulo è accessibile agli utenti che abbiano le autorizzazioni necessarie.

In particolare, la componente di gestione delle Anagrafiche dei pazienti consente l'inserimento, la visualizzazione, la modifica delle stesse, oltre che l'unione di due pazienti in un'unica anagrafica, per realizzare il profilo PIR [2.2.2].

La gestione della Cartella Clinica consente l'inserimento di prestazioni da erogare, la visualizzazione dello storico, la modifica e la stampa delle prestazioni.

Il modulo di Refertazione consente l'inserimento di referti, la modifica, la stampa e la firma digitale del referto stesso; ogni fase è opportunamente tracciata e memorizzata, mantenendo lo storico del cammino di refertazione.

Il modulo di Agenda serve alla calendarizzazione degli esami; esso consente la visualizzazioni esami richiesti e non ancora schedulati, degli esami schedulati, la prenotazione di esami, la cancellazione di un esame richiesto, sia schedulato che non, la visualizzazione degli studi precedenti per il paziente dell'esame programmato. Per MN1 il modulo consente l'inserimento di molteplici *step* per una singola procedura, e di chiudere la fase di prenotazione.

Quindi, una volta programmato un esame la gestione dello stesso è demandata alla funzionalità di Worklist, o lista di lavoro. Esso consente la visualizzazione degli esami prenotati, assegnati e da effettuare in una specifica sala diagnostica, la modifica della procedura, la visualizzazione dettagli dell'ordine, la cancellazione dell'ordine, la visualizzazione della cartella del paziente, l'accettazione della procedura e l'eventuale segnalazione manuale dell'avvenuta esecuzione Per MN1 permette di gestire le iniezioni di radiofarmaci, dalla visualizzazione alla segnalazione avvenuta esecuzione.

Infine, il modulo di Worklist permette la stampa della lista di lavoro e delle etichette adesive da applicare ai documenti cartacei

Il modulo di amministrazione, accessibile solamente agli utenti di tipo amministratore, consente la gestione di tutte gli attori che devono essere configurati. Quindi la gestione degli utenti e delle loro preferenze, dei gruppi di utenti, la manutenzione dell'agenda per la creazione degli slot di prenotazione, delle postazioni, delle stampanti, delle sale diagnostiche. Inoltre, in MN1 è presente la gestione dei radiofarmaci, la configurazione dei parametri predefiniti di somministrazione di ciascun radiofarmaco e la visualizzazione degli ordini di acquisizione di nuovi radiofarmaci.

2.4 Caratteristiche di MARiS e MN1

I progetti alla base di MARiS e MN1 ormai hanno alle spalle qualche anno di vita. Le trasformazioni intercorse nel tempo, specialmente per ESO prima e MARiS poi sono foriere degli intenti che oggi ci si prefigge di perseguire continuando lo sviluppo di questi sistemi informativi.

2.4.1 La scelta Open Source

La ben nota filosofia open source, o a codice aperto, non è stata accolta fin dalla nascita del primo progetto di sistema informativo, tuttavia col tempo opportunità tecnologiche ed economiche hanno rafforzato l'adozione di strumenti liberi che in seguito hanno introdotto questa corrente di pensiero in MARiS. L'ambiente sanitario in cui si è sviluppato il processo ha certamente favorito l'apertura, per facilitare la diffusione degli standard e linee guida internazionali.

Infatti nell'ambiente radiologico sono disponibili molti altri software, alcuni usati da MARiS e MN1, che tra i loro punti di forza annoverano proprio l'orientamento aperto. Tra gli altri ricordiamo la collezione *dcm4che*[4] e la famiglia *mirth*[5], entrambi usati dai sistemi MARiS e MN1

dmc4che è un insieme di applicazioni open source basate su una robusta implementazione dello standard DICOM a disposizione di aziende private e ospedaliere per favorire l'interoperabilità di oggetti DICOM.

mirth invece è una famiglia di strumenti, supportata da una vasta comunità, per realizzare la migliore soluzione possibile per lo scambio di informazioni basato su standard HL7 In particolare, *mirth Connect* permette di sviluppare, collaudare e rilasciare interfacce di scambio dati.

2.4.2 Questioni legali

Alle considerazioni viste nel paragrafo precedente, se ne aggiungono altre più squisitamente normative. Le questioni fondamentali sono essenzialmente legate ad aspetti di privacy [6] e del codice dell'amminstrazione digitale [7].

L'ambiente sanitario è sottoposto a diverse problematiche relative alla privacy e alla sicurezza dei dati. In particolare, l'avvento di sistemi di archiviazione elettronici ha richiesto l'intervento di precise norme, in grado di disciplinare questi nuovi metodi di gestione dei dati sensibili. Con la legge 196/2003, meglio conosciuta come Codice in materia di protezione dei dati personali, o Testo unico sulla privacy, la precedente legge sulla tutela dei dati personali, 675/1996, è stata abrogata.

Codice in materia di protezione dei dati personali

Il codice si compone di tre parti e tre allegati, con disposizioni generali relative alle regole generali in materia di trattamenti dei dati personali, disposizioni particolari per specifici trattamenti o eccezioni alle regole generali, e disposizioni relative alla tutela degli interessati e al sistema sanzionatorio.

In particolare, gli articoli 34, comma 1, e 76, comma 1, prescrive le modalità di trattamento dei dati personali con strumenti elettronici; il consentito è consentito se sono adottate misure di:

- autenticazione informatica
- procedure di gestione delle credenziali di autenticazione
- utilizzazione di un sistema di autorizzazione
- aggiornamento periodico dell'individuazione dell'ambito del trattamento consentito ai singoli incaricati e addetti alla gestione o alla manutenzione degli strumenti elettronici
- protezione degli strumenti elettronici e dei dati rispetto a trattamenti illeciti di dati, ad accessi non consentiti e a determinati programmi informatici
- adozione di procedure per la custodia di copie di sicurezza, il ripristino della disponibilità dei dati e dei sistemi
- tenuta di un aggiornato documento programmatico sulla sicurezza
- adozione di tecniche di cifratura o di codici identificativi per determinati trattamenti di dati idonei a rivelare lo stato di salute o la vita sessuale effettuati da organismi sanitari

Inoltre, gli esercenti le professioni sanitarie e gli organismi sanitari pubblici, anche nell'ambito di un'attività di rilevante interesse pubblico, ai sensi dell'articolo 85, trattano i dati personali idonei a rivelare lo stato di salute:

- con il consenso dell'interessato e anche senza l'autorizzazione del Garante, se il trattamento riguarda dati e operazioni indispensabili per perseguire una finalità di tutela della salute o dell'incolumità fisica dell'interessato

- anche senza il consenso dell'interessato e previa autorizzazione del Garante, se la finalità di cui al punto precedente riguarda un terzo o la collettività.

In sintesi, il codice stabilisce che:

- le credenziali di autenticazione debbano basarsi su:

  password

  token di sicurezza, smartcard o analoghi

  caratteristiche biometriche

  una combinazione delle precedenti

- se è presente una password nel sistema di autenticazione, anche in combinazione con altri sistemi di autenticazione, questa deve essere composta da almeno otto caratteri, (se possibile) e deve essere modifica ogni tre mesi in caso di trattamento di dati sensibili;

- gli utenti inattivi da almeno sei mesi devono essere bloccati

- esistano degli adeguati sistemi di permessi, divisi per classi di utenti o per singolo utente

Tutti questi requisiti possono essere soddisfatti e direttamente controllati da un sistema di cui sia disponibile l'intero codice sorgente, che sia anche modificabile al bisogno.

Codice dell'amministrazione digitale

Il *Codice dell'amministrazione digitale* col Decreto Legislativo del 7 marzo 2005, n. 82, al Capo slowromancapvi@ stabilisce le *Modalità di sviluppo e acquisizione* di sistemi informatici nelle pubbliche amministrazioni. In particolare l'articolo 68, comma 1, elenca i parametri di comparazione tra soluzioni disponibili sul mercato, tra cui anche programmi informatici a sorgente aperto. Il comma 2 del medesimo articolo afferma che:

> *« Le pubbliche amministrazioni nella predisposizione o nell'acquisizione dei programmi informatici, adottano soluzioni informatiche*

> *che assicurino l'interoperabilità e la cooperazione applicativa, secondo quanto previsto dal decreto legislativo 28 febbraio 2005, n. 42 , e che consentano la rappresentazione dei dati e documenti in più formati, di cui almeno uno di tipo aperto, salvo che ricorrano peculiari ed eccezionali esigenze. »*

Di conseguenza soluzioni come MARiS e MN1 a codice aperto e realizzate secondo standard condivisi non possono che costituire un vantaggio per tutte le amministrazioni sanitarie di Radiologia e Medicina Nucleare al momento della valutazione di nuovi sistemi informatici.

2.4.3 Fattori sanitari-clinici

La peculiarità forse più rilevante dei sistemi MARiS e MN1 è quella di essere stati, e di essere ancora, sviluppati fianco a fianco con figure squisitamente sanitarie, oltre che amministrative.

Ciò permette l'immediata e diretta interazione con gli utilizzatori finali della soluzione, con un duplice vantaggio:

- ridurre notevolmente i tempi tra l'espressione di un nuovo requisito e la soddisfazione dello stesso, periodo noto anche come *lead time*
- evitare tutti i passaggi che, partendo dai reparti ospedalieri fino agli sviluppatori finali, comportano perdite di informazioni e improprie trasformazioni lessicali

Inoltre, la possibilità di personalizzare al bisogno e ad hoc la soluzione consente la realizzazione di specifiche funzionalità dedicate a scopi di ricerca medica; un esempio ne è il lavoro di tesi [8], volto a classificare automaticamente referti radiologici analizzando un set di referti inerenti alla patologia dello pneumotorace come supporto alla diagnosi.

2.4.4 Motivazioni commerciali

I dipartimenti di Radiologia e Medicina Nucleare in cui sono nati MARiS e MN1 afferiscono all'ateneo Patavino. Ciò ha permesso negli anni una fattiva

collaborazione con altre strutture ed enti universitari, tra cui il Dipartimento di Ingegneria dell'Informazione, sanitari e privati di Padova.

In particolare, da tempo è stata instaurata una positiva collaborazione con un'azienda padovana [9] che fornisce servizi di supporto per un sistema informativo radiologico basato su una scissione di MARiS. La realizzazione interna di un sistema open source e ha perciò consentito anche lo sviluppo di nuove competenze e la creazione di nuovi posti di lavoro sul territorio locale.

Parte II

Il nuovo framework

Capitolo 3

Analisi e progettazione

Introduzione

L'integrazione del codice sorgente di due applicazioni raramente è banale, anche quando esse siano frutto di progetti simili. Differenze funzionali, architetturali, anche stili di programmazione diversi impongono un'accurata analisi volta a determinare la base comune da cui partire per raggiungere lo scopo di integrazione. La complessità del software, uno sviluppo non sempre organico ed omogeneo e l'ambiente operativo ospedaliero non consentono di procedere ad una immediata integrazione dei due sistemi.

Considerando la necessità di procedere all'unificazione per passi successivi, mantenendo quindi la completa compatibilità tra il nuovo codice e il vecchio, e rendere il più semplice possibile allo sviluppatore finale [1] l'uso del nuovo sistema, dopo l'analisi della stato precedente è stata progettata una nuova architettura volta a ristrutturare il codice sorgente, per favorire l'integrazione di MN1 con MARiS senza soluzione di continuità.

3.1 L'analisi dei sistemi

Il ciclo di vita di un software consiste nel modo in cui una metodologia di sviluppo suddivide la realizzazione del prodotto software in sotto attività coordinate per

[1] Lo sviluppatore finale è colui che utilizzerà il nuovo framework per realizzare i requisiti funzionali

ottenere il prodotto stesso e la documentazione ad esso associata.

Le fasi principali principali del ciclo di vita sono le seguenti:

- Analisi
- Progettazione
- Implementazione
- Collaudo
- Rilascio

La fase di analisi ha lo scopo generale di distillare, specificare e documentare le funzioni, che devono essere offerti da un sistema software o programma, per risolvere un dato problema nel contesto operativo. Le informazioni raccolte nella fase di analisi rappresentano il punto di partenza per la progettazione di un prodotto software e per l'intero processo della sua realizzazione, validazione e manutenzione.

Questa fase è in generale particolarmente complessa, ed è suddivisa in sotto attività:

- la definizione del problema che il sistema da sviluppare deve risolvere
- l'analisi di fattibilità, per capire se gli obbiettivi siano ragionevoli e raggiungibili
- l'analisi dei costi e benefici, per valutare convenienza economica, costi previsti e benefici
- l'analisi del dominio in cui il sistema dovrà agire
- l'analisi dei requisiti, che specifica le funzioni richieste al sistema

3.1.1 Definizione del problema

Lo scopo principale della riprogettazione e problema da risolvere è l'integrazione dei due sistemi RIS esistenti. Una semplice e diretta aggregazione e integrazione dei file del codice sorgente non è possibile, non vi è identità né di funzionalità, né di file sorgenti, né di base di dati. La riprogettazione delle due applicazioni

dovrà smussarne le lacune, favorirne l'integrazione, aumentare la manutenibilità, collaudabilità ed estensibilità.

3.1.2 Fattibilità

I due sistemi MARiS e MN1 sono frutto di una biforcazione del medesimo progenitore. Negli lo sviluppo è proseguito in parallelo, ma l'accortezza di minimizzare il tasso di difformità consente di prevedere che il processo di riunificazione avrà un'alta probabilità di successo.

3.1.3 Costi e benefici

I benefici della riunificazione sono chiari ed evidenti, il preponderante è il minore sforzo necessario al mantenimento di due applicativi simili ma separati. Inoltre la riprogettazione è volta ad aumentare la manutenibilità, collaudabilità ed estensibilità, riducendo a sua volta lo sforzo di sviluppo.

