UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
TESI DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

PROGETTAZIONE E SVILUPPO DI UN'APPLICAZIONE WEB PER LA GESTIONE DEI PROTOTIPI E DELLE PRESERIE

RELATORE: Prof. Carlo Ferrari

CORRELATORE: Ing. Abramo Pavan

LAUREANDO: *Nicola Celli*

ANNO ACCADEMICO 2012/2013

Ai miei genitori e a Deborah.

Glossario

API	*Application Programming Interface* è un'interfaccia implementata da un programma per rendere possibile l'interazione con altri software.
CRUD	*Create, Read, Update and Delete* sono le quattro operazioni basilari che possono essere effettuate su una base di dati: creare, leggere, aggiornare ed eliminare.
DBMS	*Database Management System* è un sistema software progettato per consentire la creazione, la manipolazione e l'interrogazione efficiente di basi di dati.
EIS	*Executive Information System* sistemi che supportano l'alta direzione nel processo decisionale fornendo informazioni in tempo reale attraverso un'interfaccia intuitiva.
EJB	*Enterprise JavaBeans* sono i componenti che implementano la logica di business di un'applicazione.
JAVA EE	*Java Platform Enterprise Edition* fornisce un'ampia serie di specifiche e interfacce (API) che definiscono le componenti tecnologiche in cui è strutturata e la modalità di interazione tra di esse.
JMS	*Java Message Service* è l'insieme di API, appartenente a Java EE, che consente ad applicazioni Java presenti in una rete di scambiarsi messaggi tra loro.
JNDI	*Java Naming and Directory Interface* è una API Java per servizi di *directory*.
JPA	*Java Persistence API* forniscono uno standard per la persistenza dei dati in Java.

JPQL	*Jave Persistence Query Language* linguaggio simile a SQL adoperato per realizzare interrogazioni o aggiornamenti delle entità del database.
JSF	*JavaServer Faces* è un *framework* per la realizzazione di interfacce utente di applicazioni web.
JSR	*Java Specification Requests* sono l'insieme di specifiche, già approvate dalla *Java Community Process* (JCP) o in fase di approvazione, che vengono numerate e rese pubbliche. La JCP è quell'istituzione che si occupa di regolare lo sviluppo della tecnologia Java.
JVM	*Java Virtual Machine* è la componente della piattaforma Java che esegue i programmi tradotti in *bytecode* dopo una prima compilazione.
MDB	*Message Driven* sono una tipologia di EJB, con comportamento asincrono grazie all'utilizzo delle specifiche JMS.
MVC	*Model-View-Controller* è un *design pattern* che si basa sull'idea di separare i dati dalla loro rappresentazione.
ODL	*Ordine Di Lavoro* è una richiesta di produzione.
ORM	*Object-Relational Mapping* denota essenzialmente il processo di mappatura dei dati fra gli oggetti e le tabelle di una base di dati.
PLCM	*Product Life-Cycle Management* terminologia utilizzata in *Carel* per identificare un nuovo prodotto ottenuto dalla modifica di uno già in vendita, chiamato "prodotto padre".
SMD	*Surface Mounting Device* sono componenti utilizzati in elettronica per l'assemblaggio di un circuito stampato senza la necessità di praticare dei fori.
UI	*User Interface* è ciò che consente l'interazione fra l'utente e la macchina
VSL	*Value Stream Leader* è personale interno *Carel* responsabile di una linea produttiva o responsabile di un sito produttivo estero.
XML	*eXtensible Markup Language* è un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

Indice

Introduzione

L'ottimizzazione dei processi aziendali è un ambito di intervento in cui un'adeguata informatizzazione consente di migliorare praticamente tutte le componenti del processo (fasi, contributo dei singoli attori, strumenti, informazioni, comunicazioni, risultati).
Il progetto ha lo scopo di creare un'applicazione web per la gestione dei prototipi in *Carel*, un'azienda che si occupa di refrigerazione, del condizionamento e dell'umidificazione dell'aria, specializzata nella realizzazione di sistemi di regolazione. Quest'applicazione verrà poi inserita all'interno del portale aziendale, dove ne sono presenti già altre.
Il processo aziendale studiato coinvolge tre funzioni: sviluppo prodotto, logistica e produzione. Tale flusso era gestito attraverso un modulo cartaceo disponibile in appendice A.
L'obiettivo è quindi di riuscire a informatizzare questo processo. Il giusto approccio per ottenerlo si basa su due macro attività:

- l'analisi del processo;
- la corretta progettazione del software per l'informatizzazione.

Il processo è stato analizzato utilizzando la tecnica di mappatura del flusso della metodologia *lean* che ha l'obiettivo di riuscire a ridurre gli sprechi in un certo processo. Con questa tecnica si parte mappando un modello dello stato attuale, che va ristrutturato in maniera più efficiente fino ad ottenere quello dello stato futuro.
L'applicazione web è stata realizzata secondo lo standard *Java Platform Enterprise Edition* (Java EE, cfr. [9]). Questo fornisce un'ampia serie di specifiche e interfacce, *Application Programming Interface* (API), che definiscono le componenti tecnologiche in cui è strutturata e la modalità di interazione tra di esse.

Si è fatto soprattutto uso del framework *JavaServer Faces* (JSF), che è stato necessario per la realizzazione dell'interfaccia utente dell'applicazione web.
I vantaggi riscontrabili con l'informatizzazione di questo processo aziendale sono:

- la possibilità di rendere il flusso delle operazioni più snello;
- poter creare uno storico, non più cartaceo, che faciliti la ricerca;
- rendere più veloci le comunicazioni: non appena un utente ha terminato una certa fase il successivo verrà informato tramite e-mail.

L'ordine dei capitoli rispecchia le due fasi precedentemente elencate. Quindi dopo il capitolo 1, che contiene un'introduzione sulle tecnologie utilizzate per la realizzazione del progetto, nel capitolo 2 viene data una presentazione più approfondita del progetto affrontato per poi passare alla progettazione del flusso aziendale. Verrà anche descritto lo standard "830-1998, IEEE *Recommended Practice for Software Requirements Specification*"[1], utilizzato per redigere il documento di specifica dei requisiti.
I capitoli successivi, invece, riguardano gli aspetti implementativi del software. Nel capitolo 3 sono mostrate tutte le fasi di progettazione e l'implementazione della base di dati a cui si interfaccerà l'applicazione web. Nel capitolo 4 è spiegata la struttura globale dell'applicazione web mentre nel capitolo 5 si sono approfondite le classi che sono state implementate per la sua realizzazione.
In finale, nel capitolo 6 sono riportate le conclusioni ed i possibili sviluppi futuri.

Capitolo 1

Tecnologie

In questo capitolo verrà fornita un'introduzione alle tecnologie dello standard *Java Platform Enterprise Edition* (Java EE, cfr. [9]) che sono alla base dell'applicazione sviluppata. Si inizierà illustrando il modello su cui si basa per poi andare ad approfondire i *layer* di maggior interesse per la comprensione del lavoro svolto.

1.1 Java Platform Enterprise Edition

Java EE rappresenta una base solida per lo sviluppo di un'applicazione. La piattaforma Java EE fornisce un'ampia serie di specifiche e interfacce, *Application Programming Interface* (API), che definiscono le componenti tecnologiche in cui è strutturata e la modalità di interazione tra di esse.

La Java EE è basata su un modello distribuito multistrato. L'applicazione è suddivisa in componenti in base alla loro funzione, i quali lavorano in diverse locazioni a seconda del livello a cui appartengono. Anche se un'applicazione Java EE può consistere in tre o quattro strati, come mostrato in figura 1.1, queste vengono solitamente considerate applicazioni a tre livelli dato che sono distribuite su tre locazioni: il client, il server dell'applicazione Java EE e la base di dati.

Lo strato *client tier* presenta all'utente finale i risultati dell'elaborazione del server. In figura 1.2 sono mostrate le possibili opzioni di implementazione: spesso sono pagine web visualizzate tramite un *browser* web, ma può talvolta essere costituito da un'applicazione client in grado di interagire direttamente con il *business tier*. Un caso particolare di applicazione client è quando si riceve dal *web tier* una pagina web che include una *embedded applet*. Scritta in linguaggio

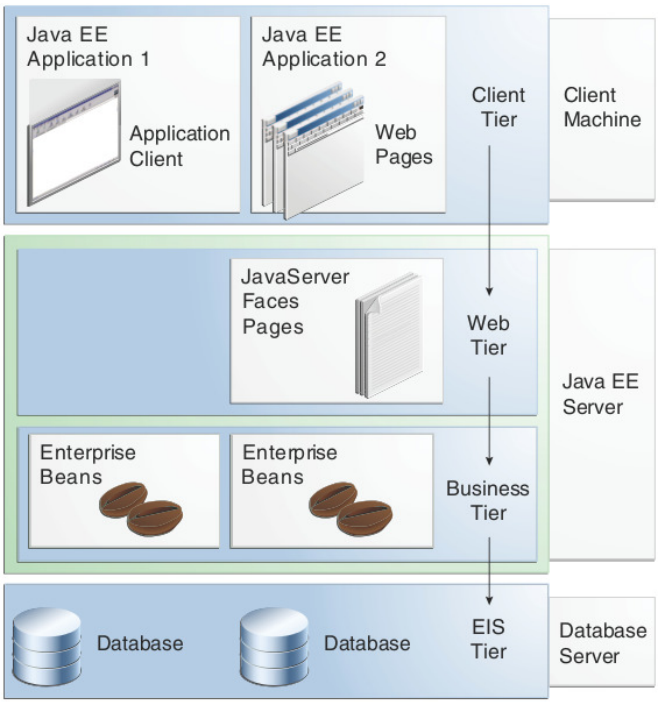


Figura 1.1: Modello multistrato di un'applicazione Java EE [9]

Java, un'*applet* è un piccolo client che viene eseguito nella *Java Virtual Machine* (JVM) del web *browser*.

