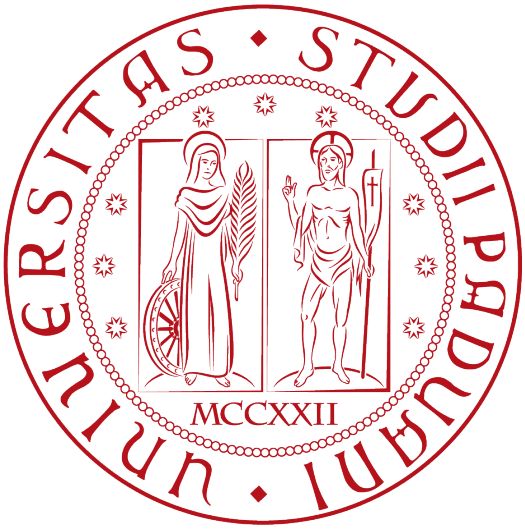
Università degli studi di Padova

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA IN INFORMATICA



Consolidamento di un sistema esperto per la gestione della sicurezza in azienda

Tesi di laurea

Relatore

Prof.ssa Ombretta Gaggi

Laureando

Fabio Ros
609724

Anno Accademico 2015-2016

Indice

Elenco delle figure

Elenco delle tabelle

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Fabio Ros presso l'azienda Moku S.r.l. Il progetto di stage prevede l'analisi, la progettazione e lo sviluppo di alcune delle logiche del sistema esperto in coordinamento con il committente ed un consulente della sicurezza. Nello specifico è prevista l'implementazione delle componenti software sia lato backend, mediante l'utilizzo di Ruby on Rails, sia lato frontend simultaneamente alla definizione delle regole di gestione del sistema esperto utilizzando il framework Drools.

1 Introduzione

1.1 L'azienda

Figura 1: Logo dell'azienda ospitante - Moku S.r.l.

Moku S.r.l è una startup nata nel 2013 e situata in H-Farm. L'organizzazione del lavoro è supportata dalla suite Atlassian che offre un servizio cloud per la gestione dei compiti e dei progetti. L'azienda lavora a stretto contatto con il cliente definendo requisiti, user experience dei prodotti realizzati ed opportunità di business. Obiettivo primario dell'azienda è la qualità del software e l'utilizzo di tecnologie recenti poiché l'alto coefficiente innovativo dei prodotti realizzati rende necessaria una costante manutenzione che si vuole sia il più agevole possibile.

1.1.1 Origine

Il nome della startup, in hawaiiano, significa *isola*. Questo perché l'idea iniziale era una applicazione web per studenti dove un *"moku"* era visto come uno spazio personale (un'isola), ma allo stesso tempo uno spazio di collaborazione, rappresentando un *arcipelago* di isole collegate tra loro. In origine si basava sulla realizzazione di una applicazione il cui uso era riservato principalmente agli studenti. Ad ogni utente era data la possibilità di caricare i propri documenti in diverse tipologie di formato. Una volta aperti, era possibile prendere degli appunti su un layer posto sopra al documento caricato. Questo può essere condiviso con i propri collaboratori. Era anche possibile registrare la lezione e mettere delle note anche alla traccia audio, la quale si poteva sincronizzare con un particolare documento.

1.1.2 Evoluzione

Ora le attività dell'azienda si concentrano nella consulenza IT, in particolare nella realizzazione di software innovativo a supporto delle esigenze concrete dei clienti. Il progetto da me svolto è un chiaro esempio di questo mutamento, infatti il prodotto finito risulterà di completa proprietà di un'azienda esterna.
Ora l'azienda, quindi, sviluppa solo marginalmente software proprietario, ma sviluppa principalmente su commissione di aziende esterne.

1.2 Organizzazione del lavoro

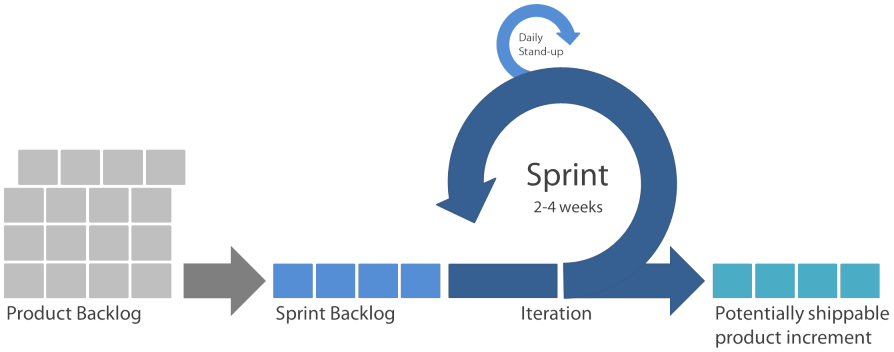


Figura 2: Metodo agile scrum.

All'interno dell'azienda il lavoro viene organizzato secondo il metodo agile scrum.
Il modello agile scrum prevede la suddivisione di un progetto in blocchi detti *Sprint* ognuno dei

quali deve produrre un avanzamento del software. Prevede, inoltre la suddivisione dei compiti in *task*, che possono essere visti come una attività abbastanza piccola da essere svolta da una sola persona nell'arco di 4/8 ore. È previsto un rapporto a stretto contatto con il cliente e la pianificazione di frequenti meeting con la funzione di punti di controllo, dove verificare lo stato di avanzamento del progetto rispetto a quanto atteso. È stato scelto questo modello poiché l'azienda sviluppa in maggioranza prodotti innovativi, quindi con tecnologie in evoluzione, e soggetti a continue variazioni dei requisiti rendendo necessaria un meccanismo di controllo flessibile. Nello specifico del progetto in analisi, la continua variazione delle norme di legge in vigore provoca continui mutamenti nell'implementazione del software ed un rapporto a stretto contatto con il cliente diventa indispensabile al fine di garantire la conformità del servizio.

1.3 Strumenti a supporto dell'organizzazione del lavoro

1.3.1 Atlassian Suite

Figura 3: Logo della suite Atlassian

Per l'organizzazione del lavoro, ci si è appoggiati alla suite offerta da Atlassian, che fornisce tutti gli strumenti necessari alla gestione di codice e compiti.
Fa seguito l'elenco degli strumenti utilizzati a supporto delle principali necessità.

1.3.2 Jira - Gestione dei Task e Sprint

Figura 4: Logo di Jira

Per la gestione dei *Task* e degli *Sprint* è stato utilizzato il software Jira della suite Atlassian. Esso permette di creare ed assegnare i task alle persone le quali possono annotare il numero di ore dedicate ad ogni specifico task. È possibile inoltre avere una visione d'insieme dello sprint corrente e dei precedenti, pianificando anche i successivi. È anche possibile collegare i task presenti su Jira con i *commit* presenti su Bitbucket facendo riferimento nel testo del *commit* al codice identificativo del *task* tenendo sotto controllo l'evoluzione del codice per ogni *task*.

1.3.3 Confluence - Gestione della documentazione

Figura 5: Logo di Confluence

Per la gestione della documentazione su norme aziendali e prodotti realizzati, viene utilizzato il software Confluence. Esso fornisce usa sorta di wiki integrato con tutte le applicazioni della suite collegata al progetto. Qui viene stesa la documentazione inerente a tutte le fasi del ciclo di vita del prodotto.

1.3.4 Bitbucket - Versionamento

Figura 6: Logo di Bitbucket

Per quanto riguarda il versionamento, viene utilizzato il servizio Bitbucket della suite. Esso utilizza `git` per la gestione del repository. Per agevolare la cooperazione è stato adottato l'approccio **git-flow**.

1.3.5 Slack - Comunicazione

Figura 7: Logo di Slack

Per quanto riguarda la comunicazione interna ai membri dell'azienda è stato scelto di utilizzare il software gratuito Slack. Si tratta di un sistema di messaggistica che distingue le comunicazioni dirette a persone da quelle dirette a canali inerenti ad un particolare argomento. Le norme aziendali prevedono la creazione di un canale per ogni progetto. Slack mette a disposizione anche un meccanismo di condivisione di file e, caratteristica essenziale, mantiene uno storico di tutti i messaggi di tutte le conversazioni al fine di mantenere un tracciamento delle comunicazioni interne. È fornita dal software anche una efficiente funzionalità di ricerca per recuperare le comunicazioni o i contenuti condivisi.