Il costo principale è dovuto al tempo necessario per apprendere e acquisire padronanza con un nuovo modello di sviluppo.

3.1.4 Dominio applicativo

Questa analisi è propedeutica alla riprogettazione di sistemi esistenti, per quali il dominio applicativo è molto simile. Il dominio fondante non cambia, la riprogettazione deve ottenere un prodotto in grado di coniugare i due domini precedenti, che sappia mantenere le peculiarità di ciascuno dei due.

3.1.5 Requisiti

I principale requisiti che la riunificazione e riprogettazione dovranno soddisfare sono i seguenti:

- mantenimento delle funzionalità di entrambi i sistemi esistenti
- maggiore compatibilità possibile con il codice sorgente legacy
- mantenimento se non miglioramento delle prestazioni del sistema
- rafforzamento di paradigmi efficienti

- incremento della collaudabilità del codice sorgente
- curva di apprendimento relativamente ripida

3.2 Le basi di dati

Lo scopo di questa fase è studiare i due database esistenti di MARiS e MN1 per poter realizzare una loro integrazione affidabile: il primo passo è una macro analisi concettuale e logica, quindi l'individuazione di criticità di normalizzazione, di ridondanza e di convenzioni di sviluppo a livello di tabelle.

Poiché, per precisa scelta progettuale, nei due database sono assenti vincoli di chiave esterna, le relazioni tra le tabelle sono state dedotte dalle interrogazioni al database realizzate dall'applicazione.

Sono state individuate le seguenti criticità:

- il motore di storage MySQL usato è MyISAM[2]
- per precisa scelta progettuale non ci sono vincoli referenziali di chiave esterna
- esistono alcune tabelle con suffisso -old, oppure _old: queste tabelle sono obsolete
- alcune tabelle sono presenti in MARiS e non in MN1, e viceversa
- analogamente, molteplici attributi sono presenti in MARiS e non in MN1, e viceversa, o hanno tipi diversi

3.2.1 Motori di memorizzazione

Il motore di storage MyISAM è veloce e parco di risorse computazionali [10] ma, solo per citare le principali deficienze, non è completamente conforme alle specifiche ACID [3] , non supporta le transazioni, gestisce il lock a livello di tabella [4] e in caso di guasto del server può essere necessario ricostruire le tabelle.

[2] My Indexed Sequential Access Method
[3] Atomicity, Consistency, Isolation, e Durability.
[4] Il lock di tabella è molto veloce in lettura ma lento in scrittura.

Al contrario il motore InnoDB è totalmente aderente alle specifiche ACID, supporta le transazioni e il lock è a livello di record, ma non supporta indici FULL TEXT e impone limitazioni sull'uso di attributi auto incrementanti.

Il maggiore limite attuale del motore InnoDB è la mancanza della capacità di ricerche FULL TEXT: ad oggi esistono alternative open source su cui fare affidamento, tuttavia l'implementazione di questa caratteristica è prevista per una delle prossime release di MySQL. Ad ogni modo, al momento gli indici FULL TEXT non sono usati nelle applicazioni oggetto di questa analisi.

Dalla versione 5.5 di MySQL il motore di memorizzazione predefinito è diventato InnoDB. Nel sito web di MySQL [11] è possibile trovare una comparazione ufficiale tra i due motori: InnoDB si dimostra decisamente più scalabile, ma soprattutto più configurabile. È possibile modificare il livello di aderenza alle specifiche ACID per ottenere gradi diversi di prestazione in lettura, pur mantenendo un tasso di protezione dei dati superiore a MyISAM.

3.2.2 Vincoli relazionali

Non essendoci vincoli di chiave esterna, forzatamente la gestione di questo tipo di vincolo è demandata al livello applicazione, perciò tutte le relazioni tra le tabelle sono state dedotte dall'applicazione stessa o da discussione col gruppo di sviluppo.

3.2.3 Attributi

È stato osservato che i nomi di relazioni e attributi non rispettano una convenzione precisa: alcuni sono in lingua italiana, altri in inglese, le chiavi primarie non sono omogenee e altro ancora. Frequentemente i nomi non sono particolarmente descrittivi o inducono ambiguità, o addirittura sono fuorvianti.

Analisi di attributi

L'analisi dei tipi degli attributi ha rivelato una progettazione delle tabelle imperfetta. In particolare, sono state realizzate valutazioni sul tipo di dato per la chiave primaria della tabella con il maggiore numero di record, la tabella *AGENDA* del database di *MN1* La tabella AGENDA memorizza i turni di prenotazione per ciascun tipo di diagnostica del reparto, chiamate stanze; i turni, e quindi i

record, sono generati anticipatamente e periodicamente. Nel database di *MN1* usato in produzione dal maggio 2008 ad oggi sono stati inseriti 313000 record circa.

Fino ad oggi e nel caso peggiore, sono inseriti record per 17 ore di turni al giorno, per al più 6 stanze. In via cautelativa, è stato considerato il caso peggiore di 24 ore al giorno, per 8 stanze, ottenendo 281088 record all'anno. Il tipo di dato INT permette di memorizzare un valore massimo di 2147483647, che diventa 4294967295 se senza segno; al tasso di riempimento ipotizzato per esaurire lo spazio per il valore massimo con segno sarebbero necessari 7639 anni. [5]

Tuttavia il tipo di dato usato per la chiave primaria di questa tabella è BIGINT, che in MySQL occupa 8 byte, al contrario di INT che ne occupa 4 [10, pag. 716], con grande vantaggio di spazio e prestazioni degli indici.

3.2.4 Struttura delle tabelle di autenticazione

Inoltre, la gestione delle credenziali e dei permessi utente è relativamente inadeguata dal punto di vista della base di dati. La modalità attuale comporta l'aggiunta di un nuovo attributo alla tabella dei privilegi per ogni nuova azione di cui si devono concedere i privilegi.

La modifica della struttura di una tabella MySQL, per di più MyISAM, può generare molti problemi [10, pag. 2330] poiché l'alterazione avviene tramite creazione di una nuova tabella con la nuova struttura, copia dei dati dalla precedente, eliminazione della precedente e cambio del nome della nuova tabella.

3.3 Il codice sorgente

Il flusso di esecuzione dei due applicativi esaminati è semplice: il ciclo di generazione delle pagine HTML restituite ai client avviene sempre attraverso un'unica pagina principale.

Si tratta del modello con interfaccia web a pagina singola, o Single Page Interface [12]. Opportuni campi nascosti [6] determinano quali altre sotto pagine

[5] Per il reparto di Radiologia le stanze sono 26, ma la sostanza non cambia, in questo caso si impiegherebbero 2350 anni

[6] Elementi HTML input di tipo hidden

debbano essere incluse, che a loro volta prevedono che tutte le azioni di richiesta all'applicazione riferiscano di nuovo alla pagina principale.

Il codice sorgente consiste di circa 400 file tra file PHP, file Javascript [7] e file di stile CSS[8]. È strutturato in una gerarchia di directory [fig. 3.1] che definisce delle sotto sezioni, ciascuna rappresentante una macro funzionalità dell'applicazione; sfortunatamente esistono molteplici deroghe a questa logica, frutto di soluzioni rapide, efficaci ma non efficienti, e che spesso da temporanee sono diventati definitive. Inoltre non c'è alcuna separazione fisica e logica dei tre tradizionali livelli di un'applicazione[9] a tutto favore di codice ridondante, duplicato e difficile da testare.

L'architettura preesistente delle due applicazioni è solo parzialmente strutturata in moduli indipendenti: se sottosistemi come quello delle varie gestioni (per esempio la gestione dei pazienti o dell'agenda) sono relativamente indipendenti, non lo sono la gestione del login, della configurazione e della sessione.

3.4 Conclusioni dell'analisi

Il codice sorgente ha raggiunto una dimensione tale che, sviluppato secondo una logica procedurale a file singolo, esso non è più rapidamente manutenibile e verificabile.

Il codice procedurale rende semplice aggiungere nuove funzioni senza modificare le strutture dati esistenti, ma rende complicato aggiungere nuove strutture dati perché le funzioni devono cambiare[13, pag. 97].

Inoltre [13, pag. 172], un sistema che non sia collaudabile non è verificabile; un sistema è verificabile se si può determinare facilmente che agisce come previsto dal progetto, e un sistema che non sia verificabile non dovrebbe essere rilasciato in produzione La collaudabilità dei due sistemi esistenti è molto bassa, l'alto accoppiamento rende difficile introdurre test efficaci, come lo Unit Testing.

Anche il tasso di misurabilità dei sistemi è basso: è difficile misurare e ottenere risultati omogenei da un cammino di gestione della richiesta e generazione dell'output che non è uniforme rispetto a tutto lo spazio delle richieste.

[7] Linguaggio di scripting lato client

[8] Linguaggio per la formattazione di documenti

[9] Il livello dei dati, il livello della logica applicativa, e il livello della presentazione

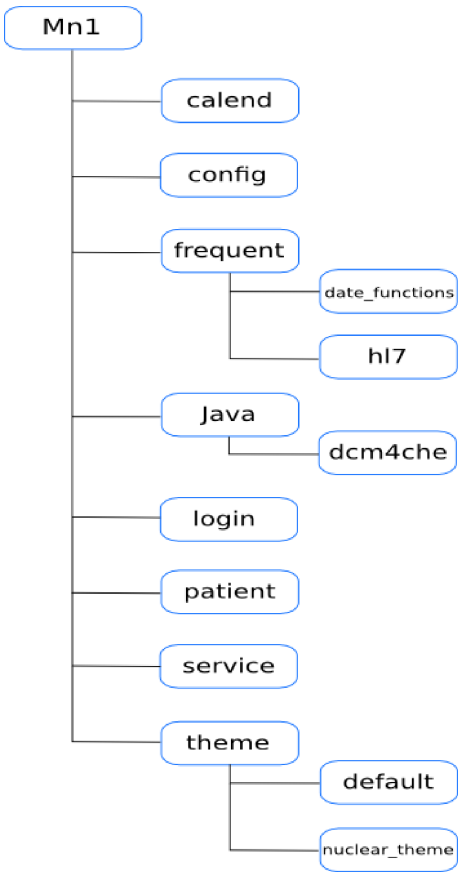

Figura 3.1: Struttura parziale di MARiS e MN1

Vista la notevole complessità del dominio applicativo, sono state considerate due possibilità per l'integrazione del codice sorgente:

- l'adozione di un framework esistente
- progettazione e sviluppo di un nuovo framework personalizzato.

3.5 Valutazione di framework esistenti

Un framework software è una struttura di supporto software che fornisce funzionalità generiche che possono essere modificate o estese selettivamente dal codice particolare dell'applicazione specifica.

Un framework è una collezione di librerie software che fornisce Application Programming Interface (API) ben definite, tuttavia è ben distinto da normali librerie software:

- fa uso del pattern *inversione del controllo*, in modo da controllare il flusso di esecuzione
- implementa un comportamento predefinito di una qualche utilità
- è estensibile in alcune sue parti per specifiche funzionalità
- una parte del codice non può essere modificata.

Un framework fine a se stesso non è assolutamente necessario, è solo uno degli strumenti che è disponibile per aiutare a sviluppare meglio e più velocemente. Esso fornisce la certezza che si sta sviluppando un'applicazione che è in pieno rispetto delle regole di business, che è strutturata, e che sia gestibile e aggiornabile. E più velocemente, perché permette agli sviluppatori di risparmiare tempo grazie al riutilizzo di moduli generici, al fine di concentrarsi su altre aree.

A lungo termine un framework assicura la longevità delle applicazioni. La struttura imposta da un framework consente a qualsiasi sviluppatore, anche se non ha mai lavorato prima sulla specifica applicazione, di iniziare a sviluppare in modo rapido e ordinato.

I criteri possibile per la valutazione e selezione di un framework applicativo open source già esistente, ma non solo, sono molteplici, tra i più importanti rileviamo:

- la dimensione della comunità open source
- il supporto, se fornito solo dalla comunità o anche da servizi terzi
- la documentazione disponibile
- la sostenibilità nel tempo del progetto
- le funzioni di sicurezza presenti nel framework
- la licenza

I framework brevemente analizzati sono i noti

- Zend Framework
- Simfony
- CakePHP

Tutti e tre si basano sul paradigma Model - View - Controller (MVC), sfruttano architetture Object Relational Mapping (ORM) per l'accesso al database e forniscono la completa gestione dell'autenticazione.

A fini di test e confronto, con i tre framework è stata parzialmente realizzata la funzionalità Agenda dei RIS in esame. L'analisi si trova nella sezione Modalità di collaudo.

3.5.1 Zend Framwork

Zend Framework [14], sviluppato da Zend Technologies Ltd., è forse il più avanzato, completo e allo stesso tempo complesso framework PHP. Il framework è potente ma relativamente difficile da imparare, e il suo uso richiederebbe un totale rifacimento dell'applicazione web; non solo per il pattern MVC ma anche per l'uso delle componenti di autenticazione/autorizzazione, di gestione della sessione, di presentazione e altro ancora.

L'elevata modularità di Zend consente di usare solo alcuni precisi moduli, ad esempio esiste un modulo per la generazione di documenti in formato PDF [10], o moduli per realizzare rapidamente la gestione CRUD [11] di semplici tabelle,

[10] Portable Document Format
[11] Create Read Update Delete

ma in questo ultimo caso lo sforzo di apprendimento sarebbe eccessivo rapportato al risultato da ottenere: Zend è certamente il framework con la curva di apprendimento meno ripida.