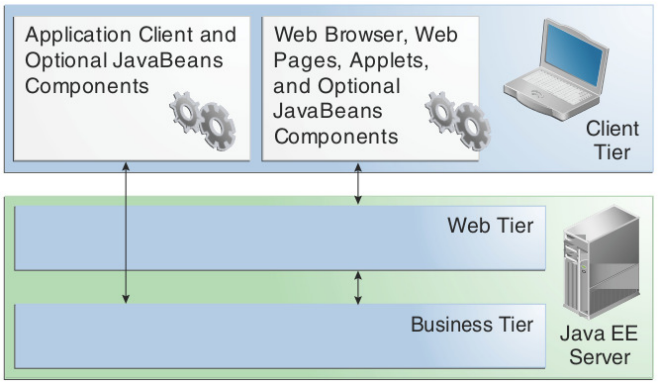


Figura 1.2: Componenti del *client tier* [9]

Il *web tier* e il *client tier* possono contenere componenti basati sull'architettura *JavaBeans* che servono a mantenere e manipolare le informazioni:

- fra un'*application client* o un'*applet* e i componenti nel server Java EE;
- fra i componenti del server Java EE e la base di dati.

Un *JavaBean* è rappresentato da una classe Java con determinati requisiti:

- lo stato dell'istanza deve avere proprietà accessibili solo all'interno della classe stessa, cioè con visibilità privata;
- devono esservi metodi pubblici specifici per poter ottenere e manipolare le informazioni, cioè dei metodi *get* e *set*;
- il costruttore non deve avere argomenti;
- può comunicare i propri cambiamenti di stato ad altre componenti mediante la generazione di eventi;
- l'istanza deve essere serializzabile, cioè deve implementare l'interfaccia *Serializable*[1] .

I componenti del *web tier*, mostrati in figura 1.3, sono le *servlet* e le pagine create utilizzando la tecnologia *JavaServer Faces*. Una *servlet* ha il compito di gestire le richieste dell'utente, effettuare eventuali elaborazioni e fornire una risposta al client. *JavaServer Faces* è un *framework* per la creazione di interfacce utente per applicazioni web che verrà approfondito nella sezione 1.2.

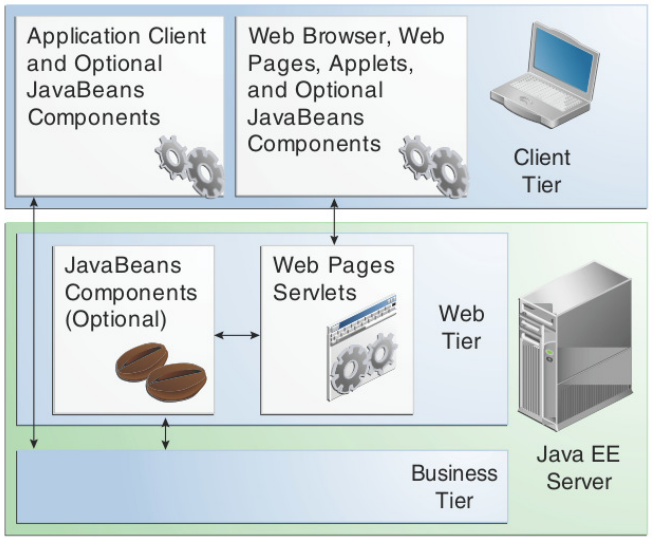


Figura 1.3: Componenti del *web tier* [9]

[1]Informazioni su quest'interfaccia sono reperibili all'indirizzo: `http://docs.oracle.com/javase/6/docs/api/java/io/Serializable.html`

Lo strato del *business tier* è quello in cui viene implementata la logica di business dell'applicazione. Si organizzano i dati e l'accesso ad essi, interagendo ad esempio con un DBMS(*Database Management System*). Nelle applicazioni più semplici viene accorpato al *web tier*. Fa uso della tecnologia *Enterprise Java Beans* che verrà approfondita nella sezione 1.3.

Lo strato *EIS (Enterprise Information System) tier* , mostrato in figura 1.4, è spesso rappresentato da un DBMS o più in generale da un EIS.

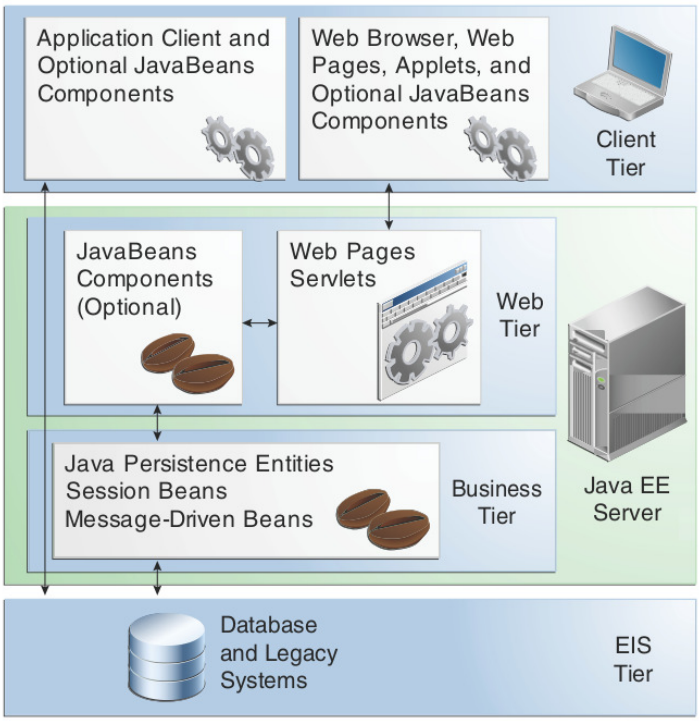


Figura 1.4: Componenti del *business tier* e del *EIS tier* [9]

1.2 JavaServer Faces

JavaServer Faces (JSF) è un *framework* per la realizzazione di interfacce utente di applicazioni web. E' definito all'interno delle *Java Specification Request* (JSR) 127 e attualmente è disponibile nella seconda versione. JSF si basa sul *pattern* MVC (*Model – View – Controller*) mostrato in figura 1.5. Un *pattern* è un modello che permette di definire la "soluzione" di un "problema" specifico che si ripresenta, di volta in volta, in un contesto diverso. Il *design pattern* MVC si basa sull'idea di separare i dati dalla loro rappresentazione, poiché mantenere

un forte accoppiamento tra essi comporta che la modifica dell'uno, implica automaticamente un aggiornamento dell'altro. Questo disaccoppiamento fra dati e rappresentazione viene ottenuto mediante la definizione di tre elementi noti come: *Model*, *View* e *Controller*.

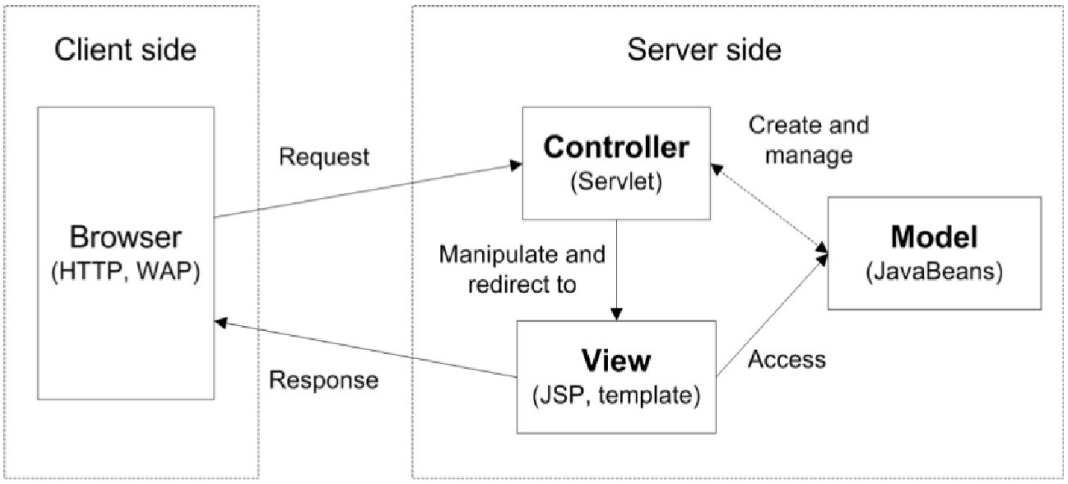


Figura 1.5: MVC *pattern* [10]

Il *Model* (Modello) è responsabile della gestione dei dati e del comportamento dell'applicazione (*data* & *behaviour*). Esso coordina la "*business-logic*" dell'applicazione, l'accesso alle basi di dati e tutte le parti critiche "nascoste" del sistema. E' indipendente dalle specifiche rappresentazioni dei dati sullo schermo e dalle modalità di input dei dati stessi da parte dell'utente.

La *View* (Vista) ha il compito di visualizzare i dati e presentarli all'utente anche in forme diverse, ad esempio in relazione al dispositivo utilizzato per accedere al sistema (es. personal computer, cellulare, ...) . Ciò vuol dire che, pur partendo dagli stessi dati, è possibile effettuare visualizzazioni ("*rendering*") diverse ed ottenere viste multiple dello stesso modello.