1.3.6 Errbit - Rilevazione degli errori

Figura 8: Logo di Errbit

Errbit è uno strumento per la rilevazione e gestione degli errori. Esso ha bisogno di un server dove girare e mantiene dei listener in ascolto dei messaggi di errore provenienti dall'applicazione alla quale è collegato. Per ogni errore rilevato, vengono resi disponibili su una pagina web le informazioni relative alla tipologia dell'errore, il backTrace, l'utente per il quale è avvenuto, ed i parametri sottomessi alla pagina in analisi. Sono inoltre esposte da questo framework delle API per AirBrake e Jira, in modo da favorirne l'integrazione. Nello stage è stato fatto uso di questo framework nel server in produzione per essere a conoscenza con esattezza delle criticità del prodotto realizzato in modo da essere reattivi nella risoluzione dei bug rilevati.

1.3.7 Airbrake - Gestione e notifica degli errori

Figura 9: Logo di Airbrake

Airbrake fornisce un sistema di notifiche push centralizzato. La sua caratteristica principale è quella di centralizzare la gestione delle notifiche permettendo un'accurata gestione dei flussi di spedizione dei messaggi e degli ambienti monitorati (sviluppo, staging, produzione).
È stato collegato Airbrake ad Errbit mediante le apposite API. Ad ogni eccezione catturata da ErrBit, AirBrake permette di visualizzare in una dashboard tutte le informazioni relative all'eccezione ed al contesto in cui essa è stata sollevata. È inoltre possibile collegare Airbrake ad altri strumenti ad esempio per la gestione di report periodici.
In questo progetto Airbrake è stato configurato in modo tale che le notifiche provenienti dal monitoraggio dell'ambiente di produzione siano spedite ad un canale di Slack appositamente creato.

2 Il progetto di Stage

Il progetto di stage consiste nell'estensione di un software web based, già sviluppato in azienda, a supporto delle aziende che intendono avvalersi dell'asseverazione in ambito della sicurezza sul lavoro, in particolare nel settore edilizio.
Con asseverazione si intende una scelta volontaria di una impresa edile al fine di dimostrare l'impegno per la prevenzione, salute e sicurezza nei luoghi di lavoro. Opera mediante la certificazione di conformità dei modelli di organizzazione e gestione aziendali e mediante visite a campione nei cantieri. L'asseverazione offre benefici economici alle aziende che la richiedono e garantisce efficacia esimente rispetto alla responsabilità amministrativa delle imprese.
Il software si pone come obiettivo la possibilità di fornire in ogni momento una fotografia dello stato della sicurezza in azienda, al fine di agevolarne il mantenimento nel tempo.
All'inserimento di nuove informazioni, verranno generate nuove domande, scadenze o vincoli. Se i risultati non sono conformi alle norme vigenti, verranno generati degli allarmi, che scompariranno solo nel momento in cui la violazione del vincolo che rappresentano non sia più verificata.
Dal punto di vista tecnico viene utilizzato un sistema esperto per mappare le norme in tema di sicurezza, individuare i punti deboli dell'azienda, ricordare al responsabile le scadenze, conservare ed indicizzare il patrimonio documentale in ambito sicurezza. Il sistema ha una funzione proattiva e dinamica, variando gli allarmi in base alla variazione delle norme e allo stato dell'azienda (es. al crescere o diminuire del numero dei dipendenti, al variare della superficie delle sedi aziendali, dei cantieri...). La webapp si comporta quindi come un consulente digitale per l'azienda, aiutandola a mantenere aggiornato il suo modello di sicurezza e a rispettarlo. Questo le permette all'azienda asseverata di accedere a significativi premi INAIL, di ottenere punti in graduatoria in bandi pubblici e di sollevare il datore di lavoro da responsabilità penali in caso di infortuni legati ad una mala gestione della sicurezza.

2.1 Il prodotto esistente

Il progetto è in una fase di sviluppo avanzato (versione alpha). Alla data di inizio dello stage, il software supporta le aziende con i codici ATECO$_G$ in ambito edilizio, permette l'inserimento di informazioni su sedi, cantieri, dipendenti, organigramma aziendale e la gestione di abitabilità, certificato prevenzione incendi e della documentazione che emerge dal Documento di Valutazione dei Rischi (DVR)$_G$. Vengono poste all'utente più di 400 domande per individuare lo stato di sicurezza.
Il sistema esperto è funzionante, ma va irrobustito e vanno implementate le regole opportune per individuare le criticità nei dati inseriti dall'utente.

2.2 Obiettivi dello stage

Il team ha l'obiettivo di rilasciare una versione beta del software entro la fine del 2015 e quindi procedere al primo test con l'utente finale, una importante azienda del settore edilizio, già individuata. Durante lo stage sono stati posti i seguenti obiettivi:

- Analisi di alcune logiche del sistema esperto in coordinamento con il cliente ed il consulente della sicurezza;
- Progettazione delle funzionalità individuate nella fase di analisi;
- Implementazione del risultato del punto precedente;
- Stesura della documentazione relativa al codice prodotto;
- Rendere il sistema più efficace, semplice ed usabile da parte di un utente competente in materia di sicurezza.

3 Pianificazione del lavoro

4 Tecnologie e strumenti

4.1 Git

Git è un sistema di controllo di versione sviluppato per facilitare la cooperazione nello sviluppo di software. Questo software di versionamento è gratuito ed open-source; è stato sritto da *Linus Torvalds* per lo sviluppo del kernel linux nel 2005 ed attualmente mantenuto da *Junio Hamano*.
Questo strumento permette di sottomettere al server le modifiche e lasciare ad esso l'onere di verificare se la versione appena inoltrata va in conflitto con quella esistente indicando esattamente in quale punto si è presentato il problema.
Viene data la possibilità di generare diversi rami (*branch*) con l'intento di differenziare una nuova versione del codice (*fork*) o per lo sviluppo di una nuova funzionalità, mentre nel primo caso il ramo diverge dal ramo da quale ha origine, nel secondo caso il *branch* ha come fine il ricongiungimento con la sorgente da cui proviene una volta soddisfatto il suo contratto. È proprio da questa idea che nasce il concetto di `git-flow`.

4.1.1 `git-flow`

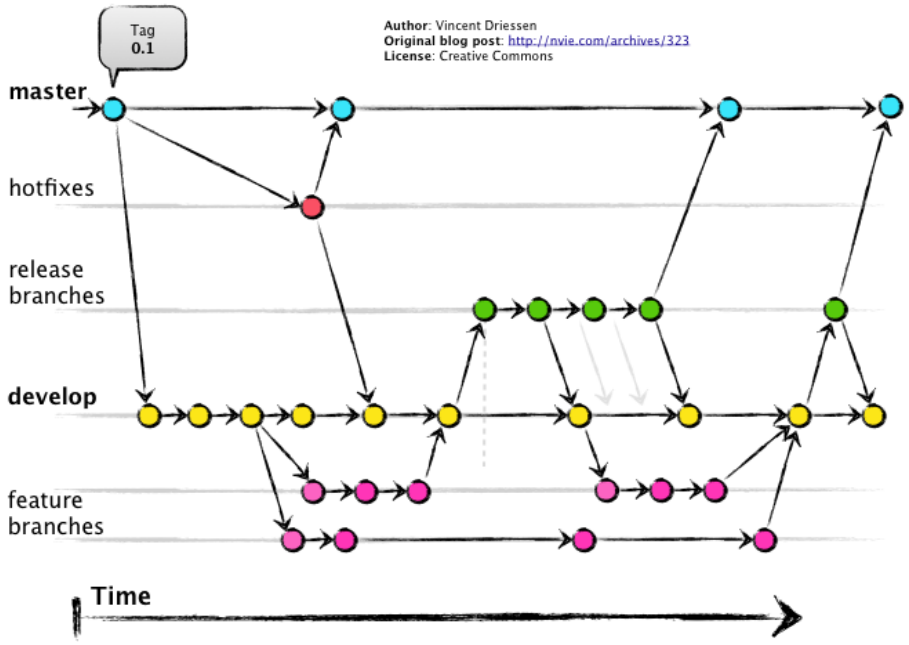


Figura 10: Grafico dei branch di `git-flow`

Git-flow è un set di estensioni di git che offre dei comandi di alto livello sul repository per utilizzare il modello di branching (vedi Figura 10) di *Vincent Driessen*, fornendo il supporto a branching e release management. Esso prevede due branch principali:

- `"master"`: ramo contenente il codice pronto per essere inoltrato nell'ambiente di produzione;
- `"develop"`: ramo di riferimento per l'attività di sviluppo.