3.5.2 Symfony

Symfony [15] è un framework potente, probabilmente più di CakePHP, completo, usato da moltissimi siti e web application, con una buona documentazione e ben supportato da un'azienda privata. Basato sull'idea di non reinventare la ruota, usa altri strumenti open source per completare le funzionalità fornite, come moduli di Zend per l'autenticazione e Doctrine come ORM. Come complessità e funzionalità è paragonabile a Zend Framework, pur con alcune differenze:

- i template delle viste e gli helper HTML usano una sintassi non-PHP, per ottenere la massima velocità possibile
- i Model usano sia PHP che il linguaggio YAML[12] per la configurazione, e devono essere configurati esplicitamente.

3.5.3 CakePHP

CakePHP [16] è probabilmente il più facile da apprendere, potente ma non quanto i due precedenti framework. Sfrutta più di Zend e Symfony le convenzioni sui nomi, di conseguenza richiede meno configurazioni iniziali, permette di realizzare facilmente template delle viste e interrogazioni al database con operazioni *join* semplici immediatamente senza scrivere codice SQL, grazie al sistema ORM realizzato in proprio. Tuttavia da necessità più complesse emergono tutti i suoi limiti, in particolare dal lato del Model e dell'associazione al database, che framework come Symfony gestiscono in maniera più efficace sfruttando specifici strumenti realizzati da terze parti.

3.5.4 Conclusioni

I framework PHP analizzati si sono dimostrati tutti validi, e sono infatti molto apprezzati dalla comunità. Sono sufficientemente flessibili per realizzare qualunque tipo di applicazione desiderata, sempre nei limiti dell'architettura web.

[12] YAML Ain't Markup Language, Linguaggio di serializzazione dei dati

In riferimento ai requisiti che la riprogettazione deve soddisfare, osserviamo che sono sicuramente soddisfacibili il mantenimento delle funzionalità di entrambi i sistemi, il rafforzamento di paradigmi efficienti e l'incremento della collaudabilità del codice.

Altrettanto non si può dire per la compatibilità con il codice legacy. Tutti i framework considerati realizzano l'architettura MVC, impiegando l'Uniform Resource Locator (URL) della richiesta per effettuare l'indirizzamento delle richieste ai controller corretti e fornire loro i parametri.

Attualmente i due RIS offuscano e bloccano l'indirizzo della richiesta sfruttando il modello a singola pagina, e tra le richieste funzionali implicite vi è il mantenimento di tale modello. Di conseguenza l'uso di campi nascosti[13] diventa obbligatorio [14].

Considerando il contesto in cui dovrebbero essere calati i framework, cioè quello di due applicazioni web esistenti sviluppate secondo una propria logica di separazione dei file sorgente, ma non per livelli logicamente disgiunti, l'adattamento al design pattern MVC comporterebbe il pressoché totale rifacimento delle due applicazioni.

Inoltre, non è possibile ignorare il contesto dovuto agli sviluppatori finali, cioè coloro che nel tempo hanno contribuito a sviluppare con molta dedizione e lavoro i due sistemi, tra cui anche figure di formazione prettamente medica ma che per questo non dispongono della specifica formazione informatica utile a sfruttare appieno framework relativamente complessi.

3.6 Progettazione

La fase di progettazione è stata quella più consistente. Nell'ingegneria del software per progettazione architetturale si intende il processo che identifica i sottosistemi e ne determina la struttura di base di controllo e comunicazione. L'esplicitazione di questo passo vuole favorire la comunicazione tra le parti interessate e l'analisi preliminare del sistema. L'aspetto chiave su cui è stata concentrata la progettazione è quello della manutenibilità, perciò ,per quanto possibile stante la

[13] Elementi HTML input di tipo hidden

[14] Pur non essendo una reale misura di sicurezza: un utente relativamente abile è in grado di scoprire facilmente i valori contenuti nei campi nascosti

situazione esistente, sono stati progettati componenti piccoli, atomici, autonomi, modificabili velocemente e con effetti collaterali minimizzati.

Per il nuovo sistema è stato necessario scegliere l'approccio da usare per la scomposizione dei sottosistemi in moduli. La scomposizione del sistema in componenti può avvenire secondo due macro stili:

- pipelining orientato alle funzioni
- scomposizione orientata agli oggetti

Stile a pipelining

Lo stile a pipelining, o a flusso di dati, consiste in trasformazioni funzionali che elaborano dati di input e producono output, con il flusso di dati che viene trasformato mentre attraversa la sequenza. I vantaggi di questo stile sono la riusabilità delle trasformazioni, l'intuitività, la facilità di implementazione. Gli svantaggi sono un minore grado di astrazione rispetto al modello ad oggetti, la necessità di realizzare il trasferimento dati con un formato comune o con accordi tra trasformazioni limitrofe, e deve esserci un flusso di dati da elaborare. Il sistema attuale è realizzato secondo uno stile simile a quello a pipelining.

Modello orientato agli oggetti

Il modello architetturale orientato agli oggetti struttura il sistema in un insieme di oggetti accoppiati debolmente e con interfacce ben definite: gli oggetti richiedono servizi offerti da altri oggetti. La scomposizione orientata agli oggetti si occupa delle classi di oggetti, dei loro attributi e operazioni. I vantaggi ben noti: debole accoppiamento e quindi implementazione variabile senza influenzare gli altri oggetti, gli oggetti possono facilmente rappresentare entità reali ed favoriscono la riusabilità. Gli svantaggi derivano da possibili modifiche alle interfacce, e dalla difficoltà nel rappresentare entità molto complesse; queste problematiche possono essere superate con un'accorta progettazione iniziale e un corretto uso dei molti design pattern noti in letteratura [17].

Modello adottato

Per la progettazione del nuovo framework si è scelto lo stile orientato agli oggetti: questo stile permette di astrarre le funzionalità rispetto alla specifica implementazione, favorisce il riuso, la modularità, il disaccoppiamento e l'estensibilità. Inoltre questa tecnica è connessa a modalità di sviluppo di tipo "bottom-up", che individuano le entità di base del sistema e costruiscono applicazioni a partire da questi componenti.

I limiti dello stile a pipelining sono palesati dalle difficoltà di ulteriore sviluppo, collaudo e integrazione dei due sistemi esistenti. Inoltre lo sviluppo ad oggetti, se realizzato correttamente, non comporta maggiori complicazioni rispetto ad uno stile inizialmente più semplice ma prono alla degenerazione in codice scarsamente manutenibile.

3.7 Conclusioni

La fase di analisi è risultata più lunga e complessa del previsto. La documentazione insufficiente e la scarsa comprensibilità del codice sorgente, esemplificazione del noto anti-pattern *spaghetti code*, hanno dilatato i tempi rispetto alle previsioni.

L'incremento costante del grado di padronanza del codice legacy durante tutta la fase di analisi ha via via reso più chiara la necessità di adottare uno strumento volto ad armonizzare il modello di sviluppo e l'architettura di sistema. La complessità del dominio applicativo impone un elevato grado di disciplina del processo di sviluppo per garantire il corretto livello qualitativo dovuto ai reali utilizzatori finali, i pazienti.

Capitolo 4

Implementazione

Introduzione

Il sistema è stato sviluppato in tre fasi: analisi e studio della situazione attuale, integrazione dei database e riprogettazione e del software. Il database fondante l'integrazione è quello di MARiS, mentre per il codice sorgente si è scelto MN1, che contempla le funzionalità più complesse.

4.1 Database

4.1.1 Integrazione e ottimizzazione

Come database di riferimento per l'integrazione śtato scelto quello di MARiS: in MARiS si distingue la fase di accettazione di un paziente dalla fase di inserimento di un ordine di un esame, mentre in MN1 l'accettazione è realizzata dall'azienda ospedaliera e viene inoltrata la sola richiesta di inserimento di un ordine di esame. Tra i requisiti funzionali dell'integrazione dei due sistemi è stato individuato il superamento di questa differenza, integrando in MN1 la distinzione presente in MARiS.

Tra le molte caratteristiche, MySQL consente di modificare il motore di memorizzazione delle tabelle dei database anche per database esistenti; addirittura è possibile utilizzare motori diversi per tabelle diverse, eventualmente per convertire ad un motore più solido solo le tabelle più critiche dal punto di vista della concorrenza e dell'integrità dei dati, anche a costo di una piccola penalizzazione

per le prestazioni. Per questi motivi il motore di storage delle tabelle è stato convertito al motore InnoDB.

Si è deciso di uniformare il tutto alla lingua inglese, di riesaminare i nomi di tabelle e attributi dai punti di vista tecnico e medico, e di codificare le nuove chiavi primarie nel seguente modo:

- se la chiave primaria è composta da un singolo attributo, questo si chiamerà ID[nometabella]

- altrimenti gli attributi saranno nominati come ID[nometabella]_nomeAttributoX

Sono stati convertiti molti attributi con tipo BIGINT al tipo UNSIGNED INT. di intervallo 0-4294967293

Per precisa scelta degli sviluppatori finali, non sono state aggiunti vincoli di chiave esterna.

4.2 L'architettura software

L'integrazione di due applicazioni utilizzando la base di codice preesistente sarebbe estremamente dispendiosa e fonte certa di nuovi bug software; tuttavia mutare il processo di esecuzione richiederebbe il pressoché totale rifacimento di quasi tutto il codice sorgente.

Una parziale riprogettazione della pagina principale e del processo di inclusione delle sotto pagine consentirebbe di mantenere il flusso di esecuzione attuale rendendolo più mantenibile e più facile da testare.

Il nuovo framework è stato progettato per realizzare una variante del pattern MVC.

L'architettura software MVC [fig. 4.1] è un pattern dell'ingegneria del software usato per isolare il livello della logica di business dell'applicazione dal livello dell'interfaccia utente, permettendo sviluppo, collaudo e manutenzione indipendenti di ciascun livello.

Il pattern si è composto da tre macro parti:

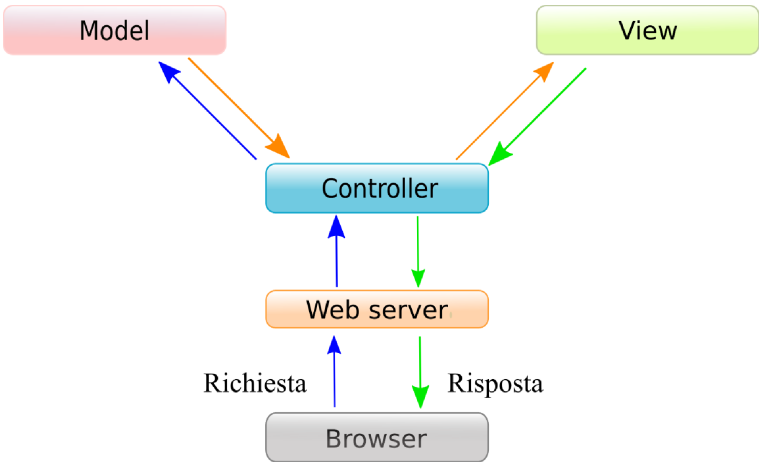

Figura 4.1: Pattern MVC classico

- il Model, che mantiene la logica dell'applicazione e gestisce i dati, fornendo e modificando lo stato applicativo
- il Controller, che riceve l'input dell'utente e notifica il Model delle azioni inviate
- la View, che si occupa di rendere disponibile all'utente il Model in un qualche tipo di interfaccia utente

In generale, il flusso di esecuzione si svolge nel seguente modo:

- l'utilizzatore interagisce con l'interfaccia utente
- il Controller coglie e gestisce l'interazione, traducendola in modo che sia comprensibile per il Model
- il Controller notifica il Model dell'azione eseguita dall'utente, possibilmente influenzando lo stato del Model
- una View, a volte su invito del Controller, altre su invito del Model, interroga il Model per generare l'opportuna interfaccia utente

I vantaggi dell'architettura MVC sono ben noti: ridurre l'accoppiamento tra i livelli dell'interfaccia utente e dei dati, aumentare la flessibilità e la manutenibilità del codice. Inoltre l'uso di sistemi di collaudo automatici trae notevole beneficio dall'implementazione di questa architettura.

Dall'analisi del codice sorgente originale si nota lo sforzo di riprodurre a grandi linee la separazione propria dell'architettura MVC, purtroppo senza duraturo successo: il principale limite dell'architettura precedente è quello di non imporre fortemente il corretto isolamento dei livelli dell'applicazione. Di conseguenza non è stato possibile calare il modello MVC senza modificarlo.

La nuova architettura implementata [fig. 4.2] realizza il modello MVC nel seguente modo:

- per ogni macro funzionalità si crea una classe [Funzione]Controller, che specializza la classe *Controller* di base

- una apposita classe funge da dispatcher, e utilizzando campi nascosti determina il corretto *Controller* concreto da istanziare
- viene creata un'istanza del controller concreto, che sa che campi nascosti aspettarsi e determina autonomamente l'azione da eseguire
- il controller esegue l'azione effettiva, e genera un insieme di dati da fornire alla vista
- l'applicazione richiama il componente di gestione della vista, passando i dati generati dal controller, e generando l'HTML in uscita
- la vista generata è inviata al client

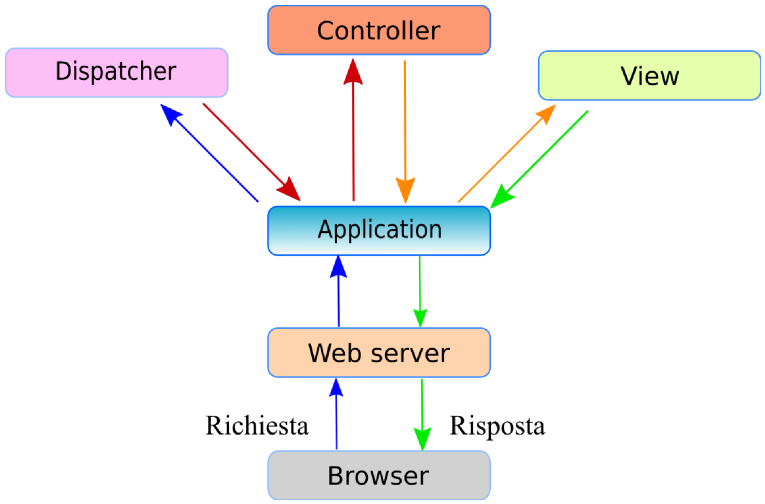

Figura 4.2: Pattern MVC implementato

Si noti che nella nuova architettura non è presente in alcun il concetto di Model. Il codice sorgente ereditato utilizza delle semplici interrogazioni al database, realizzate direttamente in linguaggio SQL, e mescolate al codice che realizza l'interfaccia utente. Poiché l'applicazione dovrà superare indenne un periodo di profonda trasformazione, l'introduzione del Model avrebbe compromesso la capacità del sistema di fare convivere le vecchie sezioni con le nuove.