Il *Controller* (Controllo) definisce il meccanismo mediante il quale il *Model* e la *View* comunicano. Realizza la connessione logica tra l'interazione dell'utente con l'interfaccia applicativa e i servizi della *business-logic* all'interno del sistema. E' responsabile della scelta di una tra le molteplici viste dello stesso modello: in base, ad esempio, al tipo di dispositivo utilizzato dall'utente per accedere al sistema o in relazione alla localizzazione geografica dell'utente stesso. Una qualsiasi richiesta (*request*), fatta al sistema, viene acquisita dal *Controller* che individua

all'interno del *Model* il gestore della richiesta (*request handler*). Ottenuto il risultato dell'elaborazione (*response*), il *Controller* stesso determina a quale *View* passare i dati per la presentazione degli stessi all'utente.

Il vantaggio principale che scaturisce da questa architettura, è che la *business-logic* definita all'interno del *Model* è separata dall'interfaccia utente che si trova all'interno della *View*. Tutto ciò favorisce il riuso dei componenti e la possibilità di apportare delle modifiche ad un livello senza avere degli effetti sull'altro.

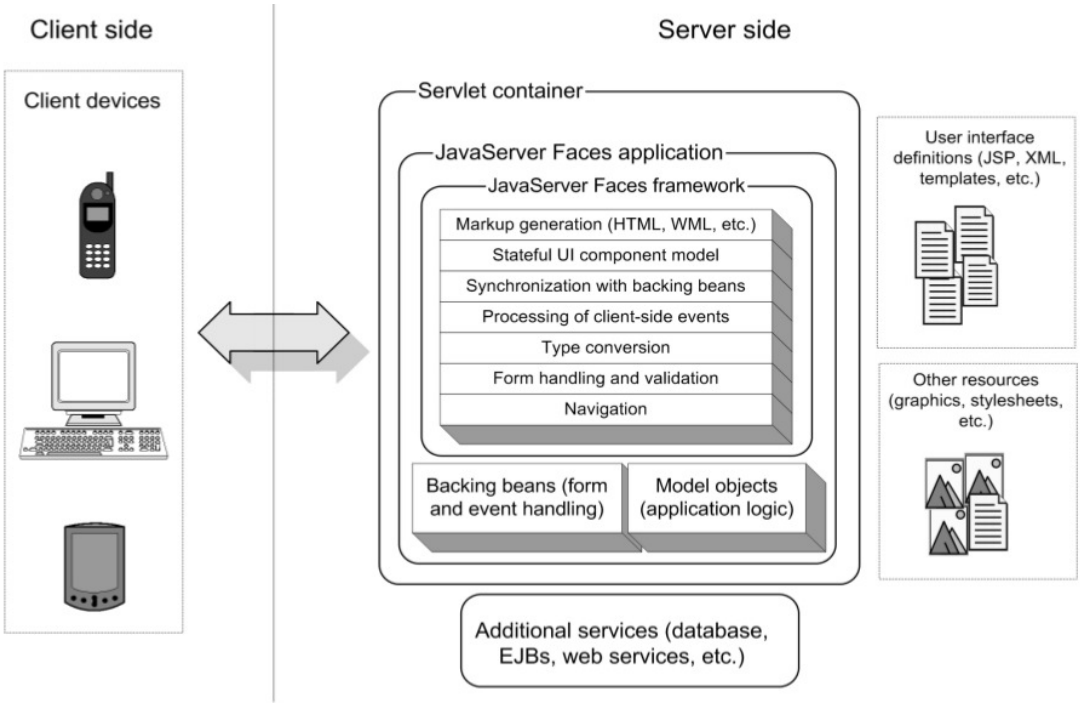


Figura 1.6: Organizzazione vista ad alto livello di un'applicazione JSF[10]

In figura 1.6 è presentata ad alto livello l'organizzazione di un'applicazione JSF, una delle principali caratteristiche è l'ampio insieme delle classi e delle relative API che permettono di:

- definire i componenti dell'interfaccia utente (UI) e gestire il mantenimento del relativo stato;
- gestire gli eventi che si verificano sui suddetti componenti, eseguirne la conversione dei valori e la validazione degli stessi;
- specificare la navigazione fra le pagine all'interno dell'applicazione web;
- supportare l'internazionalizzazione e l'accessibilità;

- realizzare dei componenti personalizzati (*Custom Components*) che estendono e potenziano le funzionalità delle classi base del *framework*;

Nella tabella 1.1 sono illustrati i termini chiave che vengono incontrati nell'implementazione di un'applicazione utilizzando JSF.

Termine	**Descrizione**
Componente UI	Un oggetto *stateful*, mantenuto nel server, che offre specifiche funzionalità per interagire con l'utente. Sono organizzati in una vista che è un albero dei componenti che possono essere visualizzati in una pagina web.
Rendered	Responsabile di visualizzare i componenti UI e trasformare un input dell'utente nel valore di un componente. I *rendered* possono essere progettati per funzionare con uno o più componenti UI e un componente UI può essere essere associato a più *rendered*.
Validatore	Responsabile di controllare che il valore inserito da un utente sia accettabile. Uno o più validatori possono essere associati ad un componete UI.
Backing beans	Particolari *JavaBeans* che collezionano i valori dai componenti UI e implementano metodi che rimangono in attesa che un certo evento accada.
Convertitore	Converte il valore di un componente UI da e in una stringa per poterla visualizzare. Un componete UI può essere associato ad un unico convertitore.
Eventi e *Listener*	JSF usa un modello basato su eventi. I componenti UI possono generare eventi e i *listener* possono essere configurati per reagire ad un certo evento.
Messaggi	Informazioni che vengono visualizzate all'utente.
Navigazione	La possibilità di muoversi da una pagina ad un'altra. Il sistema di navigazione usato da JSF è basato anch'esso su eventi.

Tabella 1.1: Tabella dei termini chiave del *framework* JSF [10]

In particolare JSF mette a disposizione una serie di componenti lato server per la realizzazione della *User Interface* (UI) dell'applicazione stessa. Questi componenti possono essere di diversa natura (pulsanti, campi di testo, link , menù e altro) e sono legati tra loro in vario modo formando una struttura ad albero detta vista. La vista è specificata mediante file XML (in termini pratici di solito i file sono basati sullo standard XHTML) chiamati modelli di vista o viste *Facelets* (a seconda della tecnologia utilizzata all'interno del file per specificare la vista). La tecnologia di visualizzazione *Facelets* fornisce un più efficiente, semplice e potente linguaggio di descrizione delle viste. Le richieste sono elaborate dal *FacesServlet*, che carica l'appropriato modello di vista, costruisce l'albero dei componenti, processa gli eventi e interpreta la risposta per il client, tipicamente in HTML.

Come la maggior parte dei *framework*, JSF ha un file di configurazione chiamato *faces-config.xml*. Questo file XML permette di definire regole per la navigazione fra le pagine, inizializzare i *JavaBeans*, registrare i componenti JSF creati e configurare altri aspetti di un'applicazione JSF.

1.3 Enterprise Java Beans

Gli *Enterprise JavaBeans* (EJB) sono i componenti che implementano la logica di business di un'applicazione. Attualmente sono alla terza versione.
Questo *framework* è costituito da componenti che vivono all'interno di un EJB *container*, mostrato in figura 1.7, che fornisce a tali componenti un certo numero di servizi tra i quali la gestione della sicurezza, delle transazioni ed il supporto per i *web-services*.
I componenti EJB sono suddivisi in tre tipi: *session bean*, *message-driven bean* ed *entity bean*. I primi due sono utilizzati per implementare la logica di business mentre gli *entity bean* sono utilizzati per la persistenza.[11]

Gli EJB di sessione o *session* EJB vengono invocati da un client per l'esecuzione di una specifica operazione di business. Il nome "*session*" implica che l'istanza di un *bean* è disponibile per un "unità lavorativa" e non resiste ad un crash o spegnimento del server. Questi elementi possono essere divisi in ulteriori due categorie:

- *session stateful*: memorizzano automaticamente lo stato, il quale può cam-

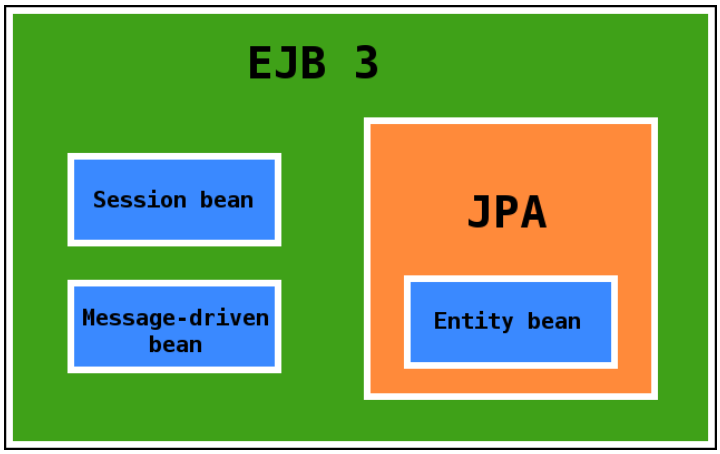


Figura 1.7: Organizzazione delle API di EJB 3

biare tramite l'invocazione di un client, senza il bisogno di scrivere del codice aggiuntivo.

- *session stateless*: contrariamente al caso precedente, questo elemento non possiede uno stato e serve ad implementare servizi che vengono completati in un'unica invocazione da parte dei client.

Gli EJB pilotati da messaggi o *Message Driven* EJB (MDB): sono gli unici EJB ad avere comportamento asincrono, grazie all'utilizzo delle specifiche JMS (*Java Message Service*). Il client che intende invocare queste componenti, deve essere in grado di pubblicare o accodare messaggi su una coda, i quali vengono raccolti al lato server dell'applicazione e forniti ad un MDB. Una volta ricevuto il messaggio, il MDB può invocare in modo sincrono altri tipi di EJB, attendere i risultati di ritorno ed accodare un messaggio di fine elaborazione al client invocante; tutto questo con la possibilità da parte del client di proseguire la sua esecuzione.