Sono previste inoltre altre tre tipologie di branch:

- `feature`: questo tipo di branch viene creato a partire dal ramo principale `develop`; dopo aver portato a compimento una particolare funzionalità, il branch `feature` viene fuso con lo stesso `develop`;
- `release`: questo branch è dedicato alla preparazione di un prodotto alla release; qui è possibile effettuare solo piccoli interventi di rimozione di bug (minor bugfix) in vista di un imminente rilascio nel branch master simultaneamente alla pubblicazione della nuova versione con il comando `git tag`;

- `hotfix`: questo ramo ha lo scopo di risolvere velocemente eventuali bug riscontrati; ha sempre origine dal branch master e, andando a modificare del software già rilasciato, provoca un avanzamento di sul numero di versione.

Vantaggi

- Permette di mantenere una storia chiara e concisa del progetto poiché tutti i commit parziali avvengono nei branch di tipo feature. Nel ramo develop, saranno presenti solo commit derivanti dalla fusione di rami di tipo feature che soddisfano pienamente il contratto per i quali sono stati creati;
- Evita di creare confusione nel repository con commit parziali potenzialmente dannosi e, dualmente, evitare le situazioni in cui lo sviluppo avviene localmente con un solo commit di grosse dimensioni al compimento del contratto del requisito perdendo i vantaggi dello storico dei commit offerti da git.

Svantaggi

- L'utilizzo di git-flow richiede agli sviluppatori di dedicare del tempo in più rispetto al metodo classico. Questo però è vero solo per progetti di piccole dimensioni poiché, al crescere dei progetti, il tempo impiegato a risolvere i conflitti derivanti dall'operare in modo classico risulta superiore a quello dedicato ad un corretto utilizzo di git-flow.
- Un ulteriore svantaggio è dato dal fatto che gli strumenti offerti dagli IDE per la gestione di git-flow non sono sempre all'altezza del loro compito.

4.2 Rubymine

Figura 11: Logo di RubyMine

Rubymine è un IDE prodotto da jetBrains, disegnato appositamente per lo sviluppo con Ruby on Rails.

Vantaggi

- Vengono messi a disposizione molti plugin gratuiti che offrono funzionalità specifiche, non incluse nel pacchetto base;
- Il salvataggio di un file avviene automaticamente ogni volta che esso perde il focus, dando la garanzia di lavorare sempre con file aggiornati;
- Il software è strutturato in modo tale da fornire aiuto attivo nel momento della stesura del codice, come autompletamento ed analisi statica, rendendo più veloce ed efficace il lavoro.

Svantaggi

- Il plugin responsabile del versionamento non funziona sempre alla perfezione, in particolare quando si utilizza git-flow.

4.3 Ruby on Rails

Ruby on Rails, è un framework open source per lo sviluppo di applicazioni web. La sua architettura si basa sul pattern Model-View-Controller (MVC), realizzando a tutti gli effetti un framework full-stack; è dunque possibile realizzare sia la parte di backend che quella di front-end$_G$.

Vantaggi

- Risorse e componenti sono integrati in modo tale che i collegamenti non debbano mai essere impostati manualmente, ma in modo automatico;
- È possibile lavorare utilizzando ActiveRecord, descritto nella sezione 4.3.1. Questo è un grosso vantaggio perché velocizza la stesura del codice e risponde bene ai cambiamenti. Nel progetto svolto, l'utilizzo di ActiveRecord è un fattore determinante poiché è necessario che il software sia sempre conforme alle normative vigenti che cambiano nel tempo;
- È possibile disaccoppiare il backend ed il front-end$_G$ perché in Rails il routing delle risorse è gestito con una interfaccia REST$_G$;
- Sono inoltre messi a disposizione i seguenti ambienti per uno stesso software, al fine di creare una separazione nel progetto:
  - development
  - test
  - production

  Ambienti distinti solitamente risiedono anche su server distinti.
- Sono presenti infine numerose librerie dette *Gemme*, che offrono molteplici funzionalità velocizzando il lavoro.

Svantaggi

- Ruby on Rails è meno performante rispetto ad altri linguaggi, ad esempio Java o Scala, ma di un ordine di grandezza non influente su progetti di medie dimensioni.

4.3.1 ActiveRecord

Active Record è il modulo di Ruby on Rails che gestisce la persistenza dei dati seguendo il pattern *Active Record* di Martin Fowler[1].
Il modulo ActiveRecord di Rails si serve di un database e prevede:

- Ogni tabella del database relazionale è gestita attraverso una classe;
- Una singola istanza della classe corrisponde ad una riga (record) nella tabella associata;
- Alla creazione di una nuova istanza viene creata una nuova riga all'interno della tabella che viene aggiornata ad ogni modifica dell'istanza associata.

Le colonne della tabella rappresentano gli attributi della classe.

Vantaggi

- È possibile associare un diverso database per ogni ambiente predisposto da Rails separando i record memorizzati per lo sviluppo, per i test, o per l'ambiente di produzione;
- È disponibile un meccanismo di gestione delle relazioni molto efficiente che facilita dichiarazione ed utilizzo delle diverse tipologie di relazione tra le tabelle del database;
- Viene messo a disposizione un meccanismo specifico per la gestione dell'evoluzione dello schema del database mediante le `migrazioni` (vedi Appendice A).

Svantaggi

- Richiede una progettazione spesso diversa da quella classica, poiché l'ereditarietà risulta molto più difficile da gestire quando una classe corrisponde ad una tabella;
- Ad ogni migrazione, è necessario pensare bene a tutti gli effetti collaterali che potrebbero avvenire sugli statement che utilizzano la tabella modificata.

[1] La spiegazione dettagliata del pattern Active Record di Martin Fowler è descritta all'indirizzo `http://www.martinfowler.com/eaaCatalog/activeRecord.html`.

4.3.2 ActiveAdmin

Figura 12: Logo di ActiveAdmin

ActiveAdmin è una libreria (Gemma) di Ruby on Rails che permette di creare un sistema di amministrazione (backoffice). Mediante un pannello di gestione, permette di inserire, eliminare ed aggiornare istanze di modelli e relazioni direttamente da browser.

Vantaggi

- Questa libreria permette ad un utente del sito di essere autonomo nell'aggiornamento dei contenuti senza dover attendere i tempi tecnici dell'azienda fornitrice;
- È fortemente personalizzabile, per questo è spesso utilizzato per la creazione di Dashboard o altre funzionalità dedicate ad utenti ai quali sono assegnati particolari privilegi d'accesso.

Svantaggi

- La grafica di ActiveAdmin risulta datata, è quindi spesso necessario riprogettare la veste grafica del pannello di amministrazione fornito di default.

4.3.3 I18n

Figura 13: Logo di I18n

I18n è l'abbreviazione della parola "internationalization", deriva dal suo spelling che interpone 18 lettere tra la "i" iniziale e la "n" finale.
Lo scopo è gestire la distribuzione del software in diverse lingue. Il meccanismo prevede un codice identificativo per ogni stringa da visualizzare, che assumerà un diverso valore per ogni traduzione.
In Ruby, esiste una apposita gemma, denominata appunto *I18N*, che fornisce le direttive per adottare correttamente questo approccio.
Spesso il concetto di internazionalizzazione, viene associato a quello di localizzazione che è l'insieme dei processi di adattamento di un software , pensato e progettato per un mercato e contesto predefinito, in modo specifico ad altre nazione e culture.
Tutte le informazioni di pertinenza di una lingua sono raccolti in un gruppo di parametri chiamato *locale*.