Invece, quello del Controller è un concetto necessario, ma è stato implementato secondo una opportuna variante, dovendo supplire all'assenza del Model. In

generale la letteratura consiglia di collocare più logica di funzionamento possibile nei Model e di mantenere i Controller leggeri. In questo caso invece i Controller sono necessariamente ***"pesanti"***: oltre a determinare in autonomia l'azione da eseguire essi comprendono anche il codice necessario ad eseguirla.

Analogamente, la componente delegata alla costruzione della vista deve poter gestire la situazione di transizione.

4.2.1 Prototipo

È stato deciso di realizzare il prototipo per la funzionalità di Agenda; essendo quella più complessa e con maggiori differenze, se la soluzione si fosse dimostrata soddisfacente per questo specifico caso, allora quasi certamente lo sarebbe anche per tutta l'applicazione web.

Il nuovo framework non è stato sviluppato direttamente nel sistema MN1, è stato precedentemente prodotto un prototipo in separata sede, solo in seguito integrato nel codice sorgente di MN1.

La figura [4.3] presenta la gerarchia delle directory del framework.

4.3 Componenti

Per quanto possibile, si è cercato di seguire il principio della singola responsabilità [13, pag. 138]: oggi oggetto dovrebbe avere una singola responsabilità, definita come una singola ragione per cambiare stato: tale responsabilità deve essere totalmente racchiusa dalla classe e i servizi offerti dalla classe dovrebbero essere strettamente allineati con quella responsabilità. Focalizzando una classe su un singolo problema la si rende più robusta, e aumenta il grado di coesione [1] dei moduli del programma. Non sempre è stato possibile o preferibile seguire in modo stringente questo principio, sia per ragioni di compatibilità con il codice sorgente legacy, sia per ragioni di opportunità legate agli sviluppatori finali.

La figura [4.4] presenta i componenti fondanti la base del nuovo framework.

[1] La coesione è una misura di quanto è fortemente correlata ogni parte di funzionalità espressa dal codice sorgente di un modulo software.

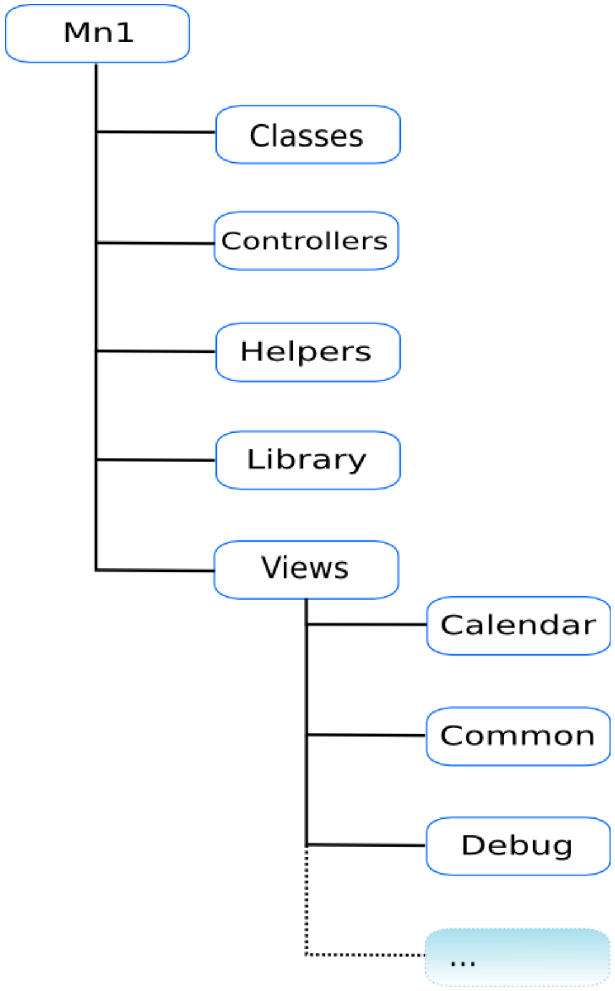

Figura 4.3: Gerarchia delle directory del framework

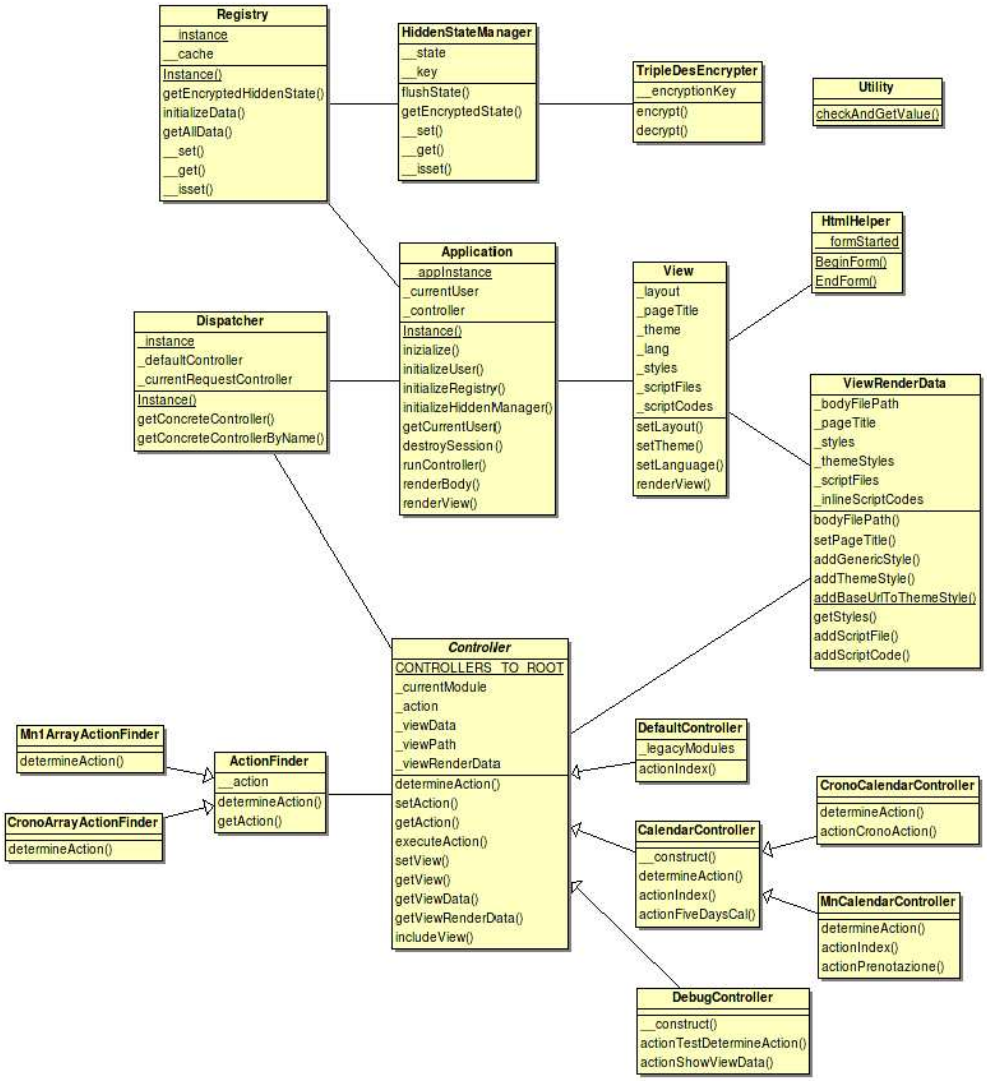

Figura 4.4: Diagramma delle classi del framework

4.3.1 Core e Application

La pietra angolare del framework è il componente chiamato Core; per come è progettato il linguaggio PHP, prima di poter usare una funzione o istanziare una classe si deve includere nello script corrente il file che contiene la definizione dell'entità richiesta. Alcune funzioni di base e quelle usate dalle componenti legacy devono sempre essere incluse: il file con i parametri di configurazione, il file con le funzioni di accesso al database e alcuni file di gestione delle date.

Tuttavia non è necessario includere esplicitamente i file delle definizioni delle classi PHP: l'apposita funzione *__autoload* viene automaticamente invocata dal runtime del linguaggio la prima volta che si tenti di usare una classe il cui file non sia già incluso. In Core questa funzione viene ridefinita: poiché si è scelto di usare la convenzione per cui una classe è contenuta nel file *NomeClasse.class.php*, l'inclusione del relativo file è semplice e rapida.

Il secondo componente chiave del framework è la classe *Application*, implementata con il pattern Singleton, che si occupa del ciclo di gestione della richiesta pervenuta al server web. La classe inizializza o distrugge la sessione PHP, avvia la gestione dell'utente che ha acceduto al sistema, inizializza il Registry e il gestore dello stato dei campi nascosti. Inoltre controlla l'esecuzione del Dispatcher e del Controller, oltre che avviare la fase di generazione della vista.

4.3.2 Registry

Uno dei primi problemi insorti con l'integrazione nell'effettivo codice sorgente di MN1 è stato quello di rendere disponibili i parametri della richiesta e tutte le variabili globali sia ai nuovi oggetti sviluppati, sia alle sezioni legacy.

La classe *Registry* implementa il design pattern chiamato proprio Registry: è un contenitore Singleton inizializzato dalla classe *Application* con i parametri della richiesta e con i valori delle variabili globali.
Ogni oggetto può cosi accedere a questi valori da ogni punto dell'applicazione; inoltre, nei punti di inclusione di script legacy tutti i valori del Registry sono copiati nello spazio di visibilità locale dello script, che può così espletare la propria funzione esattamente come sempre.

In questo particolare caso il pattern Registry non è forse la migliore soluzione possibile: introduce accoppiamento e riduce la collaudabilità del codice. Ad

esempio, il modello dell'Inversione di Controllo implementato tramite Dependency Injection[18], disaccoppia i singoli componenti di un sistema iniettando dall'esterno eventuali dipendenze. La ricerca della massima semplicità d'uso per gli sviluppatori finali ha reso preferibile una soluzione che fosse poi concretamente impiegata.

La gestione dei parametri di richiesta rende il Registry responsabile anche dello stato dei campi nascosti, estensivamente usati per passaggio di dati tra richieste successive. La classe *HiddenStateManager* critta e decritta lo stato e fornisce un'interfaccia per la sua modifica in modo automatico, centralizzato e trasparente alle altre componenti. La classe *TripleDesEncrypter*, usata da *HiddenStateManager*, si occupa dell'effettiva crittazione usando l'algoritmo Triple Data Encryption Algorithm (TDEA)[19].

4.3.3 Dispatcher

Fedele al principio della singola responsabilità, il *Dispatcher* ha un'unica funzione: creare il Controller concreto che deve gestire la richiesta corrente. Prima di poterlo creare, il *Dispatcher* deve determinare quale sia il corretto Controller da istanziare: per farlo si basa su un campo nascosto chiamato 'tab_button', che esiste sempre ad ogni richiesta, e in base al valore del campo verifica se esiste un Controller specifico per quel valore. Se non dovesse esistere il *Dispatcher* suppone di trovarsi nella condizione di f̈allback; cioè nella situazione di dover gestire la chiamata da uno script legacy. Solo in questo caso istanzierà un Controller apposito, chiamato *DefaultController*.

4.3.4 Controller

I controller costituiscono le componenti più importanti e complesse del nuovo ambiente; inoltre rappresentano il fulcro dell'integrazione dei due sistemi MARiS e MN1.
Essi costituiscono una gerarchia di classi, ossia un grafo aciclico interconnesso, la cui radice è la classe astratta *Controller*. Questa classe definisce l'interfaccia comune a tutti i controller concreti, a meno dei metodi che realizzano le azioni peculiari di ciascun controller concreto.

La classe astratta *Controller* non può essere istanziata direttamente, solo

le classi derivate possono esserlo. Ogni controller derivato realizza una macro funzionalità dell'applicazione: ad esempio il controller Calendar [fig. 4.5] per la funzionalità agenda, il controller Worklist per la funzionalità worklist e via seguendo.

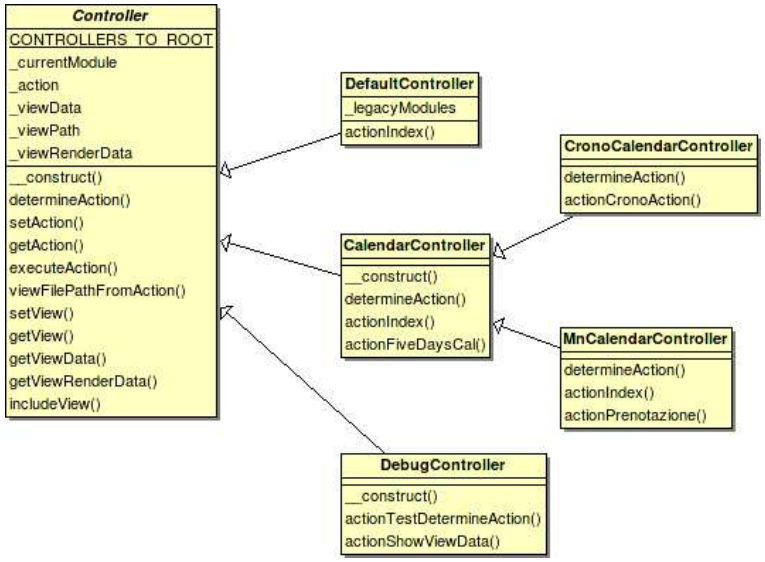

Figura 4.5: Struttura parziale gerarchia controllers

Pur se sarebbe stato possibile perpetrare la soluzione legacy, in cui ci sono singole pagine si occupano di una macro funzionalità, si è scelto di seguire una strategia che rendesse palese quello che già in molti casi le pagine legacy fanno, cioè usare una successione costrutti condizionali (if-then-else, piuttosto che switch) per discriminare flussi diversi di esecuzione.