Gli EJB di Entità o *Entity* EJB: sono EJB dedicati alla rappresentazione dei dati. Attraverso questi oggetti è possibile leggere e scrivere dati da sorgenti persistenti, generalmente basi di dati.

Dal momento che un EJB *container* offre ai componenti EJB dei servizi, deve essere possibile configurare l'accesso ad essi, ciò può essere fatto in due modi:

- mediante annotazioni, particolari tipi di interfacce che consentono di aggiungere informazioni addizionali, dette attributi, a classi, interfacce, metodi e variabili;

- mediante il *deployment descriptor*, un file XML che contiene le informazioni di configurazione dell'applicazione; in EJB 3 il *deployment descriptor* è opzionale (grazie all'uso delle annotazioni), ma costituisce un buon meccanismo per separare il codice dalla configurazione.

Per assicurare la persistenza degli *entity bean* è necessario ricorrere ad un *persistence provider*. Il quale è essenzialmente un *framework* ORM (*Object-Relational Mapping*), ad esempio *Hibernate* o *TopLink*, che supporta le *Java Persistence* API (JPA). Il termine ORM denota essenzialmente il processo di mappatura dei dati fra gli oggetti e le tabelle di una base di dati mediante annotazioni o attraverso il *deployment descriptor*. Le JPA consentono di:

- configurare l'ORM;
- effettuare operazioni CRUD (*Create*, *Read*, *Update*, *Delete*) sulle entità;
- cercare e recuperare entità persistenti mediante JPQL (*Java Persistence Query Language*).

Capitolo 2

Progettazione del flusso aziendale e specifica dei requisiti

In questo capitolo verrà fornita una più approfondita introduzione del progetto affrontato per poi passare all'analisi dei requisiti e alla progettazione del flusso aziendale effettuata con la tecnica *Value Stream Mapping*[12], una metodologia *lean*. Verrà anche descritto lo standard "830-1998, IEEE *Recommended Practice for Software Requirements Specification*"[1], utilizzato per redigere il documento di specifica dei requisiti.

2.1 Introduzione del progetto

Il progetto è stato svolto nell'azienda *Carel Industries* S.p.A., un'azienda tra le più importanti a livello mondiale nei settori della refrigerazione, del condizionamento e dell'umidificazione dell'aria, specializzata nella realizzazione di sistemi di regolazione[6].
Il progetto ha lo scopo di creare un'applicazione web per la gestione delle preserie e dei prototipi in azienda. Questa applicazione farà poi parte del portale aziendale, il cui menù principale è rappresentato in figura 2.1. Il flusso aziendale in questione coinvolge tre diverse funzioni: sviluppo prodotto, logistica e produzione. Le fasi che caratterizzano questo flusso sono:

1. richiesta di produzione;

2. schedulazione della produzione;

3. produzione effettiva;

4. controllo di qualità;

5. validazione finale;

Una volta raggiunta l'approvazione finale, se la preserie ha superato i controlli necessari è quindi vendibile.

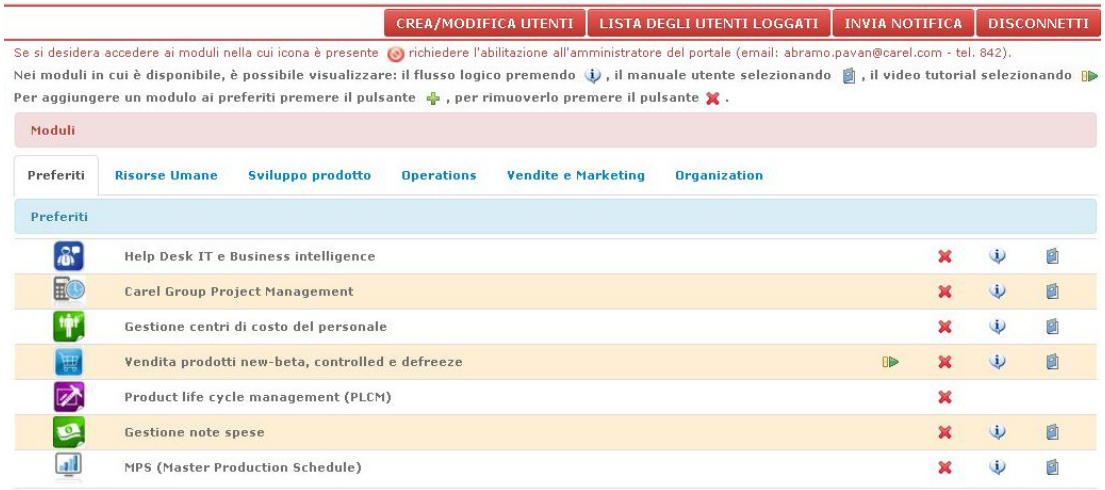


Figura 2.1: Menù principale del portale aziendale in *Carel*

"Il prototipo è il modello originale o il primo esemplare, rispetto a una sequenza di eguali o similari realizzazioni successive"[4]. Essendo un prodotto ancora in fase embrionale non richiede la validazione, il suo flusso all'interno dell'applicazione web termina dopo la produzione. Con preserie si intende un prodotto che è già ad una fase successiva e che necessità di essere validato, cioè di superare una serie di controlli di qualità, per poter diventare vendibile. Esistono due tipi di preserie:

- preserie derivante da un nuovo progetto;
- preserie PLCM (*Product Life-Cycle Management*), cioè la modifica di un prodotto già in vendita che viene chiamato "prodotto padre"; a causa di questa modifica la nuova PLCM necessita di essere validata.

Esiste una sottocategoria delle preserie PLCM, ovvero le PLCM della famiglia PJEZ[1] in cui avvengono solo modifiche firmware o parametriche (sono prodotti

[1]Rappresentano una gamma di regolatori elettronici a microprocessore con visualizzazione a LED realizzati per la gestione di unità frigorifere, vetrine e banchi frigo.[6]

in cui certi parametri sono personalizzabili per il cliente). In questo caso non necessitano di essere prodotte per passare la validazione ed esisterà un tipo di richiesta apposita, chiamata richiesta light(per differenziarla dall'altra che verrà chiamata richiesta standard). Questa tipo di richiesta potrà in seguito essere adottata da altre famiglie di prodotto, puntando a mantenere invariato il flusso dell'applicazione web. L'implementazione della richiesta light non avverrà nella prima versione del software.

2.2 Metodologia lean

Per progettare il flusso aziendale è stata utilizzata la tecnica del *Value Stream Mapping* che fa parte della metodologia *lean*. Il termine *lean production* (produzione snella) è stato ideato nel 1992 dai ricercatori Womack e Jones del *Massachusetts Institute of Technology*, nel "La Macchina che ha cambiato il mondo", in cui illustrano il sistema di produzione che ha permesso all'azienda giapponese *Toyota* di ottenere risultati superiori ai concorrenti nel mondo.[7]
Da allora molte organizzazioni nel mondo hanno adottato il modello *lean*, nell'industria come nei servizi, in quanto applicabile a tutti i processi operativi, quindi non solo strettamente produttivi, ma anche logistici, amministrativi, o di progettazione e sviluppo prodotto.

Figura 2.2: Modello del miglioramento continuo

La produzione snella è un insieme di principi, metodi e tecniche per la gestione dei processi operativi, che mira ad aumentare il valore percepito dal cliente finale

e a ridurre sistematicamente gli sprechi. Per far ciò si applica il modello di miglioramento continuo mostrato in figura 2.2.

Il *lean* si fonda su cinque principi raffigurati in 2.3:

- Valore (*Value*): il punto di partenza è sempre la definizione del valore secondo la prospettiva del cliente. Valore è solo quello che il cliente è disposto a pagare.

- Mappatura (*Map the Value Stream*): per eliminare gli sprechi occorre "mappare" il flusso del valore, ovvero delineare tutte le attività in cui si articola il processo operativo distinguendo tra quelle che aggiungono valore e quelle che non ne aggiungono[13].

- Flusso (*Flow*): il processo di creazione del valore è visto come un flusso che deve scorrere in modo continuo.

- Produzione "tirata" (*Pull*): soddisfare il cliente significa produrre solo quello che vuole, solo quando lo vuole e solo quanto ne vuole. La produzione è così "tirata" dal cliente, anziché "spinta" da chi produce.

- Perfezione (*Perfection*): è il punto di riferimento a cui si deve tendere attraverso il miglioramento continuo e corrisponde alla completa eliminazione degli sprechi.

Figura 2.3: I principi fondamentali della metodologia *lean*[5]

È spreco tutto ciò che consuma risorse, in termini di costo e tempo, senza però creare valore per il cliente. Gli sprechi sono classificati in otto tipologie (vedi figura 2.4), tra cui la più grave è la sovrapproduzione (*overproduction*), in quanto è all'origine di altri tipi di sprechi, in particolare delle scorte(*excess inventory*), dei difetti (*defects*) e dei trasporti(*trasporting*).