Vantaggi

- Grazie alla gemma *i18n* , è possibile riutilizzare il codice responsabile della logica di business dell'applicazione e localizzare le stringhe per ogni lingua;

- L'aggiunta di nuove lingue richiede solo lo sforzo riguardante la traduzione ed eventuali conversioni nelle unità di misura appropriate;
- È possibile riutilizzare in contesti diversi le stringhe già definite, evitando quindi di introdurre errori di battitura;
- L'aggiunta di nuove lingue non provoca alcuna modifica nell'applicazione esistente, ma solo l'aggiunta del file in formato `yml` con le traduzioni.

Svantaggi

- La scrittura di codice con questo approccio richiede generalmente più tempo se il software esiste in una sola lingua.

4.4 JRuby

Figura 14: Logo di JRuby

JRuby è un'implementazione in Java del linguaggio di programmazione Ruby.
Sebbene non siano coperte tutte le implementazioni delle librerie standard offerte da Ruby, è comunque possibile utilizzare tutti la maggior parte delle funzionalità del linguaggio.

Vantaggi

- JRuby è un software completamente gratuito;
- Gira su una JVM, quindi è possibile integrare l'interprete Ruby in una applicazione Java ed, allo stesso tempo, scrivere direttamente codice Java.

Svantaggi

- È possibile incontrare situazioni in cui si necessita di una libreria che non è stata implementata;
- L'utilizzo di JRuby provoca un significativo appesantimento del software ed un conseguente peggioramento delle performance.

4.5 Drools

Figura 15: Logo di Drools

Drools è un Business Rules Management System (BRMS) basato su forward and backward chaining inference (vedi Appendice B).
Questo strumento permette di definire delle regole che, al soddisfacimento delle condizioni

iniziali, permettono a Drools di prendere delle decisioni e di conseguenza determinare in che modo il software deve agire.

4.5.1 Architettura

Ad alto livello, Drools si può vedere come tre componenti che cooperano (vedi Figura 16): *Production Memory*, *Working Memory* ed *Inference Engine*. Quest'ultimo si compone di due sotto componenti: *Pattern matcher* ed *Agenda*.

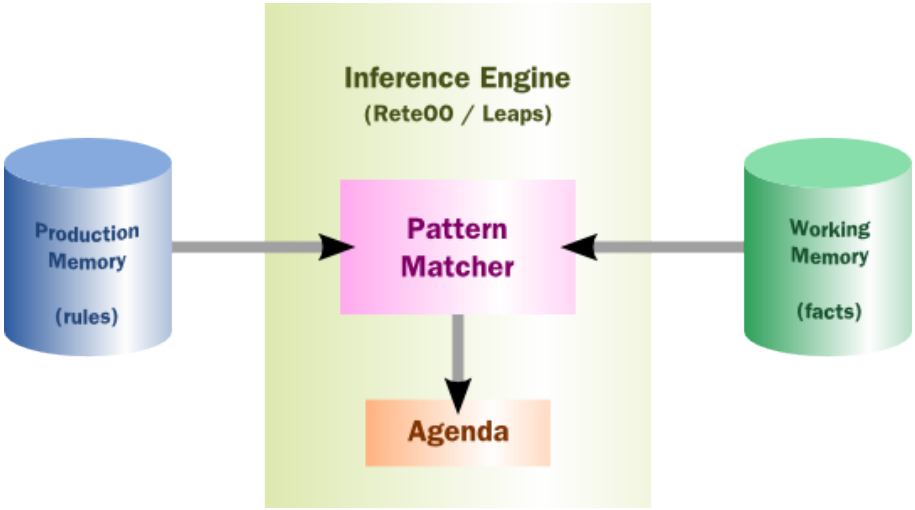


Figura 16: Architettura di Drools

- **Production Memory:**
  Componente che contiene tutte le regole che compongono la Knowledge Base (KB);
- **Working Memory:**
  Componente che contiene tutti i fatti. In Drools, i fatti sono degli oggetti Java.
  Si chiama *Working Memory*, poiché è possibile effettuare operazioni di modifica, inserimento e rimozione dei fatti che contiene;
- **Inference engine:**
  Macrocomponente che ha la responsabilità di valutare i fatti della *Working Memory* sulla base della Knowledge Base presente in *Production Memory*. Per fare ciò si avvale di due sottocomponenti:
  - **Pattern matcher:**
    Componente che ha lo scopo di verificare, quando richiamata dall'inference engine, quali regole soddisfano la condizione presente nella LHS_G sulla base dei fatti presenti nella *Working Memory* al momento dell'invocazione. Verranno poi eseguite le azioni presenti nella RHS_G di tali regole. Per fare ciò in modo efficiente con molti dati, viene utilizzato l'algoritmo Rete, brevemente descritto nell'Appendice D;
  - **Agenda:**
    Ha lo scopo di risolvere i conflitti tra le regole. Essi avvengono nel momento in cui più regole soddisfano la condizione nella loro LHS_G con gli stessi fatti.
    In questi casi, interviene l'Agenda che decide un ordine di esecuzione tra le regole che vanno in conflitto.

Vantaggi

- In sistemi con grandi quantità di dati e vincoli, un approccio di questo tipo facilita notevolmente il mantenimento della verità ed il controllo sui dati inseriti. La definizione dei vincoli e l'aggiornamento degli stessi, risulta molto meno oneroso e di più facile verifica;
- L'elaborazione, risulta essere più veloce e performante rispetto ad un approccio tradizionale.

Svantaggi

- Per poter integrare questo sistema con Ruby on Rails é necessario utilizzare jRuby ed implementare in Java, le classi corrispondenti ai modelli sui quali si vogliono definire delle regole. Tecnicamente, qusto si fa con dei file di template java.erb di Rails che ottengono le informazioni riguardo classi e membri mediante Reflection$_G$;
- Per lavorare in modo efficace con questa tecnologia, è necessario investire molto tempo per conoscerne a fondo tutte le sfaccettature.

4.6 Foundation

Figura 17: Logo di Foundation

Foundation è una collezione di framework$_G$ per lo sviluppo della componente front-end$_G$ di applicazioni web.
In particolare è composto da tre pacchetti:

- **Foundation for Sites:** per la realizzazione di siti web;
- **Foundation for Emails:** per la realizzazione di email formattate in html;
- **Foundation for Apps:** per la realizzazione di applicazioni web.

La struttura di ogni framework$_G$ è modulare e consiste essenzialmente di fogli di stile in formato SASS$_G$ - SCSS$_G$.
Per limitare il peso dei fogli di stile, è possibile selezionare le sole componenti di cui si necessita al momento del download: saranno le sole presenti nel pacchetto da includere nel sito o da scaricare con un CDN$_G$.
Come la maggioranza dei framework$_G$ moderni per il front-end$_G$ Foundation fornisce un sistema di posizionamento a griglia dei componenti. Sono previste dodici colonne, visibili una accanto all'altra quando la larghezza dello schermo è maggiore di 940 pixel. Il numero di colonne viene adattato automaticamente a seconda dello spazio a disposizione, rendendo il sito, l'applicazione o l'email responsive.

Vantaggi

- Oltre ai comuni elementi HTML, sono messi a disposizione componenti aggiuntivi per la realizzazione delle sezioni di interfaccia maggiormente usate, ad esempio:
  - Gruppi di bottoni;
  - Menu a tendina;
  - Breadcrumb$_G$;
  - Messaggi formattati per avvisi, errori, aiuto.
- Foundation permette di realizzare interfacce grafiche accattivanti senza dedicare eccessivo tempo alla loro implementazione;
- La collezione di framework$_G$ che compone Foundation è stata testata sui principali browser$_G$ ottenendo ottimi risultati (vedi Figura 18 per il dettaglio sulla versione utilizzata, la 5.5).

Browser/OS	The Grid	Layout/UI	JS
Chrome	✓	✓	✓
Firefox	✓	✓	✓
Safari	✓	✓	✓
IE10	✓	✓	✓
IE11	✓	✓	✓
IE9	✓	✓	✓
IE8	✗	✗	✗
IE7	✗	✗	✗
iOS (iPhone)	✓	✓	✓
iOS (iPad)	✓	✓	✓
Android 2, 4 (Phone)	✓	✓	✓
Android 2, 4 (Tablet)	✓	✓	✓
Windows Phone 7+	✓	✓	✓
Surface	✓	✓	✓

Figura 18: Compatibilità di Foundation 5.5

Svantaggi

- Per utilizzare questo framework$_G$ è necessario investire del tempo per apprenderne appieno le caratteristiche.