La tecnica individuata consiste nell'esecuzione di un'azione diversa per ogni possibile flusso di esecuzione: ogni controller derivato determina in autonomia la specifica azione da eseguire, e quindi il relativo metodo di classe. Nel caso non fosse possibile determinare con precisione l'azione da eseguire, viene invocata l'azione predefinita *Index*.

Il metodo da invocare segue una precisa convenzione dei nomi:

action[NomeAzioneIndividuata]

Questo metodo deve essere definito nel controller direttamente o indirettamente derivato dalla classe *Controller*.

L'oggetto *Application* non invoca direttamente il metodo dell'azione essendo ignoto finché non viene stabilita l'azione, ma il metodo *executeAction* implementato dalla classe *Controller*, che a sua volta si occupa di invocare il metodo dell'azione:

```
$actionMethod = "action".ucfirst($this->_action);
if (method_exists($this, $actionMethod))
  $this->$actionMethod();
else
  throw new Exception("Method for action ".
    ucfirst($this->_action)." not found!");
```

$this riferisce alla classe concreta controller, e *$actionMethod* è la variabile che contiene il nome del metodo da invocare.

In questo modo ogni controller implementerà solo le proprie azioni, e diventa possibile implementare una gerarchia di controller. Ad esempio, nel prototipo sviluppato per la macro funzionalità dell'agenda sono stati creati i controller *Calendar* e i derivati *MN1Calendar* e *CronoCalendar* . *Calendar* riunisce le funzionalità comuni delle agende di MARiS e MN1, mentre *MN1Calendar* e *CronoCalendar* implementano alcune azioni specifiche di ognuno; inoltre attualmente MN1 e Crono calendar determinano l'azione da eseguire in modo diverso, e con tale gerarchia possono fare convivere questa differenza durante la fase di transizione. Se però dovesse essere eseguita la stessa azione, il metodo da invocare sarà definito in *Calendar*.

In futuro auspicabilmente le differenze saranno ridotte fino al punto di poter mantenere un unico controller *Calendar*, ma grazie al framework progettato il processo può avvenire gradualmente.

Uno speciale controller merita una menzione separata: il *DefaultController*. Esso svolge il ruolo di proxy tra le pagine legacy e la nuova modalità di gestione delle richieste, mantenendo l'associazione tra il campo nascosto usato per determinare quale script includere e la pagina stessa. Questo controller definisce il metodo *actionIndex* per l'unica azione che può essere invocata in questo caso, la quale valorizza i parametri necessari al motore di generazione della vista

per includere la pagina legacy, che eseguirà il proprio codice senza soluzione di continuità rispetto alla vecchia modalità di gestione del flusso di esecuzione.

4.3.5 ActionFinder

Le classi *ActionFinder* e derivate sono state introdotte per astrarre gli algoritmi usati dai controller per determinare l'esatta azione da eseguire. Nel codice legacy sono usati molti modi per determinare l'azione, sia tra moduli diversi sia tra MARiS e MN1. Tuttavia attualmente lo stesso algoritmo è semplicemente copiato da un modulo all'altro, con tutti gli inconvenienti del caso, e non esiste un'interfaccia comune per questi algoritmi.

La classe *ActionFinder*, implementata con il pattern Strategy [17], fornisce questa interfaccia tramite il metodo *determineAction*, che le classi figlie ridefiniscono caso per caso. Ad esempio, sempre per la funzionalità di agenda, sono state create le classi *Mn1ArrayActionFinder* e *CronoArrayActionFinder*, rispettivamente per MN1 e per MARiS, determinando una gerarchia [fig. 4.6]simile a quella vista per i controller.

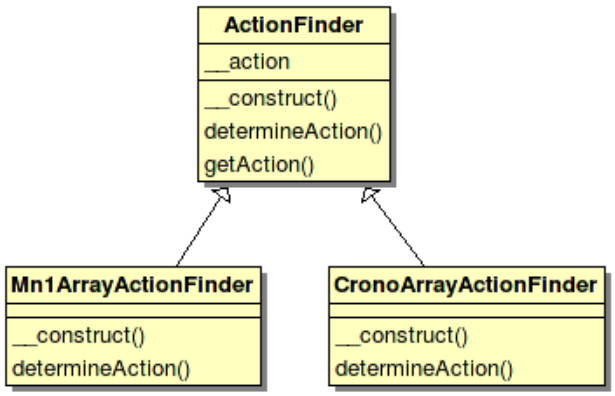

Figura 4.6: Esempio gerarchia strategie delle azioni

Col procedere dell'integrazione gli algoritmi potranno essere spostati verso la radice della gerarchia senza necessità di particolari modifiche agli oggetti utilizzatori.

4.3.6 ViewRenderData

La classe *ViewRenderData* è usata dai controller come contenitore di tutti quei parametri propri del livello di presentazione specifici del controller stesso e che sono usati dal motore di generazione della vista.

Essa gestisce il nome del file che costituisce il corpo della vista, il titolo della pagina generata, i file di stile CSS e di scripting lato client che devono essere inclusi nella pagina HTML finale.

4.3.7 View

La classe *View* gestisce la fase di generazione dell'output a partire da un modello comune a tutte le pagine, permettendo di modificare durante l'esecuzione il layout da usare, il tema dello stile e la lingua delle etichette. In questa fase è inclusa la particolare vista, richiesta dal controller, a cui sono forniti i dati imputati dal controller stesso durante l'esecuzione dell'azione.

Le directory delle viste sequono delle precise convenzioni :

- per ogni controller esiste una directory contenuta in View con lo stesso nome del controller
- in ciascuna directory dei Controller si trovano i file PHP delle viste, in generale uno per ogni azione definita nel controller e con lo stesso nome dell'azione
- i layout sono posti e cercati dal framework nella cartella Layout
- file comuni o usati più volte sono posti nella directory Common

Ad esempio per l'azione *Index* del controller *Calendar* si trova il file *index.php* nella cartella Calendar, per l'azione *Prenotazione* si trova il file *prenotazione.php*.

Un aiuto allo sviluppo della vista giunge dai cosidetti Helper Html, funzioni PHP che permettono di produrre codice quanto più uniforme possibile. L'unico helper veramente comune e necessario a MARiS e MN1 è quello che genera un form HTML, in quanto ci sono campi nascosti che devono esserci per il corretto funzionamento del framework ma che fino ad ora sono stati lasciati alla libera implementazione degli sviluppatori finali.

4.3.8 Altro

La riprogettazione del sistema non si è limitata ad aggiungere nuove entità. Sia per fini di funzionamento del nuovo framework, sia per razionalizzazione e generali miglioramenti di manutenibilità, sia la pagina principale che costituisce il cuore della Single Page Interface, sia il layout principale sono stati pesantemente rivisti.

4.4 Flusso di esecuzione

Il flusso di esecuzione della gestione di una richiesta che perviene all'applicazione web è stato progettato in modo da essere: trasparente al codice legacy, rapido e semplice da usare per gli sviluppatori finali. Salvo il caso della procedura di autenticazione, la sequenza dipende esclusivamente da parametri della specifica richiesta e da poche variabili persistenti nella sessione dell'utente autenticato.

Nel flusso di esecuzione [fig. 4.7] osserviamo che il primo passo consiste nell'inizializzazione dell'oggetto *Application*, creando il *Registry* e l'*HiddenStateManager* e caricando dalla sessione i parametri dell'utente autenticato.

Quindi *Application* trasferisce il controllo al Dispatcher, che determina quale sia il controller concreto da istanziare e lo istanzia. La gestione ritorna ad *Application* che provvede ad invocare sempre lo stesso metodo *executeAction* della classe *Controller*, che a sua volta si occupa di invocare il metodo relativo all'azione corrente stabilita in precedenza nel metodo *determineAction* ridefinito dal controller concreto.

Il controller in esecuzione ha anche la possibilità di trasferire l'esecuzione ad un altro controller noto; per esempio questa possibilità viene sfruttata dal controller di autenticazione che, in caso di autenticazione completata con successo, trasferisce l'elaborazione al controller relativo al modulo predefinito per l'utente corrente.

Terminata l'esecuzione della logica di funzionamento insita nei controller, la gestione ritorna all'oggetto *Application* che richiede al controller i dati da fornire alla vista, quindi crea e trasferisce all'oggetto *View* la responsabilità di generazione dell'output, fornendogli i dati ottenuti dal controller. Nel caso di

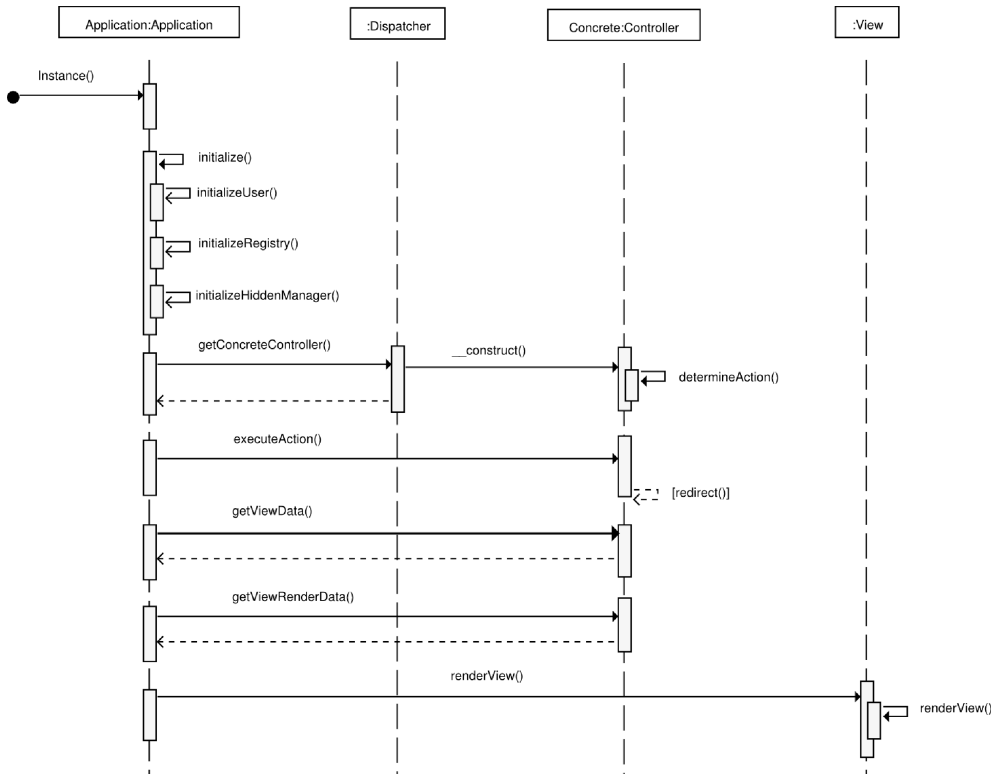

Figura 4.7: Flusso di eseuzione del framework

redirect tra controller, ai fini di generazione della vista viene considerato l'ultimo controller istanziato.

4.5 Conclusioni

Il framework progettato e sviluppato favorisce l'integrazione dei due sistemi informativi in vari modi.

È stato pensato per adattarsi alla situazione attuale: la compatibilità con il codice legacy permette di integrare un modulo funzionale per volta mantenendo inalterati gli altri.

La nuova struttura a livelli separati consente di focalizzare l'integrazione solo in alcune parti, e consente di suddividere più efficacemente le attività tra logica di applicazione e presentazione.

La modularizzazione disaccoppia i livelli facilitando i collaudi: sarà possibile variare e collaudare la logica nei controller e le parti di presentazione in modo totalmente indipendente; l'associazione tra un controller e una vista dipenderà solo dal contratto stabilito tra i due, cioè dai dati che il controller dovrà fornire alla vista sotto forma di variabili.

La gerarchia di controller permette di riunire funzionalità comuni tra i due sistemi mantenendo la possibilità di specializzare quelle peculiari. Per esempio nell'implementazione realizzata il controller *Calendar* riunisce le azioni comuni a MARiS e a MN1, mentre i due controller derivati *CronoCalendar* e *MN1Calendar* fornisce azioni specializzanti.

Capitolo 5

Analisi del nuovo sistema

Introduzione

La misura è essenziale in tutte le scienze, e lo è anche nella disciplina dell'ingegneria del software. Ricercatori e professionisti del settore si sforzano incessantemente di portare la scienza della misura nello sviluppo software. La qualità di un software può essere gestita solamente se può essere misurata. Una metrica software è una misura di una qualche proprietà di una parte di programma o delle sue specifiche. Lo scopo finale è ottenere misure quantitative, oggettive e riproducibili, con notevoli ricadute nella pratica: pianificazione dei piani di lavoro, dei bilanci, stime dei costi, assicurazione della qualità, debug del software, ottimizzazione delle prestazioni. Se non fosse possibile misurare non sarebbe neanche possibile valutare quali miglioramenti e quali involuzioni il nuovo sistema ha introdotto. In questo capitolo vedremo quali metodiche e quali strumenti sono stati usati per misurare le prestazioni dei sistemi originali, del nuovo framework e dei framework PHP più noti. Le misure ottenute sono state usate per comparare i vari sistemi.