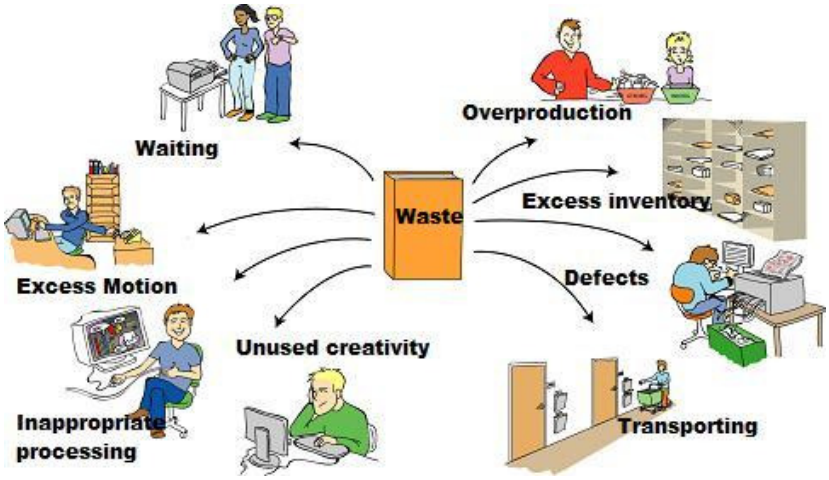


Figura 2.4: Gli otto tipi di spreco della metodologia *lean*

2.3 Progettazione del flusso aziendale

Per progettare il flusso delle preserie/prototipi si è quindi partiti dalla mappatura dello stato attuale della tecnica del *Value Stream Mapping*. La quale nasce con lo scopo di descrivere i processi industriali. Si è dovuto quindi adattarla per rappresentare il flusso all'interno di un'applicazione web. Come prima cosa va modellato quello che viene chiamato "modello dello stato attuale", nel quale va raffigurato come viene gestito in quel momento il flusso aziendale che si sta analizzando. Questo deve essere la base di partenza da cui poi vanno eliminati gli sprechi, fino ad ottenere il "modello dello stato futuro".
Si è quindi partiti identificando chi è il cliente, in questo caso è quello che verrà chiamato richiedente e rappresenta il dipendente *Carel* che richiede la produzione di un prototipo o la validazione di una certa preserie. Il cliente viene raffigurato con l'icona in figura 2.5.
Si vanno poi ad identificare gli altri profili che fanno parte del flusso. Questi verranno raffigurati come in figura 2.6, dove l'intestazione identifica il profilo e

Figura 2.5: Icona che indica il cliente nel *Value Stream Mapping*

sotto sono indicate tutte le azioni che deve portare a termine in quella fase. Le icone sono collegate attraverso frecce formando il flusso del processo aziendale studiato.

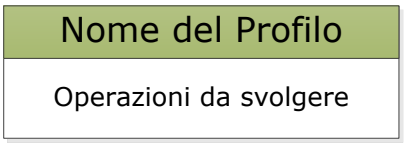


Figura 2.6: Icona che indica un profilo e le operazioni che deve compiere

Nello stato attuale gli altri profili identificati sono:

- *pianificazione*: personale del reparto *Operations* addetto alla pianificazione degli ODL (Ordini Di Lavoro) sulle linee produttive;
- *produzione*: personale del reparto *Plant* che si occupa delle linee produttive.

Il flusso aziendale in analisi era gestito attraverso due moduli cartacei, uno per le richieste standard ed uno per le richieste light, che sono allegati in appendice A. Analizzando il documento cartaceo, si è quindi giunti alla mappatura dello stato attuale rappresentata in figura 2.7. Le macro fasi riscontrabili nel flusso sono:

1. il richiedente inserisce i dettagli riguardanti la richiesta di produzione della preserie/prototipo e riguardanti il materiale in stato *new beta*, cioè il materiale che non è ancora stato ufficialmente omologato e standardizzato nella produzione *Carel*: perché è del tutto nuovo o è ancora in fase di test;
2. la pianificazione pianifica la produzione della preserie/prototipo;
3. la produzione approva la data di completamento del processo;

4. avviene la produzione vera e propria della preserie/prototipo, nel caso sia un prototipo il flusso termina qui;

5. il richiedente esegue la validazione di prodotto, cioè verifica che il prodotto sviluppato sia conforme agli obiettivi stabiliti, e la validazione di processo produttivo, cioè la validazione di tutte le fasi di produzione del prodotto affinché siano sempre esattamente replicabili ed ingegnerizzate per essere efficienti;

6. nel caso di sito produttivo esterno, il *project leader* esegue la validazione di prodotto;

7. la produzione esegue la validazione finale;

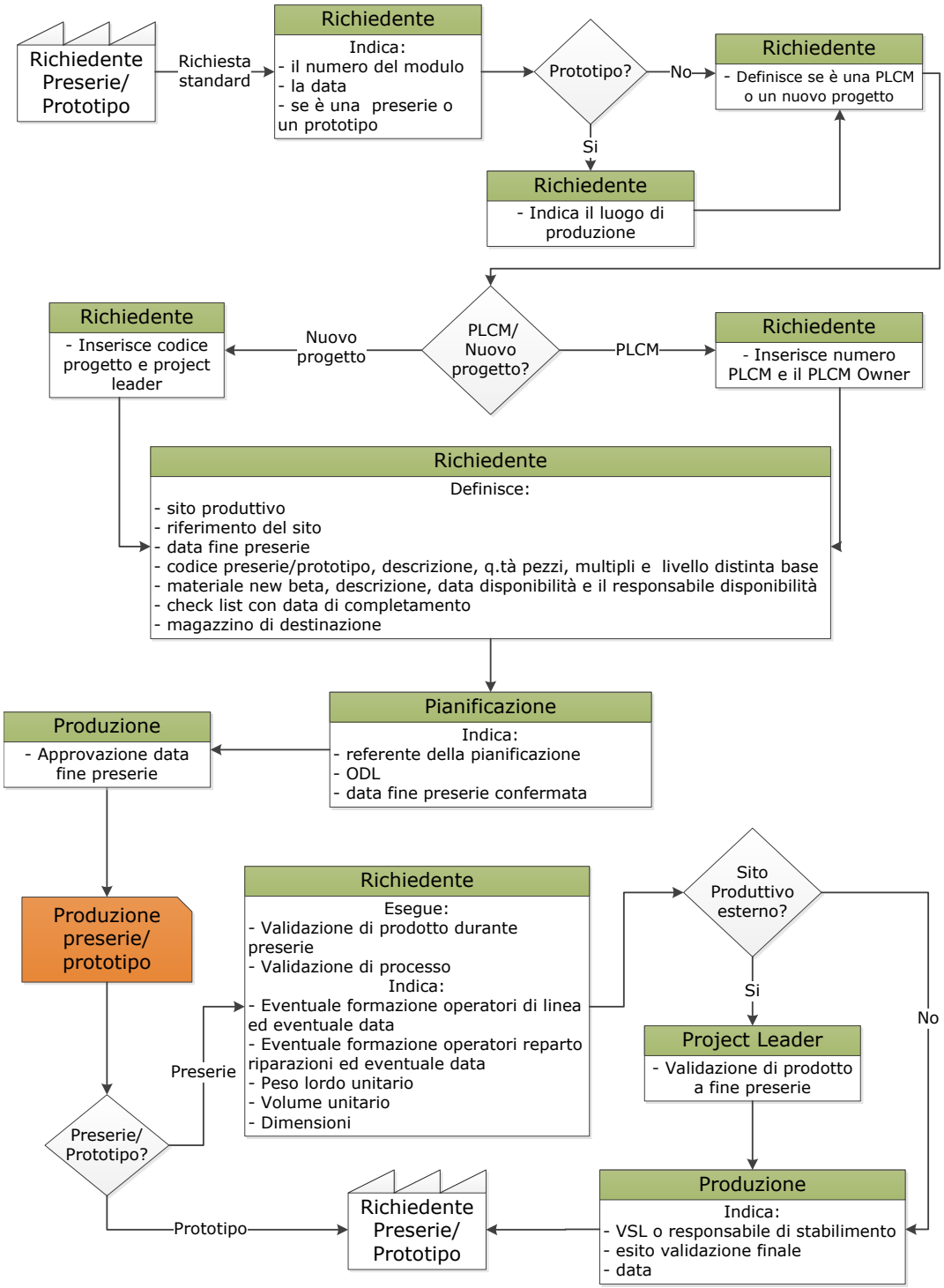


Figura 2.7: Lo stato attuale della tecnica *Value Stream Mapping*

Si è quindi passati alla prima modellazione dello stato futuro, rappresentato in figura 2.8. Il risultato finale è arrivato a seguito di diverse riunioni. Con un fumetto sono indicati i punti che necessitavano di essere migliorati.
Una delle principali modifiche è stato l'inserimento di un nuovo profilo chiamato *owner*. L'*owner* è un dipendente *Carel* che in accordo con il richiedente sarà responsabile della preserie/prototipo lungo tutto il flusso. Questo permette di diminuire i compiti del richiedente e affidarli all'*owner*, il quale dovrebbe avere competenze maggiori. Inoltre il profilo produzione è stato sostituito dal *VSL* (*Value Stream Leader*). La differenza sta nel fatto che prima poteva essere un qualsiasi dipendente del reparto produzione a prendersi in carico la preserie/ prototipo, ora invece verrà indicato dal richiedente. Il VSL potrà essere o il responsabile di una linea produttiva o il responsabile di un sito produttivo estero. Questo ha permesso di rimuovere la doppia validazione di prodotto presente nello stato attuale in figura 2.7, la quale era necessaria per le preserie/prototipi prodotti all'estero. Altre modifiche sono state:

- per i prototipi non è più necessario indicare il luogo produttivo, il quale poteva essere una linea produttiva o *outsourcing*[2]. Questo modulo verrà utilizzato soltanto per i prototipi che utilizzeranno le linee produttive della sede centrale o di una qualsiasi delle filiali;
- per le PLCM non è più necessario indicare l'*owner* delle PLCM, è stata ritenuta un'informazione superflua nel flusso della preserie. Nei casi in cui fosse necessario è comunque possibile ricavarlo dal numero della PLCM;
- discorso analogo vale per il *project leader* di un nuovo progetto;
- non è più necessario indicare il responsabile della disponibilità del materiale in stato *new beta* poiché sarà l'*owner* stesso ad essere responsabile del reperimento di tale materiale;
- non è più necessario che la produzione approvi la data di fine preserie, il compito è stato assegnato alla pianificazione;
- non sono più richieste informazioni che non risultino utili al fine del flusso di approvazione, quali: peso lordo, volume unitario e dimensioni della preserie.