4.7 jQuery

Figura 19: Logo di jQuery

jQuery è una libreria Javascript$_G$ per applicazioni web. Nasce con l'obiettivo di velocizzare la gestione del Document Object Model (DOM)$_G$ e semplificare l'utilizzo delle chiamate Ajax$_G$ e degli eventi.

Vantaggi

- Permette di eseguire chiamate Ajax$_G$ in modo veloce, rendendo così l'applicazione web più dinamica;
- Velocizza l'accesso e la gestione degli elementi del DOM$_G$ delle pagine HTML$_G$;
- È stata testata sui principali browser$_G$;
- Permette di semplificare e ridurre il codice, velocizzando il raggiungimento degli obiettivi.

Svantaggi

- È sempre necessario importare tutto il pacchetto. Tuttavia, la versione compressa, occupa poco meno di 100 KiloBite.

5 Analisi del prodotto esistente

Al momento dell'arrivo in azienda era già presente una versione alpha del software.
Le aziende che il prodotto si prefigge di supportare in questa fase, sono quelle con i codici ATECO$_G$ in ambito edilizio. In particolare, al mio arrivo erano gestite le informazioni relative a:

- Sedi;
- Cantieri;
- Dipendenti;
- Organigramma aziendale;
- Abitabilità;
- Certificato di prevenzione degli incendi;
- DVR$_G$ e documentazione collaterale ad esso.

Il software esponeva una collezione di oltre 400 domande. Sulla base delle risposte a queste domande ed alle informazioni relative alle entità sopra indicate, venivano verificati alcuni vincoli mediante regole del sistema esperto.

5.1 Architettura del software

5.1.1 Architettura ad alto livello

Il software è gestito mediante tecnologia Software as a Service (SaaS)$_G$, un modello di distribuzione del software applicativo dove il fornitore del software si occupa della sua implementazione e manutenzione. Il servizio viene erogato al cliente mediante una applicazione web fruibile via internet.
L'applicazione web risiede fisicamente su una macchina virtuale della piattaforma cloud Amazon Web Services (AWS)$_G$, al fine di evitare oneri e spese di gestione di una infrastruttura informatica dedicata e garantire la scalabilità del servizio.

Come è possibile osservare da Figura 20, ogni istanza di macchina virtuale contiene un clone dell'intera architettura. È stata scelta questa soluzione perché è prevista la possibilità di vendere il pacchetto a più utenti che possono fungere da distributori.
Ogni istanza è composta da quattro componenti principali:

- *Rule Engine;*
- *Database;*
- *Web Application;*
- *Storage.*

La componente *Storage*, in particolare, è stata dedicata al salvataggio di documenti in formato PDF esportabili in ogni momento da ogni azienda. Questa funzionalità non è ancora stata implementata.
Il routing delle richieste viene effettuato con un Reverse proxy$_G$, nello specifico *NGINX*.
In particolare Ruby on Rails permette nativamente di creare e versionare i database a seconda dell'ambiente nel quale si opera (*development, test, staging e production*). Questa caratteristica è tornata molto utile per lo sviluppo, dal momento che è stato possibile fare test accurati senza intaccare i dati in produzione. Inoltre ciò rende possibile evitare di utilizzare la console in sandbox di Rails, la quale presenta criticità nella consistenza dei dati visibili da shell ripetto a quelli visibili da browser$_G$.
La persistenza dei dati è stata gestita mediante il modulo `ActiveRecord` di Ruby on Rails (descritto nella sezione 4.3.1) il quale salva le informazioni su un database *PostgreSQL*.

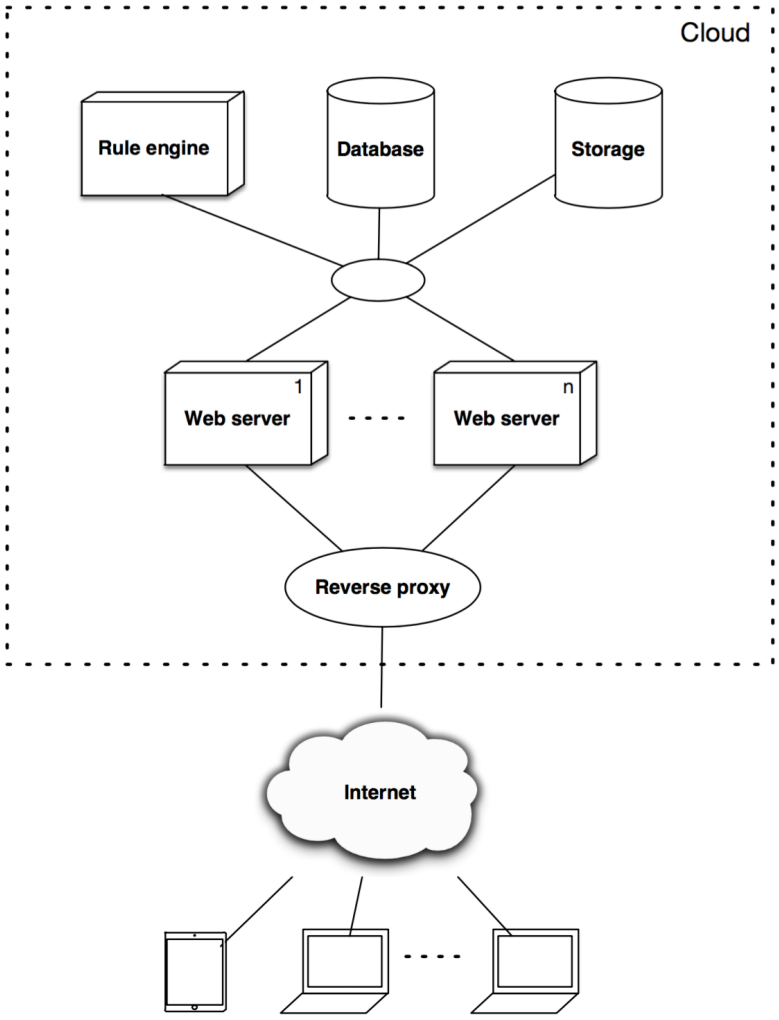


Figura 20: Architettura del sistema

5.1.2 Architettura a basso livello

L'applicazione web è organizzata secondo il pattern Model View Controller (MVC)$_G$. Per il raggiungimento delle viste e l'accesso alle informazioni necessarie al corretto funzionamento dell'applicazione è stata implementata una interfaccia REST$_G$.
Il sistema ha come entità principale il modello `Company` al quale sono riferite, direttamente o indirettamente, tutte le risorse.
Sono poi presenti numerosi modelli relativi alle entità che partecipano alla procedura di Asseverazione$_G$. I modelli più significativi sono:

- `Alert` per rappresentare gli allarmi;
- `Answer` per rappresentare le risposte alle domande;
- `Company` per rappresentare una azienda;
- `ConstructionSite` per rappresentare un cantiere;
- `Dpi` per rappresentare un dispositivo di protezione individuale;
- `Duty` per rappresentare una mansione;
- `FireExtinguisher` per rappresentare un estintore;
- `FirstAidBox` per rappresentare una cassetta di primo soccorso;
- `Individual` per rappresentare una persona;

- `Location` per rappresentare un edificio aziendale, ovvero una sede operativa, una sede amministrativa, una sede legale oppure un magazzino;
- `Machine, LiftingEquipment, ElectricTool` per rappresentare un mezzo oppure uno strumento presente nel parco macchine;
- `MedicalVisit` per rappresentare una visita medica;
- `Procedure` per rappresentare una procedura aziendale, sia essa di prassi o di sistema;
- `Question` per rappresentare una domanda;
- `Training` per rappresentare un corso.

Per ciascun modello, sono stati realizzati opportuni *controller* e *viste*.
Sono stati implementati, inoltre numerosi Concern$_G$ per modularizzare le funzionalità indipendenti dalla singola classe ed allo stesso tempo riutilizzabili da altre classi. Ad esempio, ogni modello che, se istanziato, provoca un aumento del numero delle domande in attesa di risposta, include un apposito Concern$_G$ per l'aggiornamento di un contatore dedicato a tale scopo. Per il calcolo del valore, il Concern$_G$ ricava il nome della classe dell'oggetto corrente mediante Reflection$_G$ ed incrementa automaticamente il valore del numero di domande correlate al tipo individuato.