5.1 Materiali e metodi

5.1.1 Piattaforma

Tutti gli strumenti usati si caratterizzano per essere open source, o comunque disponibili liberamente per chiunque, seguendo la filosofia alla base del progetto

MARiS.

Le fasi di progettazione, sviluppo e collaudo sono state eseguite su una macchina basata sulla piattaforma Core 2 Duo dotata di 3GB di RAM installata. L'ambiente di progettazione e sviluppo si compone come segue:

- sistema operativo Gnu
  Linux Fedora, versioni 14 e 16
- kernel Linux versioni 2.6.35 e 3.2.1, rispettivamente per Fedora 14 e 16
- server web Apache, versione 2.2.21
- server di database MySQL, versione 5.5.19
- console di gestione di MySQL phpmyadmin, versione 3.4.9
- ambiente PHP, versione di runtime 5.3.9
- ambiente di sviluppo integrato Netbeans, versione 6.9 e 7.1
- ambiente di sviluppo MySQL Workbench, versione 5.2.37

I framework Zend Framework, Symfony e CakePHP sono stati collaudati usando le versioni rispettivamente 1.11.11, 2.0.9 e 2.0.5.

5.1.2 Design pattern

Nell'ingegneria del software, un pattern è una soluzione generale riusabile per un problema ricorrente nel contesto della progettazione software, e trasformabile direttamente in codice sorgente[17]. Il pattern fornisce una descrizione o un modello sul modo di risolvere un problema che può essere incontrato in molte situazioni differenti. Nella progettazione orientata agli oggetti, i pattern modellano relazioni e interazioni tra classi e oggetti ma senza specificare le particolari implementazioni coinvolte. L'uso di Pattern noti consente a sviluppatori diversi di capire immediatamente come è stato risolto il particolare problema implementativo.

I principali tipi di design pattern sono i seguenti:

- creazionali: nascondono i costruttori delle classi fornendo un'unica interfaccia che nasconde l'implementazione delle classi

- strutturali: consentono di riusare classi esistenti tramite un'interfaccia più adatta alle esigenze degli utilizzatori
- comportamentali: forniscono soluzioni a modalità comuni di interazione tra oggetti
- architetturali: esprimono schemi di base per impostare l'organizzazione strutturale di un sistema software

Nel framework implementato sono stati utilizzati molteplici pattern:

- MVC (semplificato): è un pattern architetturale, introdotto nel capitolo precedente
- Singleton: è un pattern creazionale molto semplice e comune usato per assicurare che di una classe esista una sola istanza fornendo un punto di accesso globale alla stessa
- Registry [20, pag. 480]: è un oggetto ben noto che tutti gli altri possono usare per ricavare oggetti e servizi comuni.
- Factory method: ha il compito di istanziare altri oggetti, incapsulando le regole che determinano quali oggetti siano istanziati, disaccoppiando l'uso di un oggetto dalla sua creazione.
- Template Method: è un pattern comportamentale che in una gerarchia di classi consente alle classi derivate di ridefinire alcuni passi di un algoritmo mantenendo inalterati gli altri.
- Strategy: altro pattern comportamentale, è la perfetta incarnazione dell'approccio di progettazione orientata al cambiamento, poiché è un modo per definire una famiglia di algoritmi che realizzano la stessa funzione con implementazioni diverse ma intercambiabili senza che gli utilizzatori si preoccupino di quale sia la reale implementazione invocata.

I pattern sono stati usati in tutto il framework:

- MVC è la base di tutto il sistema
- Singleton è stato usato per le classi *Application*, *Registry* e *Dispatcher*

- Registry è stato usato per la classe *Registry*
- Factory Method è stato usato nella classe *Dispatcher*
- Template Method è stato usato dalle classi derivate da *Controller*, per ridefinire il metodo da invocare per determinare l'azione
- Strategy è usato dalle classi *ActionStrategy*, che implementato l'algoritmo di determinazione dell'azione

5.2 Collaudo

Il collaudo del software è un'analisi condotta per fornire informazioni circa la qualità del prodotto o servizio collaudato, secondo una visione oggettiva e indipendente del software. È il processo di validazione e verifica del fatto che un software:

- soddisfa i requisiti che ne ha guidato la progettazione e lo sviluppo
- lavora come atteso

Un difetto software è un errore, imperfezione o un guasto che impedisce al programma di comportarsi come previsto; la maggior parte dei difetti sono dovuti a errori degli sviluppatori.

Il collaudo del software può essere introdotto in qualunque fase del processo di sviluppo, dipendentemente dal metodo impiegato. Tecniche di collaudo del codice come lo Unit Testing permettono di collaudare il codice durante la fase di sviluppo, tuttavia la maggior parte avviene dopo che i requisiti sono stati definiti e la fase di sviluppo è stata completata.

Il termine collaudo del software è spesso usato in associazione con verifica e validazione: il programma realizza le specifiche? il sistema è proprio quello che il cliente vuole? Spesso i due lemmi sono usati intercambiabilmente nell'industria, ma sono anche definiti non correttamente.

L'International Organization for Standardization (ISO)[21] è la più importante organizzazione mondiale di promulgazione di standard industriali. L'Institute of Electrical and Electronic Engineers (IEEE)[22] è un'associazione interna-

zionale di scienziati professionisti con l'obiettivo della promozione delle scienze tecnologiche.

Secondo il glossario standard di IEEE:

- la verifica è il processo di valutazione di un sistema o di componenti dello stesso per determinare se i prodotti di una data fase di sviluppo soddisfa le condizioni imposte all'inizio di quella fase.
- la validazione è il processo di valutazione di un sistema o di componenti dello stesso durante o alla fine del processo di sviluppo per determinare se i requisiti specificati sono soddisfatti.

Secondo lo standard ISO 9000:

- la verifica è la conferma data dalla fornitura di evidenza oggettiva della soddisfazione delle specifiche richieste
- la validazione è la conferma data dalla fornitura di evidenza oggettiva della soddisfazione dei requisiti di uno specifico uso previsto o dell'applicazione

5.3 Metriche di collaudo

Il cambiamento apportato dal nuovo framework ha modificato la modalità di gestione delle richieste gestite dalle applicazioni in esame. Di conseguenza diventa più difficile confrontare le misure ottenute da basi di codice sorgente così diverse.

Per il solo nuovo framework è stato valutato il grado di retrocompatibilità, mentre per tutti i sistemi MARiS, MN1, Zend Framework, Symfony, CakePHP e il nuovo sviluppato sono state considerate varie metriche di prestazione.

5.3.1 Retrocompatibilità

Per retrocompatibilità di un programma si intende la capacità dello stesso di usare file e dati creati con una versione precedente dello stesso programma, o di fornire tutte le funzionalità della versione precedente.

È un concetto che esprime la relazione tra due componenti, piuttosto che essere un attributo di solo uno dei due, stabilendo una relazione diretta e storica.

È un caso particolare della compatibilità semplice, in cui un'interfaccia definita permette l'interoperabilità tra componenti e prodotti sviluppati separatamente.

La retrocompatibilità è importante perché elimina la necessità di ricominciare da capo ogni volta che si aggiorni ad una nuova versione o a un nuovo prodotto. Tuttavia a volte può essere necessario sacrificare la retrocompatibilità per sfruttare nuove tecnologie.

5.3.2 Prestazioni

Il collaudo delle prestazioni è eseguito per determinare come un sistema si comporta in termini di reattività e stabilità in certe condizioni di carico. Può servire anche per investigare, misurare e validare proprietà di qualità come scalabilità, affidabilità e quantità delle risorse di sistema usate.

In questo caso è stato eseguito il benchmarking dei sistemi, ovvero è stata determinata la capacità dei vari software di svolgere più o meno velocemente la loro funzione. Sono state usate le seguenti metriche:

- tempi di risposta del server web, in millisecondi (ms)
- tempi di generazione delle pagine, in millisecondi (ms)
- throughput delle applicazioni, in richieste al secondo

Il throughput è inteso come numero di richieste servite dal server per unità di tempo. Il tempo è calcolato a partire dall'inizio della prima richiesta al termine dell'ultima, inclusi intervalli di attesa tra le richieste, in quanto si suppone rappresentino il carico sul server.

5.4 Strumenti di collaudo delle prestazioni

Per determinare il tempo di generazione delle pagine, e solo questo, è stata realizzata una classe apposita, la classe *Timing*. Questa classe è stata usata in tutti i progetti, aggiungendo il codice necessario all'inizio e alla fine del flusso di gestione della richiesta del framework o applicazione specifica. La classe è in grado di mostrare il tempo impiegato direttamente sulla pagina generata, sia memorizzarlo in una tabella del database di test.

Per la misura delle altre metriche, sono stati utilizzati diversi strumenti:

- Apache Jmeter
- ab Apache Benchmark

Apache JMeter[23] è un'applicazione desktop open source sviluppata in linguaggio Java e progettata per collaudare comportamenti funzionali e misurare prestazioni in generale. Nata come strumento di benchmark di applicazioni web, col tempo è stata estesa per il collaudo di altre funzioni.

Apache JMeter può essere usata per misurare le prestazioni di risorse statiche e dinamiche, come file, script Perl, oggetti Java, basi di dati, server FTP e altro ancora. Può essere impiegata per simulare pesanti carichi a server, reti o oggetti, per l'analisi delle prestazioni in diverse condizioni di carico. Inoltre dispone di estensioni per visualizzare graficamente [fig. 5.1] i risultati del test.

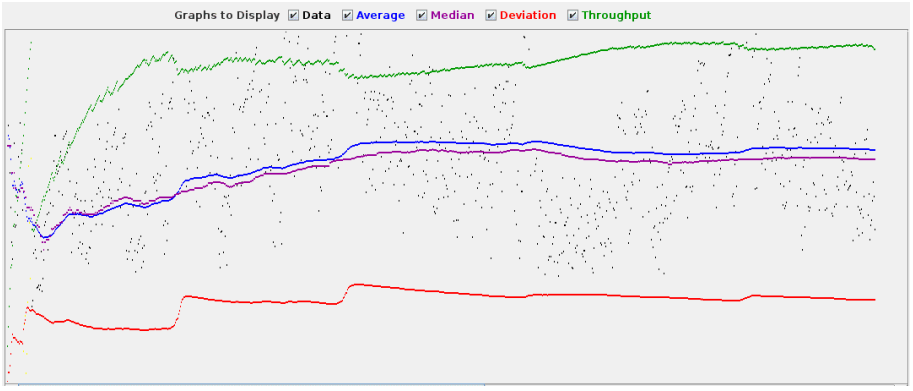

Figura 5.1: Esempio di grafico prodotto da Apache JMeter

ApacheBench AB[24] è uno programma open source usabile da terminale per la misura delle prestazioni di server web HTTP. Originariamente progettato per il solo server Apache, è ora generico a sufficienza per collaudare qualunque server web. AB è un programma a singolo thread, perciò è in grado di mostrare quante richieste al secondo e con quale latenza è in grado di servire un server a singolo thread. Per server a thread multipli, o comunque progettati per usare processori multi-core, istanze differenti di AB devono essere eseguite da client diversi. Il vantaggio di AB è di essere compreso nell'installazione standard del server web Apache, e come questo è distribuito con licenza Apache.

Gli esiti dei test ottenuti dai due strumenti non possono essere confrontati tra di loro. Sviluppati con linguaggi di programmazione diversi, con diverso supporto al protocollo HTTP e con funzioni diverse, i valori risultanti possono essere notevolmente differenti. Ad esempio *Apache Benchmark* richiede al server web esclusivamente la risorsa indicata, che in questi test è la pagina HTML generata, ma non richiede risorse associate come fogli di stile associati o file Javascript. Devono perciò essere confrontati tra di loro i risultati ottenuti dai vari sistemi con i due strumento di misura.

5.4.1 Modalità di collaudo

Ai fini del collaudo ci è concentrati sulla funzionalità di Agenda 2.3 dell'applicazione MN1. Sono stati collaudati le applicazioni MARiS, MN1, il nuovo framework e i tre framework analizzati in 3.5. Con questi ultimi è stata realizzata una parziale reimplementazione della funzionalità di Agenda, utilizzando per quanto possibile il medesimo codice sorgente e le medesime interrogazioni alla base di dati usati da MN1.

Poiché i database di MARiS e MN1, e quindi del sistema integrato, non sono coincidenti, le interrogazioni al database non sono e non possono essere identiche.

Per la valutazione delle prestazioni dei framework è stato realizzato il seguente ambiente:

- il server di database è stato installato nella stessa macchina del server web
- è stato creato un apposito database di test *TEST_FRAMEWORKS*
- sono state ricreate le tabelle AGENDA, FILLERS, ORDERS, PATIENT e WARDS, usate dalla funzionalità di Agenda
- è stata creata una classe *TestFrameowkr*, riusata in ogni framework, che usa le stesse funzioni di accesso alla base di dati di MARiS e MN1
- sono stati installati i framework in esame ed è stata mantenuta la configurazione predefinita
- in ogni framework è stato creato un controller per l'accesso ai dati tramite la suddetta classe

- in ogni framework è stata creata una vista pressoché identica nei vari sistemi, salvo le diverse modalità di accesso ai parametri ricevuti dal controller.