[2] "E' in economia l'insieme delle pratiche adottate dalle imprese di ricorrere ad altre imprese per lo svolgimento di alcune fasi del processo produttivo" [3]

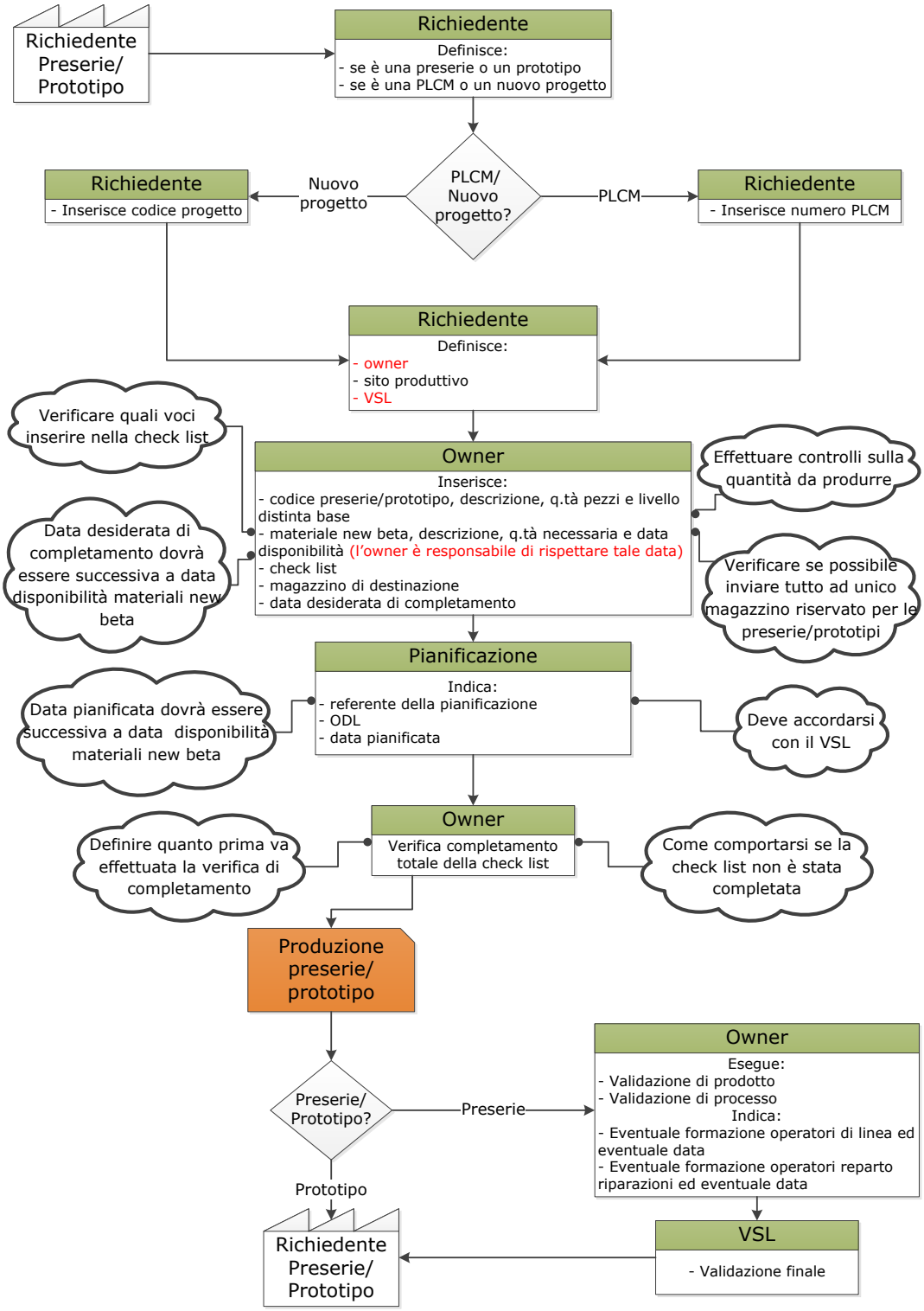


Figura 2.8: La prima versione mappata dello stato futuro

La seconda versione è presentata in figura 2.10, le scritte in rosso sono le modiche effettuate rispetto alla prima versione.
Si è dato la possibilità alla pianificazione di decidere se l'*owner* dovesse essere presente durante la produzione della preserie. Questo permette di inserire la produzione di una preserie in orari in cui l'*owner* non è presente a lavoro, aumentando la capacità produttiva. C'è però il problema che un dipendente del reparto pianificazione potrebbe non avere le competenze necessarie per prendere tale decisione. Si è quindi stabilito che la pianificazione deve accordarsi con il VSL per questa decisione e che il richiedente può inserire una breve motivazione della richiesta di produzione in cui fornire ulteriori dettagli che possono essere utili per decidere.
Si è inoltre pensato di automatizzare la verifica di completamento della *check list* prima della data pianificata e il comportamento da adottare nel caso non fosse completa. In tal caso l'*owner* dovrà richiedere la pianificazione di una nuova data, che naturalmente dovrà essere successiva alla precedente e l'andrà ad inserire nel modulo web. Non si è voluto che tale azione fosse compiuta dal profilo pianificazione per evitare che si formasse un ciclo tra profili diversi in cui il flusso potrebbe incagliarsi. L'obiettivo dell'*owner* è riuscire a validare la preserie, dovrebbe quindi essere interessato ad ottenere una nuova data per portare a termine il suo lavoro.
Sono state forniti inoltre altri possibili miglioramenti:

- automatizzazione del calcolo del rendimento;
- possibilità di indicare attività correttive nella validazione finale;
- definire i parametri secondo cui è possibile evitare la validazione di processo.

E' stata modificata anche la *check list* presente nel documento cartaceo, disponibile in figura 2.9, con le voci presentate in tabella 2.1. L'obiettivo della *check list* è di fare da promemoria per le principali operazioni da compiere prima della produzione della preserie/prototipo. Dato che verrà sottoposta all'*owner*, il quale ha maggiori competenze rispetto al richiedente, è stato possibile rimuovere alcune voci riguardanti operazioni basilari. Sono inoltre state riconosciute delle voci che dovranno essere compiute obbligatoriamente per qualsiasi preserie/prototipo. Si è pensato inoltre, nel caso fosse disponibili, di fornire un link ad un documento contenente le specifiche necessarie per il completamento della voce in questione.

Checklist		**Data di disponibilità / *Availability date***	
Ciclo di lavoro in Oracle (con linea e battente) / *Oracle work instructions / routings*	☐	Distinta Base / *B.O.M.*	☐
Ciclo di lavoro cartaceo / *Paper work instrution / routings*			
Schema elettrico / *Electrical drawings*	☐	Istruzioni di assemb. e imball. / *Assembling & packaging instructions*	☐
Planimetrie / *Electronic hardware layout*	☐	Istruzioni di resinatura / *Coating instructions*	☐
Istruzioni di collaudo / *Test instructions*	☐	Lista attrezzature / *Equipments list*	☐
Lamine per serigrafia SMD / *SMD screen printing frame*	☐	Programma per ispezione ottica / *Optical check software*	☐
Programma per serigrafia SMD / *SMD screen printing software*	☐	Programma per stampigliatura laser / *Laser printing software*	☐
Programma Pick & Place / *Pick & Place software*	☐	Attrezzature di produzione e collaudo / *Production and Tests equip.*	☐
Disponibilità materiale / *Material availability*	☐	Firmware & Software / *Firmware & Software*	☐
Attività/ Controlli supplementari / *Additional activities/checks*	☐	Piano di controllo / *Control plan*	☐
Etichette prodotto / *product labels*	☐	Set-up di linea / *Line set-up*	☐
Note / *Notes*			

Figura 2.9: Check list del modulo cartaceo in appendice A

Voce	**Obbligatorietà**
Ciclo di lavoro in Oracle	obbligatorio
Distinta base	obbligatorio
Disponibilità materiale	obbligatorio
Lamine per serigrafia SMD	non obbligatorio
Programma per serigrafia SMD	non obbligatorio
Programma *Pick and Place*	non obbligatorio
Programma per ispezione ottica	non obbligatorio
Programma per stampigliatura laser	non obbligatorio
Attrezzatura di produzione e collaudo	non obbligatorio
Presenza in produzione di Firmware e Software	non obbligatorio
Etichette prodotto	non obbligatorio
Set-up di linea	non obbligatorio

Tabella 2.1: Tabella delle nuove voci della *check list* con indicato quali voci erano obbligatorie