Relazioni tra domande, risposte ed allarmi

Le risposte sono direttamente collegate alle domande ed ad una entità. Quando un utente risponde ad una domanda, viene aggiornato il recod relativo a quella risposta. A seguito di un inserimento o aggiornamento di una risposta, interviene Drools che valuta se le informazioni inserite rispettano tutti i vincoli previsti, altrimenti viene sollevato un allarme.
Per questioni di efficienza, gli allarmi relativi al vincolo di presenza di una risposta, vengono gestiti direttamente dalle funzionalità di validazione di Rails.
Come per le answer, anche gli allarmi sono sempre collegati ad una risorsa, che di default è l'azienda corrente, ma può assumere come valore una qualsiasi istanza di un modello, purché non sia nulla. Particolarmente degna di nota è l'associazione di una risposta o di un *allarme* alla relativa risorsa. Per fare ciò si è utilizzato l'approccio standard di Rails per il supporto alle associazioni polimorfe.
Come si può vedere da Figura 21, indicando il tipo e l'id della risorsa interessata, l'accesso avviene tramite valutazione della classe e ricerca del relativo *id*.
Si può pensare, ad esempio, all'allarme scatenato alla scadenza di un estintore.
La Risposta (*Answer*) avrà i due campi `answerable_type` ed `answerable_id` impostati rispettivamente a *"FireExtinguisher"* ed al suo *id*. L'allarme avrà un riferimento alla domanda per la quale è stata data una risposta. In questo modo è possibile conoscere il punto esatto dove dirottare l'utente al click sull'allarme per raggiungere il punto di non conformità. Questo aspetto è stato considerato di fondamentale importanza poiché la mole delle informazioni nel software è molto grande, è quindi necessario fornire il maggior numero di strumenti possibili all'utente per facilitarne la navigazione e l'orientamento nel sistema.

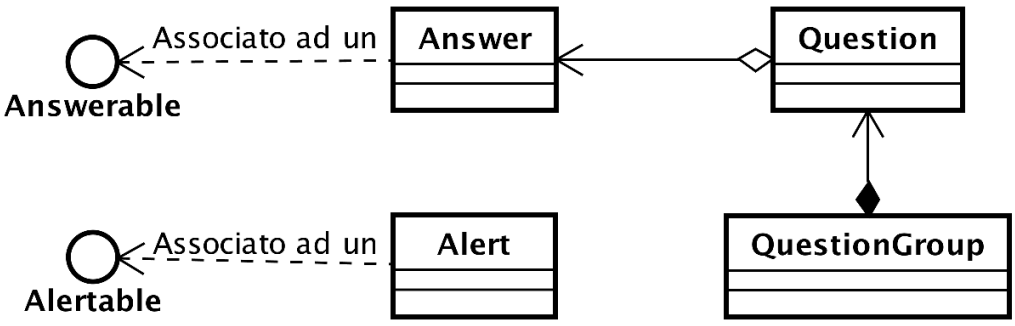


Figura 21: Diagramma delle classi delle associazioni polimorfe di Risposte ed Allarmi.

5.1.3 Integrazione tra Ruby on Rails e Drools

Il codice è stato scritto per la maggior parte utilizzando Ruby on Rails ma il rule engine (Drools) è un framework$_G$ scritto in Java. I due linguaggi non sono nativamente compatibili. Per ovviare a questo problema, è stato utilizzato JRuby, un interprete del linguaggio Ruby scritto in Java, quindi in esecuzione su una Java Virtual Machine (JVM)$_G$.
Per il corretto funzionamento di Drools, è necessario inserire le informazioni nella *Working Memory* come *"fatti"* che sono richiesti come oggetti di tipo JavaBean$_G$ o POJO$_G$.
Per far cooperare i due ambienti, è stato implementato un apposito Concern$_G$ chiamato `"act_as_fact"`. Questo modulo viene incluso nelle classi delle quali è necessario tenere traccia nella *Working Memory* e vengono generati, mediante Reflection utilizzando i template di Rails (ERB$_G$), le classi Java corrispondenti.

5.1.4 Flusso dei dati ed interazione

Sezione nuovissima Un aspetto che merita di essere esaminato è il flusso con il quale vengono generate e valutate domande e risposte (Figura 22).

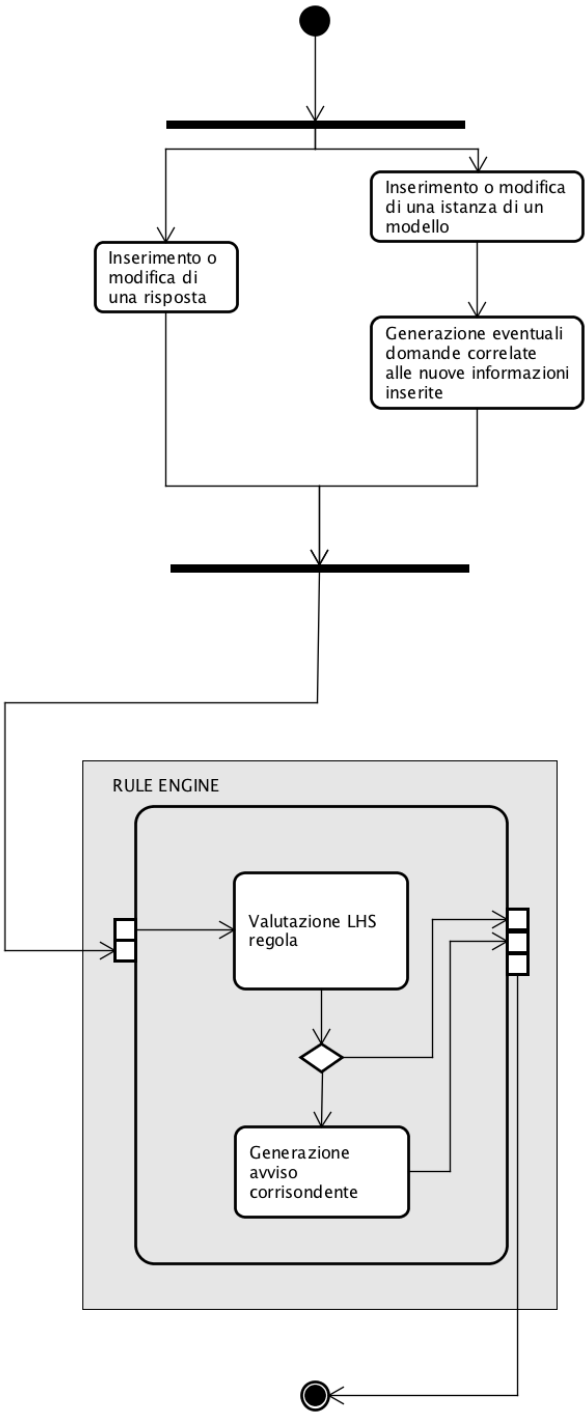


Figura 22: Diagramma di attività del flusso di una risposta o l'inserimento di un oggetto a modello

Nel momento in cui un utente inserisce o modifica un'istanza di un modello oppure risponde ad una domanda, viene aggiornata la *Working Memory*.
Nel caso un cui venga generata o aggiornata un'istanza di modello, può essere necessario l'inserimento di alcune domande ad essa direttamente correlate.
Un esempio è rappresentato dall'aggiunta di un estintore ad un cantiere. Ad ogni estintore in ogni cantiere deve essere associata una posizione specifica che deve essere riportata nel layout di cantiere per permetterne il facile reperimento in caso di incendio. La posizione nel layout di un estintore è rappresentata dal software come una risposta ad una domanda in un apposito

questionario generato ad ogni associazione di un estintore ad un cantiere. Se tale informazione non è specificata viene sollevato un allarme.
Sia per l'inserimento o aggiornamento di una risposta, sia per la generazione o modifica di una istanza di modello, il passo successivo è rappresentato dalla valutazione delle informazioni inserite sulla base della *Knowledge Base*.
È in questo momento che agisce il rule engine Drools che per ogni regola presente nella *Knowledge Base* valuta la LHS_G sulle nuove informazioni inserite e, se soddisfatta, solleva gli allarmi corrispondenti. Un esempio di regola è il seguente:

```
rule "Individuo ricopre una mansione per la quale non è formato"
when
$t: Training()
$d: Duty(trainings contains $t)
$i: Individual(duties contains $d, trainings not contains $t)
then
System.out.println($i.getFirstName() + ' ' + $i.getLastName() + "ha la mansione" +
$d.getName() + " ma non la formazione " + $t.getName() );
end
```

- $t contiene tutte le formazioni;
- $d contiene tutte le mansioni che necessitano della formazione $t;
- $i contiene tutti gli individui che svolgono la mansione $d ma non sono in possesso della formazione $t.