La procedura di autenticazione è stata automatizzata per i sistemi MARiS, MN1 e il nuovo framework: poiché JMeter è in grado di fornire parametri con la richiesta di tipo POST mentre Apache Benchmark non lo è, le variabili per l'autenticazione sono state inviate direttamente con l'URL, in modo che in tutti i test la modalità fosse esattamente la stessa. La fase di autenticazione non è stata inclusa nelle implementazioni con i framework, in quanto sarebbe dovuta essere implementata con gli strumenti resi disponibili dagli stessi, influenzando i risultati con fattori non controllabili direttamente. Perciò i valori che ci si aspetta di ottenere per i framework collaudati potrebbero essere superiori a valori realistici.

Con gli strumenti di test del server web Apache presentati in Strumenti di collaudo delle prestazioni sono stati simulati 10 client che hanno inviato in totale 1000 richieste per ciascun test. Questi valori si sono dimostrati essere quelli più equilibrati rispetto alla piattaforma di collaudo: quantità inferiori di richieste hanno prodotto risultati inficiati dal possibile avvio dei thread del server web, quantità superiori hanno prodotto risultati favoriti dalle capacità di caching del server, condizione che normalmente non si verifica nel dominio in esame.

La simulazione è stata ripetuta 5 volte per ogni framework. Prima di ogni misura il server web è stato riavviato e la tabella con i tempi di generazione delle pagine svuotate. Al termine sono stati calcolati i valori medi delle misure svolte.

5.5 Risultati

5.5.1 Retrocompatibilità

Il nuovo framework realizzato è stato espressamente progettato per adattarsi alla situazione attuale: la compatibilità con il codice legacy permette di integrare un modulo funzionale per volta mantenendo inalterati gli altri. Perciò il collaudo del tasso retrocompatibilità è consistito solamente nella verifica diretta del corretto funzionamento dei moduli esistenti.

Esattamente come atteso, i moduli realizzati con codice legacy sono risultati

funzionanti, tranne per tre casi particolari in cui il campo nascosto *tab_button* usato dal *Dispatcher* viene ridefinito in modo improprio, e che dovranno essere gestiti dai controller appositi.

5.5.2 Prestazioni

I valori ottenuti con i due strumenti differiscono notevolmente tra loro. Le tabelle Apache JMeter e Apache Benchmark presentato i risultato sintetici per ciascun strumento usato. Le tabelle con tutte le misure sono disponibili in appendice Risultati benchmark.

Il valore del tempo di risposta di JMeter comprende anche il tempo necessario al programma per la creazione della richiesta e il ricevimento totale della risposta. Poiché JMeter è scritto in linguaggio Java questi tempi risultano abbastanza prolungati.

Coi primi test Symfony ha fornito risultati molto scadenti (pur se non quanto CakePHP) molto lento, circa la metà dei concorrenti, tranne Cake. Scoperto che installazione di default usa un kernel del framework di sviluppo, con molto codice di debug. L'abilitazione della modalità produzione ha condotto a risultati in linea coi concorrenti.

Apache JMeter

Considerando il throughput come metrica principale, al primo posto [Tab. 5.1] troviamo MARiS, quindi il nuovo framework, seguito da Zend Framework, Symfony, MN1 e CakePHP.

In generale, i tre framework collaudati sono relativamente lenti, pur implementando una versione parziale della funzionalità di Agenda. Tra di essi spicca Zend, abbastanza sorprendentemente stante la non eccelsa nomea. CakePHP è sorprendentemente molto lento, addirittura al punto di ipotizzare un problema con l'installazione dello stesso. Tuttavia una nuova installazione non ha prodotto risultati apprezzabilmente differenti, come anche la disattivazione di tutte le modalità di debug. Come si vede nel test con Apache Benchmark, non si tratta di un caso isolato.

Tra i RIS, MN1 risulta più lento del previsto. Il nuovo framework utilizza il database frutto dell'integrazione, ma la base del codice è quella di MN1. La

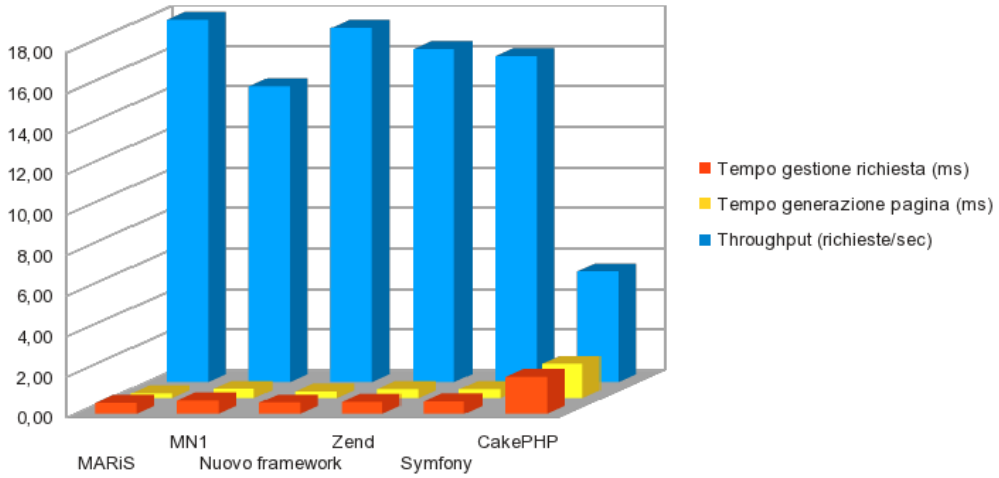

Figura 5.2: Risultati misure JMeter

Metriche	MARiS	MN1	NF[a]	Zend	Symfony	Cake
Throughput[b]	17,87	14,59	17,47	16,42	16,07	5,454
T. risposta[c]	0,535	0,658	0,553	0,712	0,602	1,826
T. generazione[d]	0,227	0,445	0,321	0,516	0,439	1,679

[a] Nuovo Framework

[b] Richieste/s

[c] Tempo di risposta (millisecondi)

[d] Tempo di generazione pagina (millisecondi)

Tabella 5.1: Prestazioni rilevate con Apache JMeter

funzionalità di agenda è praticamente la stessa di MN1, ma già la ristrutturazione della pagina principale è sufficiente a ottenere prestazioni migliori. Si può solo immaginare il margine di miglioramento col procedere dell'integrazione e ottimizzazione.

Non sorprende il primo posto di MARiS, il database è più semplice, come anche le interrogazioni e la pagina prodotta.

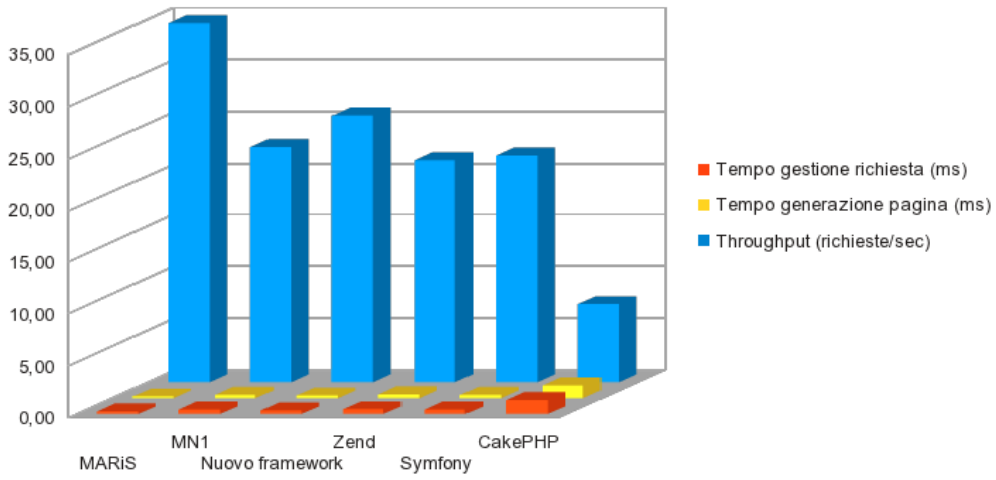

Figura 5.3: Risultati delle misure di benchmarking con Apache Benchmark

Apache Benchmark è sviluppato secondo il modello a thread singolo, ma il fatto di essere stato sviluppato in linguaggio C ha permesso di ottenere risultati apprezzabili. Sicuramente le misure ottenute sono favorite dal fatto che Apache Benchmark non richiede le risorse associate alla pagina, come fogli di stile CSS e file Javascript. Tuttavia, pur se configurato in tal senso, Apache Benchmark non è stato in grado di sfruttare il protocollo HTTP 1.1, che, riusando la stessa connessione Transmission Control Protocol (TCP) per ricevere più risorse dal server, consente di ridurre la latenza. Apache Benchmark non ha stabilito nessuna connessione che usasse la funzione *KeepAlive*.

Considerando il throughput come metrica principale, al primo posto [Tab. 5.2] troviamo MARiS, quindi il nuovo framework, seguito da MN1, Zend Framework, Symfony, e CakePHP.

Metriche	**MARiS**	**MN1**	**NF**[a]	**Zend**	**Symfony**	**Cake**
Throughput[b]	34,62	22,72	25,75	21,45	21,91	7,49
T. risposta[c]	0,266	0,4272	0,351	0,451	0,444	1,323
T. generazione[d]	0,150	0,316	0,242	0,340	0,327	1,218

[a] Nuovo Framework

[b] Richieste/s

[c] Tempo di risposta (millisecondi)

[d] Tempo di generazione pagina (millisecondi)

Tabella 5.2: Prestazioni rilevate con Apache Benchmark

Con questo strumento di benchmarking, MARiS è sorprendentemente veloce. Solo per MARiS e solo per questo strumento di benchmarking, è stata realizzata una seconda sessione di test di MARiS [tab. B.2], realizzata in un momento successivo, che ha ottenuto risultati simili pur se non così elevati.

Le considerazioni riguardo l'ottimizzazione possibile per MN1 sono identiche a quelle viste per il collaudo con JMeter.

I tre framework collaudati stavolta sono tutti in coda, ma mantengono le posizioni relative tra loro. Le loro prestazioni sono simili a quelle di MN1 e del nuovo framework pur svolgendo meno attività. La realizzazione completa della funzionalità di Agenda non può che ridurre ulteriormente le prestazioni.

Tuttavia, la valutazione delle prestazioni di CakePHP è stata più articolata del previsto. In particolare, esso si è dimostrato pressoché incompatibile con la modalità *KeepAlive* di Apache Benchmark, che realizza richieste con il protocollo HTTP 1.1. Con tale modalità attiva lo strumento di benchmark non è stato in grado di completare nessuna connessione. Disabilitando tale modalità è stato ottenuto un throughput di circa *1.6 richieste/sec*, estremamente basso. Disabilitando del tutto il protocollo HTTP 1.1 a livello del server web è stato ottenuto il throughput nella tabella Apache Benchmark, comunque non soddisfacente.

Chiaramente la release di CakePHP utilizzata soffre di problemi di incompatibilità con le versioni usate di sistema operativo e server web. La versione 2.0 di CakePHP ha introdotto una nuova modalità di generazione della risposta che fa uso del protocollo HTTP 1.1, salvo diversa configurazione del framework. Le prime release di tale versione soffrivano di incompatibilità [25, 26] con Apache Benchmark, evidentemente non del tutto risolte.

5.6 Conclusioni

Le misure realizzate hanno sostanzialmente confermato due fatti:

- i framework di sviluppo rapido introducono sempre un degrado delle prestazioni
- non è sufficiente usare un solo strumento di misura

Un risultato che spicca tra gli altri è il confronto tra MN1 e il nuovo framework. Le già limitate ottimizzazioni introdotte per adattare il ciclo di gestione codice legacy al nuovo framework sono sufficienti per ottenere un lieve incremento delle prestazioni. È solo possibile intuire i vantaggi prestazionali ottenibili estendendo il grado di efficienza a tutto il codice legacy.

La sorpresa negativa è CakePHP: questo framework è stato decisamente problematico da misurare. CakePHP è sicuramente il più semplice da installare, imparare ed estendere tra i tre framework, tuttavia non sono del tutto risolte criticità della nuova versione 2.0. Non è stata verificata la precedente versione 1.3, destinata a scomparire, è perciò auspicabile una pronta risoluzione di tutti i problemi con il protocollo HTTP 1.1.

Capitolo 6

Conclusioni

6.1 Considerazioni

L'analisi svolta in questo lavoro di tesi ha posto le basi per una fattiva integrazione di due sistemi informativi radiologici, di cui uno dedicato alla specialità di Medicina Nucleare.

Il dominio sanitario si è rivelato particolarmente complesso; l'ambiente informativo si compone di molteplici applicativi che, interagendo tra di loro, realizzano gran parte delle linee guida IHE, che direttamente sul campo si sono dimostrate efficaci.

Se per la disciplina di Radiologia esistono soluzioni dedicate e affermate, altrettanto non si può dire per Medicina Nucleare. Alcune sottili ma importanti peculiarità impongono l'uso di strumenti adattati a questo ambiente, se non specifici. La ristretta dimensione del mercato non ha permesso la nascita di applicativi pensati per questa specialità, da cui la nascita di MN1.

Tuttavia, sviluppare e mantenere due applicazioni, simili ma non uguali, ha un costo non giustificato dal numero relativamente ridotto di diversità tra le due applicazioni MARiS e MN1. L'integrazione delle due permetterà di concentrare gli sforzi per le parti comuni e di accelerare lo sviluppo delle componenti particolari.

Inoltre, un processo di sviluppo scarsamente guidato da tecniche e paradigmi affermati e ricorrenti ha determinato la produzione di codice sorgente sempre più difficile da adattare, espandere e correggere. L'occasione fornita dall'esigenza

di integrare i due sistemi è stata il momento ideale per realizzare una prima parziale riprogettazione, sia per revisionare il codice sorgente che per favorire l'integrazione stessa.

I vantaggi ottenuti dal nuovo framework così realizzato sono molteplici:

- l'integrazione può avvenire gradualmente, secondo paradigmi noti e qui fissati
- aumenterà l'efficienza del processo di sviluppo
- i due sistemi sono più modulari
- potranno essere introdotte nuove funzionalità più facilmente.