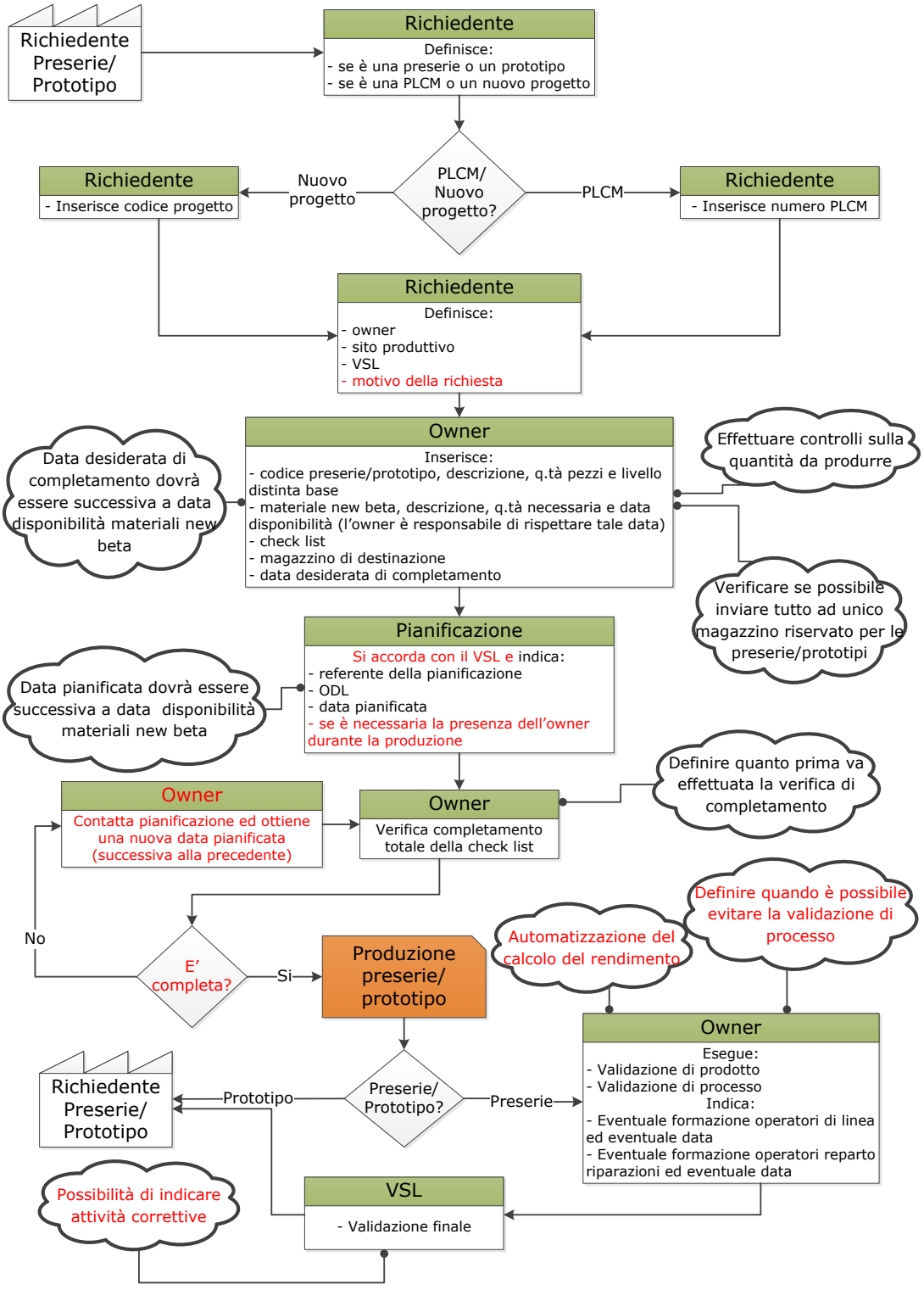


Figura 2.10: Seconda versione mappata dello stato futuro, raffinamento della prima

Nella terza versione in figura 2.11 sono stati inseriti i controlli necessari per le date, cioè né la data di completamento né quella in cui è pianificata la produzione possono essere in una data antecedente a quella di disponibilità dei materiali *new beta*. E' stata inoltre data la possibilità all'*owner* di specificare di voler essere presente durante la produzione, in tal caso la pianificazione non potrà modificare tale decisione e dovrà pianificare la produzione in orari in cui l'*owner* o un suo referente siano presenti.
Si è data la possibilità di indicare da parte dell'*owner* che non è necessaria la validazione di processo motivandone il perché. La validazione di processo può essere evitata se si tratta di una PLCM avente le stesse fasi di processo del "prodotto padre" da cui deriva e tali fasi impiegano gli stessi tempi del "prodotto padre".
E' stato inserito un controllo sulle quantità da produrre. Essendo una preserie è necessario che non vengano prodotti né troppi elementi, dato che potrebbero non essere conformi alle specifiche, né troppo pochi, perché c'è bisogno di una certa significatività statistica. E' stato quindi posto un limite massimo di 50 unità, ma non un limite inferiore perché, soprattutto in ambito di refrigerazione, la produzione di un macchinario può richiedere diverse ore ed imporre un limite minimo andrebbe a limitare la produzione. Un'altra modifica derivante dai tempi di produzione di un macchinario per la refrigerazione è la possibilità da parte del VSL d'indicare delle attività correttive. Questo caso si presenta quando il VSL non validerebbe la preserie presa in considerazione, che però apportando leggere modifiche verrebbe validata. Il VLS assegnerà quindi una serie di attività correttive all'*owner* e una data massima entro cui svolgerle.
E' stato fissato il termine ultimo per completare la *check list*, che sarà entro il terzo giorno lavorativo antecedente alla data di produzione pianifica. Verrà comunque inviata una e-mail di promemoria.
E' stato deciso di automatizzare il calcolo del rendimento che va inserito nella validazione di processo. Il rendimento è dato dal rapporto fra il tempo teorico e quello realmente impiegato per la produzione. Questo viene calcolato autonomamente nelle linee di produzione, però tiene conto anche dei tempi di attività non connessi al ciclo in atto: ad esempio i problemi alle attrezzature o eventuali pause. E' quindi necessario che, per automatizzare questo inserimento, gli operatori di linea vengano istruiti a stoppare il timer nel caso stiano producendo una preserie e accadino eventi non connessi al ciclo in atto.

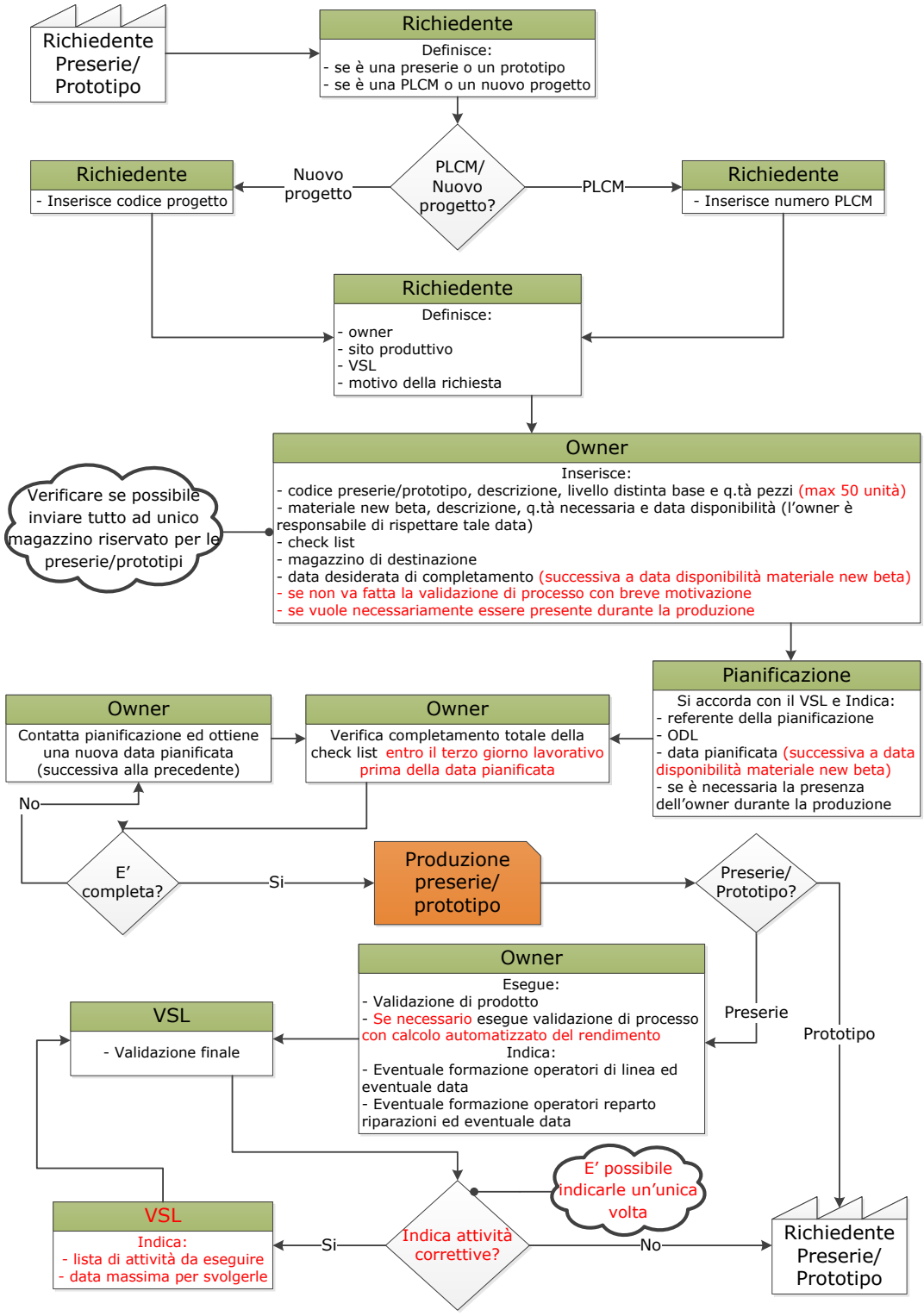


Figura 2.11: Terza versione mappata dello stato futuro, raffinamento della seconda

L'ultima versione è rappresentata in figura 2.12. In questa versione sono state effettuate una serie di piccoli accorgimenti:

- il VSL viene definito dal richiedente solo se non è un prototipo, dato che i prototipi non necessitano di essere validati;
- il livello della distinta base di una certa preserie è stata ritenuta un'informazione non interessante;
- non viene più richiesto il magazzino di destinazione, dato che è stata creata una porzione appositamente dedicata alle preserie;
- la possibilità di non eseguire la validazione di processo è stata data soltanto alle PLCM
- la decisione finale di eseguire o meno la validazione di processo è stata data alla pianificazione, naturalmente dopo essersi confrontata con il VSL;
- è stata data la possibilità alla pianificazione di modificare il VSL, nel caso il richiedente non avesse indicato una persona con le competenze adatte.