Le corrispondenze di questa regola sono tutti gli individui individui che svolgono una qualunque mansione per la quale non sono correttamente formati.
Al verificarsi di queste condizioni, viene generato un allarme il cui contenuto è specificato dalla stampa disposta dalla RHS_G.

6 Definizione dei casi d'uso

6.1 Legenda

6.2 UC 1

6.3 UC 1.1

6.4 UC 1.2

6.5 UC 1.2.1

6.6 UC 2

6.7 UC ...

7 Sviluppo

7.1 Refactor delle componenti esistenti

7.1.1 Refactor della componente: *Figure di sistema*

7.1.2 Refactor della componente: *Dispositivi di protezione individuale*

7.1.3 Refactor della componente: *Mansioni e formazioni correlate*

7.2 Refactor della componente: *Questionari*

7.3 Nuove componenti

7.3.1 *Segnalazioni*

Requisiti

Progettazione

A Migrazioni

Una migrazione è un file nel quale viene indicato cosa cambia nella struttura del database rispetto allo stato precedente ad essa e le variazioni ai dati ad essa conseguente. Questo permette a tutti gli sviluppatori di avere lo stato della struttura del database sempre aggiornato. Qui di seguito un esempio di migrazione che crea una tabella Bookmark (nel metodo statico up), e specifica cosa fare in caso di annullamento della migrazione (nel metodo statico down).

```
class CreateBookmarks < ActiveRecord::Migration
  def self.up
    create_table :bookmarks do |t|
      t.string :url
      t.string :title
      t.text :description

      t.timestamps
    end
  end

  def self.down
    drop_table :bookmarks
  end
end
```

Una volta eseguita la migrazione verrà creata una tabella Bookmarks con i tre campi *url*, *title* e *description*.

B Inferenza e Motori di inferenza

Con il termine inferenza si intende il processo con il quale si passa da una proposizione vera ad una seconda proposizione la cui verità è derivata dal contenuto della prima.
Questo processo avviene mediante l'applicazione di regole, dette regole di inferenza. Al verificarsi di una o più condizioni (*antecedent*), queste regole traggono delle conclusioni ed agiscono di conseguenza sul sistema, sulla base di ciò che è definito nella seconda parte della regola (*consequent*).
Prendiamo ad esempio la regola `"Se piove apro l'ombrello"`.
L'*antecedent* è rappresentato dalla condizione: `"Oggi piove?"`. Se la condizione dell'*antecedent* è soddisfatta, verrà tratta la conclusione descritta nel *consequent*, in questo caso `"apro l'ombrello"`.

Questo genere di approccio assume maggior significato se applicato contemporaneamente ad un insieme di fatti o circostanze.
Un motore inferenziale (*inference engine*), è un software il cui algoritmo simula le modalità con cui la mente umana trae delle conclusioni logiche attraverso il ragionamento utilizzando le regole di inferenza. Le due modalità con cui si opera in questi contesti sono principalmente le seguenti:

- **Backward chaining**
  Rappresenta il ragionamento di tipo induttivo, il quale dall'analisi di informazioni di carattere particolare permette di ricavare informazioni di carattere generale.
  Nella pratica, si considerano una serie di obiettivi e si cerca tra i dati disponibili se ce ne sono di disponibili tali da supportare tutti gli obiettivi fissati.
  Un motore inferenziale utilizza questo approccio cercando tra le regole di inferenza finché non ne individua una che abbia il *consequent* che soddisfa l'obiettivo. Se non è provata la verità dell'*antecedent* della regola individuata, allora l'*antecedent* stesso diventa un nuovo obiettivo poiché, una volta soddisfatto, sarà soddisfatto anche l'obiettivo precedentemente individuato.
- **Forward chaining**
  Rappresenta il ragionamento di tipo deduttivo, ovvero permette di partire da principi di carattere generale per estrarne uno o più di carattere particolare.
  L'approccio utilizzato è quello di iniziare con i dati già disponibili ed usare le regole di inferenza per estrarne di nuovi fino a quando non si è raggiunto l'obiettivo fissato.
  Un inference engine che usa il forward chaining, itera sulle regole di inferenza eseguendo quelle che hanno disposizione tutte le informazioni per poter calcolare i nuovi dati e fermandosi quando è stato raggiunto l'obiettivo fissato.
  A differenza del backward chaining, questo è un approccio orientato ai dati ed effettua i *"ragionamenti"* al fine di ottenere delle risposte.
  È consigliabile rispetto al backward chaining nelle situazioni in cui i dati cambiano frequentemente, perché in seguito alla modifica, alla cancellazione ed all'immissione di nuovi dati, viene innescato il processo di inferenza dilazionando il costo computazionale nel tempo.

C Sistemi Esperti

Un sistema esperto è un programma che cerca di riprodurre le capacità di decisione di una o più persone esperte in un determinato campo di attività.
Per il corretto funzionamento, è necessario che siano fornite procedure di inferenza sufficienti alla risoluzione dei problemi alla quale si vuole fornire risposta.
Un sistema esperto è sempre in grado di esibire i passaggi logici che hanno portato ad una particolare decisione.
Le principali componenti sono:

- **Base di conoscenza (Knowledge Base)**
  Componente che contiene tutte le regole necessarie al sistema per prendere le decisioni; non contiene i dati;
- **Motore inferenziale**
  (vedi Appendice B);
- **Interfaccia Utente**
  Componente che permette l'interazione fra il soggetto umano ed il software che deve dare risposta ad un suo particolare problema.

I sistemi esperti si dividono in due categorie principali: quelli basati su regole e quelli basati su alberi. Per la realizzazione del progetto di stage è stato utilizzato *Drools*, un framework per la realizzazione di sistemi esperti basati su regole. Non verranno quindi trattati su questa tesi i sistemi basati su alberi.
I sistemi esperti basati su regole sono dei programmi composti da regole della forma `IF condizioni THEN azione`.
La parte condizionale viene detta Left Hand Side (LHS_G), mentre quella relativa all'azione viene detta Right Hand Side (RHS_G).
Le regole vengono valutate ad ogni variazione sui dati che avviene sempre mediante l'introduzione di nuovi fatti.
Esempio:
Fatti :

- Mal di Testa
- Raffreddore
- Temperatura >=38

```
IF( (Mal di testa) AND (Raffreddore) AND (Temperatura >=38))
THEN ( Diagnosi: Influenza)
```

D Algoritmo RETE

L'algoritmo RETE è stato inventato da Charles Forgy nel 1978 e successivamente perfezionato nel 1979.
Può essere separato in due parti:

- Compilazione delle regole;
- Esecuzione runtime.

Questo algoritmo prevede l'utilizzo di una sorta di rete che filtra i dati mano a mano che essi la attraversano. I nodi all'inizio della rete avranno, con probabilità molto alta, un numero elevato di match. Mano a mano che si scende in profondità, le corrispondenze tenderanno sempre più a diminuire di numero. In fondo alla rete, troveremo i nodi terminali.
Il funzionamento prevede la creazione di un nodo radice, attraverso il quale tutti gli oggetti entrano nella rete. Ogni oggetto, dopo essere passato per il nodo radice, viene subito indirizzato verso un altro nodo chiamato *ObjectTypeNode* che ha lo scopo di assicurare che ogni nodo venga inoltrato verso nodi che sanno gestire oggetti con un tipo compatibile a quello dell'oggetto in analisi[2].
Gli *ObjectTypeNode*, propagano gli oggetti verso dei nodi detti *AlphaNode*, ognuno dei quali verifica una condizione.
Tramite altre due tipologie di nodi (*JoinNode* e *NotNode*), si verificano condizioni che coinvolgono oggetti di diverso tipo, per poi giungere infine ai nodi terminali.
Una volta raggiunto un nodo terminale, si ha la garanzia che la regola è stata soddisfatta e viene eseguita l'azione descritta nella RHS_G corrispondente.