Come tutte le innovazioni di una certa portata, anche in questo caso si sono presentano alcuni svantaggi:

- un nuovo paradigma deve essere compreso e acquisito
- non è stato possibile integrare del tutto il codice legacy, alcune (piccole) parti sono state forzatamente riprogettate
- un framework potrebbe in prospettiva produrre prestazioni leggermente inferiori.

Non tutte queste condizioni sono però a discapito del nuovo framework; infatti il nuovo modello di sviluppo consentirà di produrre codice più collaudabile, quindi potrà essere dedicato più tempo per ottimizzarlo. Paradigmi affermati consentiranno a nuovi sviluppatori di impiegare meno tempo per apprendere il modello di esecuzione dell'applicazione. I benchmark hanno dimostrato come le prestazioni del nuovo framework sono simili se non superiori a quelle del sistema MN1, ma con molte chiamate a funzione e classi in più. Le limitate ottimizzazioni introdotte hanno già aumentato l'efficienza dell'applicazione.

6.2 Sviluppi futuri

Di rado la prima versione di un nuovo software è realizzata nel miglior modo possibile, il nuovo framework realizzato non fa eccezione. Molteplici migliorie sono possibili e auspicabili, e il design permette la graduale loro introduzione.

Sicuramente il pattern MVC implementato in modo semplificato può essere perfezionato: la logica applicativa vera e propria può essere facilmente estratta dai controller creando delle apposite classi che imitino il comportamento dei Model.

Inoltre, è ipotizzabile l'introduzione di uno strumento ORM per astrarre il livello dei dati, rendendosi indipendenti dal particolare server di database. Ad oggi la base di dati non fa uso di vincoli di chiave esterna proprio per assicurare tale autonomia. Inoltre la collaudabilità del codice risulterebbe migliorata dall'uso di uno strumento affermato; ad esempio Symfony fa uso dell'ORM Doctrine, evitando di reinventare la ruota.

Più nel dettaglio, la classe *Registry* può essere rivisitata introducendo il pattern Inversion of Control, implementato col paradigma Dependency Injection[18] , inoculando il registro nelle classi in cui è necessario, diminuendo così il grado di accoppiamento tra di esse e la classe *Registry* stessa.

Chiaramente, quando tutte le pagine legacy saranno convertite al nuovo framework non sarà più necessario mantenere il codice di compatibilità, migliorando ulteriormente le prestazioni.

Dal lato delle viste, durante la riprogettazione del codice legacy è auspicabile realizzare una libreria di controlli PHP / HTML per uniformare il più possibile comportamento e aspetto di una moltitudine di elementi riusati frequentemente.

Inoltre, l'ottimizzazione delle viste consentirà in futuro di utilizzare tecniche come Asynchronous JavaScript and XML (AJAX) senza particolari modifiche alle stesse, rendendo l'applicazione più efficiente e interattiva.

6.3 L'esperienza personale

Il processo di evoluzione e perfezionamento del framework realizzato potrà proseguire di pari passo con quello dell'integrazione di MARiS e MN1. Ora le prospettive di integrazione sono ben affermate e stabilite. L'efficienza del processo di sviluppo, dopo un periodo di transizione, non potrà che migliorare, grazie a tutti i vantaggi visti.

Il lavoro qui svolto afferma l'importanza dell'esistenza di progetti open source come questi due applicativi. Essi permettono la corretta adozione e diffusione di standard internazionali, e forniscono prodotti a basso costo a tutte le aziende

ospedaliere, anche relativamente piccole quali non potrebbero permettersi RIS troppo costosi.

L'esperienza personale realizzata nei reparti di Radiologia e Medicina Nucleare è stata sicuramente appassionante. L'ambiente ospedaliero s'è dimostrato essere molto complesso, articolato, inatteso, molto più di quanto non appaia agli utenti che loro malgrado, o per fortuna, sono costretti ad usufruirne. Aver osservato il dietro le quinte di un tale complesso cambia la visione che normalmente si può avere nelle occasioni nelle quali si è dalla parte dei pazienti. Ancora non è chiaro dove il futuro condurrà la sanità italiana, ma certamente molte persone lavorano e usano il loro tempo per migliorare la sua efficienza.

Appendice A

Acronimi

ACID	Atomicity, Consistency, Isolation, Durability
ADT	Admsission Dimission Transfer
ACR	American College of Radiology
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATNA	Audit Trail and Node Authentication
CTN	Central Test Node
DBMS	DataBase Management System
DICOM	Digital Imaging and Communications in Medicine
eco-TC	Ecografia-Tomografia Computerizzata
EHR	Electronic Healthcare Record
EPR	Electronic Patient Record
EVN	Event
HIMSS	Healthcare Information and Management Systems Society

HIT	Health Information Technology
HL7	Health Level 7
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronic Engineers
IHE	Integrating the Healthcare Enterprise
ISO	International Organization for Standardization
LAN	Local Area Network
MN	Medicina Nucleare
MSH	Message Header
MVC	Model - View - Controller
MyISAM	My Indexed Sequential Access Method
NEMA	National Electrical Manufacturers Association
ORM	Object Relational Mapping
PACS	Picture Archiving and Comunication System
PIR	Patient Information Reconciliation
PHP	PHP: Hypertext Preprocessor
PID	Patient ID
RIS	Radiology Information System
RM-PET	Risonanza Magnetica-Positron Emission Tomography
RSNA	Radiologica Society of North America
SINR	Simple Image and Numeric Report
SIS	Sistema Informativo Sanitario

TCP Transmission Control Protocol

TDEA Triple Data Encryption Algorithm

URL Uniform Resource Locator

Appendice B

Risultati benchmark

B.1 MARiS

B.1.1 Apache JMeter

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	17,78	0,537	0,222
2	18,18	0,524	0,213
3	17,91	0,535	0,238
4	17,52	0,548	0,239
5	17,97	0,531	0,225

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

B.1.2 Apache Benchmark

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	35,69	0,262	0,141
2	31,29	0,280	0,164
3	35,05	0,260	0,158
4	34,76	0,271	0,143
5	36,31	0,259	0,142

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

Tabella B.1: Prima sessione di test

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	28,09	0,355	0,225
2	29,52	0,333	0,228
3	33,60	0,300	0,211
4	33,77	0,317	0,219
5	32,88	0,311	0,216

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

Tabella B.2: Sessione di test di controllo

B.2 MN1

B.2.1 Apache JMeter

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	14,34	0,667	0,460
2	14,79	0,639	0,433
3	14,48	0,669	0,456
4	14,35	0,669	0,451
5	14,99	0,648	0,424

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

B.2.2 Apache Benchmark

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	22,23	0,430	0,319
2	23,36	0,425	0,310
3	22,15	0,440	0,322
4	22,94	0,417	0,316
5	22,92	0,424	0,313

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

B.3 Nuovo framework

B.3.1 Apache JMeter

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	16,90	0,572	0,337
2	17,74	0,542	0,313
3	17,21	0,557	0,316
4	17,82	0,543	0,321
5	17,66	0,551	0,319

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

B.3.2 Apache Benchmark

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	25,01	0,351	0,251
2	25,35	0,354	0,245
3	26,50	0,349	0,235
4	26,31	0,360	0,238
5	25,60	0,342	0,242

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

B.4 Zend

B.4.1 Apache JMeter

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	17,41	0,556	0,392
2	16,93	0,575	0,408
3	15,98	0,596	0,432
4	16,03	0,605	0,431
5	15,77	0,615	0,459

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

B.4.2 Apache Benchmark

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	21,01	0,454	0,348
2	20,68	0,454	0,352
3	22,10	0,441	0,332
4	21,29	0,461	0,339
5	22,16	0,446	0,331

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

B.5 Symfony

B.5.1 Apache JMeter

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	16,05	0,603	0,439
2	16,09	0,602	0,430
3	16,46	0,588	0,428
4	15,62	0,621	0,443
5	16,16	0,598	0,455

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

B.5.2 Apache Benchmark

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	20,20	0,450	0,361
2	22,41	0,440	0,323
3	22,67	0,438	0,314
4	22,53	0,440	0,321
5	21,74	0,454	0,316

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

B.6 CakePHP

B.6.1 Apache JMeter

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	5,27	1,857	1,728
2	5,33	1,844	1,680
3	5,38	1,825	1,680
4	5,32	1,841	1,681
5	5,98	1,764	1,628

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

B.6.2 Apache Benchmark

Test	Throughput[a]	T. risposta[b]	T. generazione[c]
1	7,43	1,331	1,220
2	7,52	1,321	1,221
3	7,48	1,318	1,219
4	7,51	1,325	1,213
5	7,52	1,318	1,217

[a] Richieste/s

[b] Tempo di risposta (millisecondi)

[c] Tempo di generazione pagina (millisecondi)

Elenco delle tabelle

Elenco delle figure

Bibliografia

[1] RSNA, "Ihe product registry." documentazione online, January 2012. http://product-registry.ihe.net.

[2] ISO, "Standard iso hl7." documentazione online, January 2012. http://www.iso.org/iso/catalogue_detail.htm?csnumber=44428.

[3] RSNA, "Ihe technical frameworks." documentazione online, January 2012. http://www.ihe.net/Technical_Framework/.

[4] "Dcm4che." documentazione online, January 2012. http://www.dcm4che.org.

[5] "mirth." documentazione online, January 2012. http://www.mirthcorp.com/community/overview.

[6] Garante per la protezione dei dati personali, "Testo unico privacy." documentazione online, February 2012. http://www.garanteprivacy.it/garante/doc.jsp?ID=1311248.

[7] Repubblica italiana, "Codice dell'amminstrazione digtale." documentazione online, February 2012. http://www.camera.it/parlam/leggi/deleghe/testi/05082dl.htm.

[8] R[affaella] Tono, "Natural language processing e tecniche semantiche per il supporto alla diagnosi." documentazione online, February 2012. http://tesi.cab.unipd.it/25036/1/Tesi_per_pdf.pdf.

[9] a-thon, "Opportunita commerciali." documentazione online, February 2012. http://www.a-thon.it/.

[10] M. AB. Mondadori Informatica, 1st ed., 2006. ISBN: 8-861-14004-1.

[11] M. AB, "Confronto tra innodb e myisam." http://www.mysql.com/why-mysql/white-papers/mysql_5.5_perf_myisam_innodb.php, 2012. [Online; accessed 26-January-2012].

[12] "Single page interface manifesto."

[13] R. C. Martin. Prentice Hall, 1st ed., 2009. ISBN: 0-132-35088-2.

[14] Zend Technologies Ltd., "Zend framework." documentazione online, January 2012. http://framework.zend.com/.

[15] Sensio Labs, "Symfony." documentazione online, January 2012. http://symfony.com/.

[16] Cake Software Foundation, Inc., "Cakephp." documentazione online, January 2012. http://cakephp.org/.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Addison-Wesley, 1st ed., 1995. ISBN: 0-201-63361-2.

[18] M. Fowler, "Dependenct Injection." documentazione online, February 2012. http://martinfowler.com/articles/injection.html.

[19] ISO/IEC, "Triple data encryption algorithm." documentazione online, January 2012. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=37972.

[20] M. Fowler, "Patterns of Enterprise Application Architecture," 2012. ISBN: 0-321-12742-0.

[21] ISO, "International organization for standardization." documentazione online, January 2012. http://www.iso.org/.

[22] IEEE, "Institute of electrical and electronic engineers." documentazione online, January 2012. http://www.ieee.org/.

[23] A. Foundation, "Apache JMeter." documentazione online, February 2012. http://jmeter.apache.org/.

[24] A. Foundation, "Apache Benchmark." documentazione online, February 2012. http://httpd.apache.org/docs/2.0/programs/ab.html.

[25] "Forum CakePHP." documentazione online, February 2012. http://cakephp.lighthouseapp.com/projects/42648/tickets/2175.

[26] "Forum CakePHP." documentazione online, February 2012. http://cakephp.lighthouseapp.com/projects/42648/tickets/1846-cakephp-20-rfc-issue-with-apache-ab-and-cakeresponse.

[27] RSNA. documentazione online, January 2012. http://www.rsna.org/.

[28] RSNA, "Integrating the healthcare enterprise." documentazione online, January 2012. http://www.rsna.org/Informatics/ihe.cfm.

[29] RSNA, "Ihe integration profiles." documentazione online, January 2012. http://www.ihe.net/profiles/.

[30] Hl7, "Health level 7." documentazione online, January 2012. http://www.hl7.org/.

[31] I[an] Sommerville. Pearson Education, 8th ed., 2007. ISBN: 978-88-7192-354-3.

[32] C. Larman. Prentice Hall, 1st ed., 2005. ISBN: 0-131-48906-2.

[33] H. Erdogmus and O. Tanir. Springer, 1st ed., 2002. ISBN: 0-387-95109-1.

[34] S. A. Gabarro. Wiley & Sons, 1st ed., 2007. ISBN: 0-471-77391-3.

[35] J. Langr. Prentice Hall, 1st ed., 2005. ISBN: 0-131-48239-4.

[36] L. Koskela. Greenwich Manning, 1st ed., 2008. ISBN: 1-932-39485-0.

[37] M. Feathers, "Working effectively with legacy code." documentazione online, 2002. [Online; accessed 31-November-2011].

Ringraziamenti

Vorrei ringraziare Mosca Paolo e il dott. Cecchin Diego, che sono stati il mio riferimento per tutto il tempo del lavoro di tesi.

Vorrei ringraziare il Prof. Ferrari, per aver reso possibile la realizzazione di questo progetto.

Un ringraziamento speciale ai miei genitori, che mi hanno sempre sostenuto, e alla mia compagna Lucia.