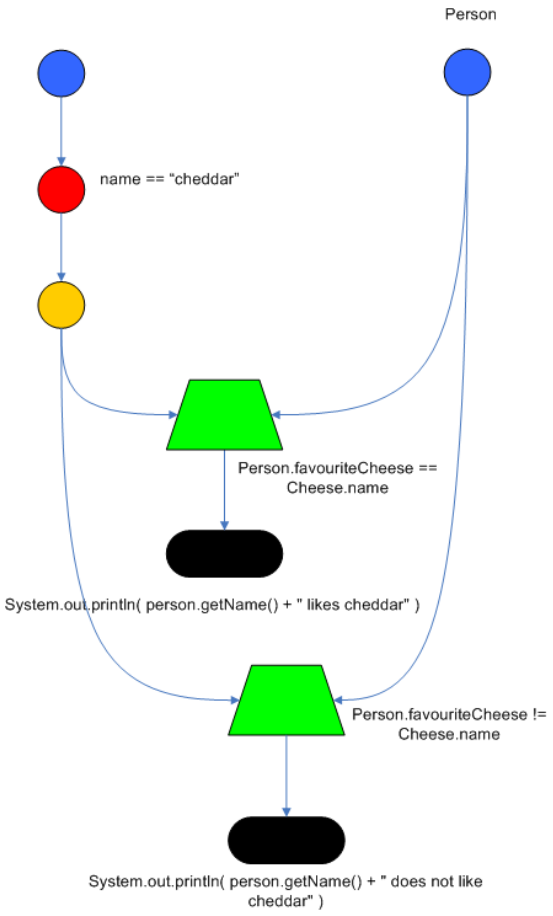


Figura 23: Esempio di flusso di valutazione di una regola Drools.

[2]Tecnicamente questo passaggio avviene mediante l'invocazione dell'operatore `instanceof` di Java.

Un esempio di flusso di funzionamento è raffigurato nella Figura 23.
In particolare i nodi azzurri sono degli *ObjectTypeNode* e rappresentano nell'ordine formaggi e persone.
Il nodo rosso, invece, è di tipo *AlphaNode* e verifica la condizione che il nome del formaggio sia "cheddar".
Il nodo giallo è di tipo *LeftInputAdapterNode* e serve per scegliere dove dirottare il flusso a seguito del soddisfacimento o meno di una condizione prevista da un *AlphaNode*.
Se la condizione prevista dall'*AlphaNode* è verificata, il flusso viene inoltrato al primo punto di join (identificato da un trapezio verde) che verifica se il formaggio preferito dalla persona è il *cheddar*. si dice punto di join perché per operare ha bisogno delle informazioni provenienti da due diverse entità gestite da degli *ObjectTypeNode* distinti.
se la condizione prevista dall'*AlphaNode* non è verificata, si passa al secondo punto di join.
In caso di verifica delle condizioni dei *JoinNode*, si giunge ai nodi terminali ovvero al soddisfacimento delle condizioni che permette di prendere una decisione. In questo caso viene solo stampata una stringa, ma in generale è possibile scegliere l'azione più adatta al contesto di utilizzo.

TODO Sistemare le etichette dei nodi

Glossario

Ajax Tecnica di sviluppo software per la realizzazione di applicazioni web interattive. Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente.

Asseverazione Con asseverazione si intende una scelta volontaria di una impresa edile al fine di dimostrare l'impegno per la prevenzione, salute e sicurezza nei luoghi di lavoro. Opera mediante la certificazione di conformità dei modelli di organizzazione e gestione aziendali e mediante visite a campione nei cantieri. L'asseverazione offre benefici economici alle aziende che la richiedono e garantisce efficacia esimente rispetto alla responsabilità amministrativa delle imprese.

ATECO Sigla della classificazione ISTAT(Istituto Nazionale di Statistica) per determinare la tipologia delle attività economiche (**AT**tività **ECO**nomiche).

AWS Amazon Web Services

Breadcrumb Letteralmente si traduce in *briciole di pane*. Rappresenta il percorso dalla home page, o comunque da una radice, al punto corrente della navigazione.
Il nome deriva dalla fiaba di Hansel e Gretel, in cui Hansel sparge lungo il suo cammino delle briciole di pane per poter riconoscere la strada precedentemente percorsa e poter tornare sui suoi passi.

browser Un browser è un programma che consente di visualizzare i contenuti delle pagine web e di interagire con esse.

CDN Sta per *Content Delivery Network*, ovvero rete di consegna di contenuti. Questo meccanismo viene utilizzato per reperire il codice relativo agli strumenti di terze parti direttamente dalla rete, senza tenere una versione del codice stesso nel proprio server web.

Concern Un concern rappresenta un modulo di Rails utilizzato per facilitare la gestione di funzionalità assegnabili a più classi ma allo stesso tempo gestibili in modo atomico.

DOM Document Object Model

DVR Documento di Valutazione dei Rischi

ERB Sta per Embedded RuBy.
È il sistema per la gestione dei template in Rails. ERB permette di generare file in moltissimi formati generando a tutti gli effetti dei template. In particolare, le informazioni vengono reperite mediante Ruby On Rails, ma la struttura del file nel formato specificato rimane invariata, adattando a tutti gli effetti le informazioni presenti al mommento della invocazione ad una struttura definita in precedenza.

framework Architettura software sulla quale un software può essere progettato e realizzato.

front-end Parte di un sistema software che gestisce l'interazione con l'utente o con sistemi esterni che producono dati di ingresso.

HTML HyperText Markup Language. Linguaggio di markup per la strutturazione delle pagine web.

JavaBean Letteralmente si traduce in "Chicchi di Java". Le JavaBean sono classi scritte in Java seguendo le seguenti convenzioni:

- Un costruttore senza parametri;
- Devono essere definiti tutti i metodi accessori per le variabili manipolabili (in particolare getter e setter);
- La classe dovrebbe essere serializzabile;
- La classe non dovrebbe contenere metodi per la gestione degli eventi.

Le classi JavaBean, sono spesso utilizzate per incapsulare più oggetti in un singolo oggetto in modo da passare un solo oggetto ai metodi che necessitano di tutti qusti oggetti come parametri.

Javascript TODO

JVM Java Virtual Machine

LHS Sta per Left Hand Side. Rappresenta l'insieme delle condizioni che verificano una regola del rule engine Drools.

MVC Model View Controller

POJO Sta per `Plain Old Java Object`. Rappresenta un oggetto Java non legato ad alcuna restrizione diversa da quelle costrette dalla specifica del linguaggio Java.

Reflection Capacità di un programma di eseguire elaborazioni che hanno per oggetto il programma stesso, ed in particolare la struttura del suo codice sorgente. Un utilizzo comune, è l'utilizzo di questo approccio per individuare il nome della classe oppure degli attributi a partire da una istanza di un oggetto.

REST Tipo di architettura software per i sistemi di ipertesto distribuiti come il World Wide Web, si riferisce ad un insieme di principi di architetture di rete, i quali delineano come le risorse sono definite e indirizzate.

Reverse proxy Si tratta di un tipo particolare di proxy che recupera le informazioni per conto di un client da uno o più server. Le risorse ottenute vengono restituite al client come se provenissero direttamente dal Proxy server, rendendo trasparente il recupero distribuito delle informazioni all'utente. Questo tipo di approccio risulta molto utile nei contesti in cui il recupero delle informazioni risulta oneroso perché, permettendo la distribuzione su più macchine, si ottiene un bilanciamento del carico.

RHS Sta per Right Hand Side. Rappresenta l'insieme delle azioni da intraprendere se tutte le condizioni presenti nella LHS del rule engine Drools sono soddisfatte.

SaaS Software as a Service

SASS È un preprocessore CSS che analizza il codice scritto in SCSS e ne elabora il risultato. Si tratta a tutti gli effetti di una estensione di CSS3 che aggiunge:

- Regole annidate;
- Variabili;
- Mixin;
- Ereditarietà tra i selettori.

Home page del progetto: http://sass-lang.com/.

SCSS Sta per Sassy CSS ed è la sintassi utilizzata da SASS.