In figura 2.13 sono indicate anche le e-mail che vengono inviate durante il flusso della preserie. Sono per lo più e-mail di notifica di superamento di una certa fase del flusso, ci sono inoltre due e-mail di "alert", cioè di promemoria:

- una viene inviata al referente della pianificazione e all'*owner* tre giorni lavorativi prima della data pianificata di produzione, nel caso la *check list* non sia stata ancora completa;
- una viene inviata all'*owner* e al VSL il giorno precedente al termine massimo per svolgere le attività correttive necessarie per la validazione;

Un altro caso particolare è la e-mail che viene inviata al reparto dei riparatori nel caso venga indicato che è stata fornita loro la formazione su una certa preserie, per evitare che vengano inserite informazioni non vere in questo campo.

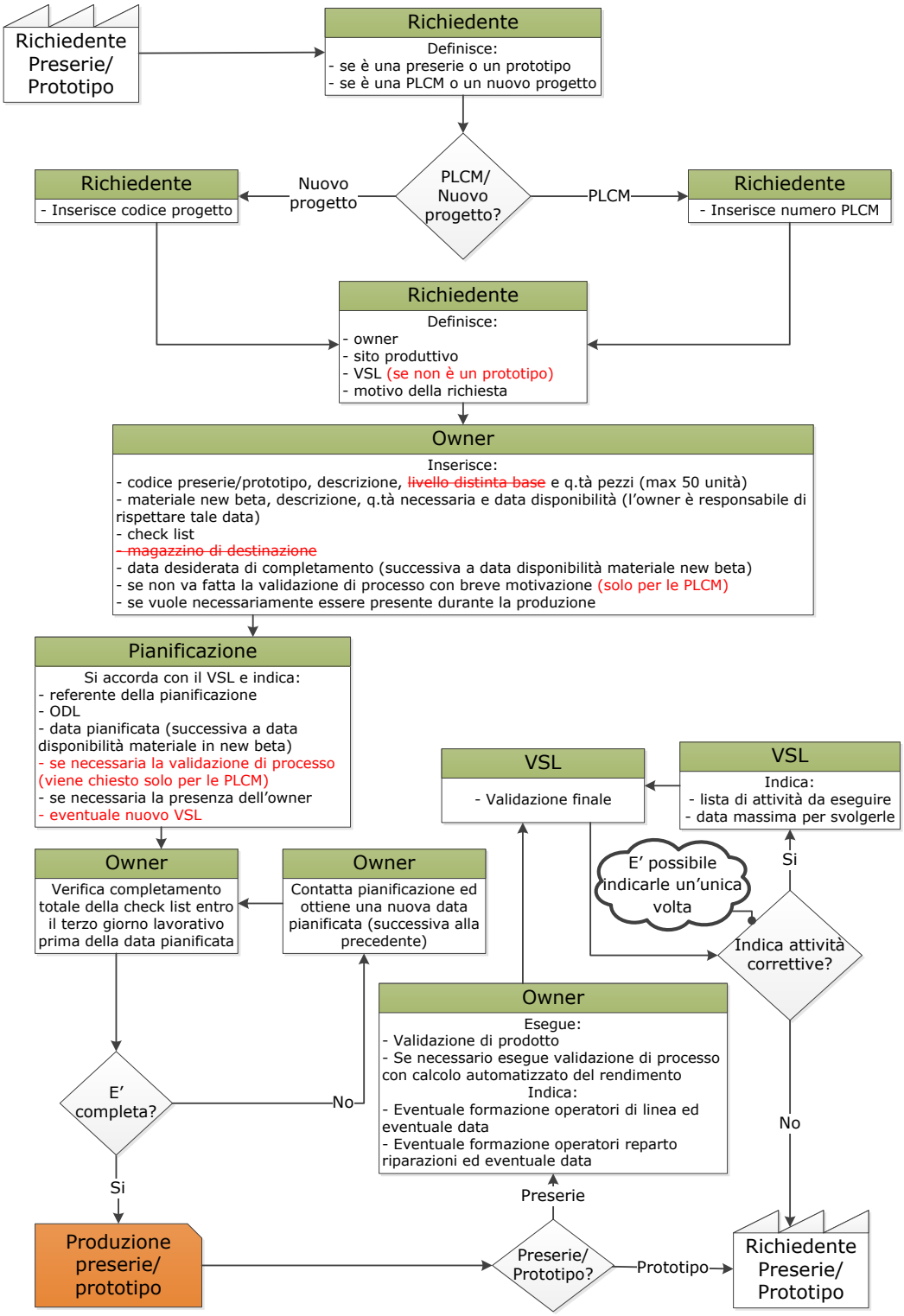


Figura 2.12: Ultima versione mappata dello stato futuro

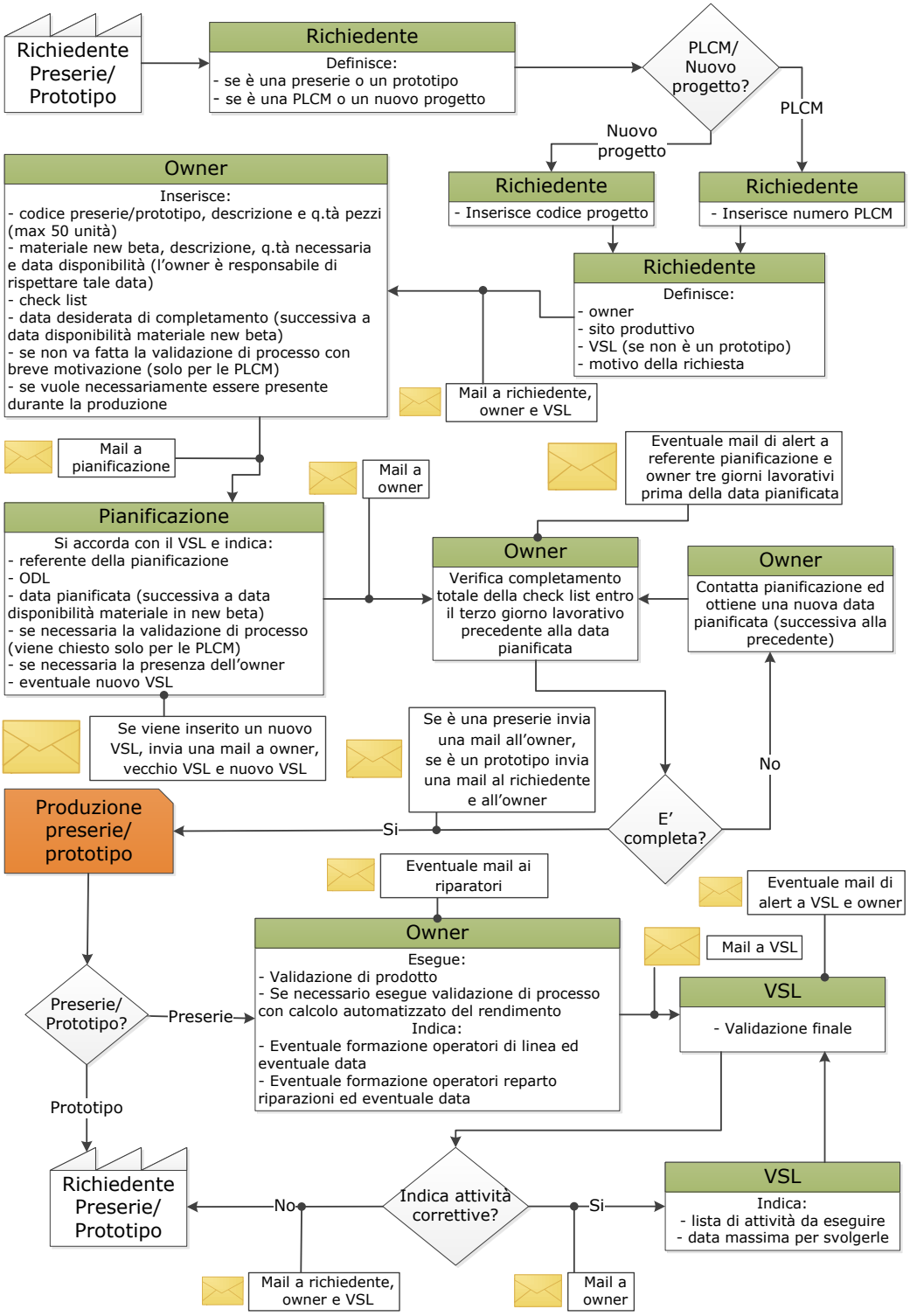


Figura 2.13: Ultima versione mappata dello stato futuro in cui sono indicate anche le e-mail che vengono inviate

2.4 Specifiche del progetto

Nello standard "830-1998, IEEE *Recommended Practice for Software Requirements Specification*" viene descritto come redigere un documento di specifica dei requisiti, il quale offre i seguenti benefici[1]:

- stabilire una base comune di accordo fra il cliente ed il venditore su cosa il software dovrà fare;
- ridurre gli sforzi di sviluppo, la preparazione della stesura dei requisiti obbliga le varie parti in gioco a considerare rigorosamente tutto ciò di cui necessitano prima di passare alla fase di design;
- fornire una base per il calcolo dei costi e dei tempi necessari per lo sviluppo;
- fornire una base per la validazione e la verifica del software;
- serve come base per futuri miglioramenti.

1. INTRODUZIONE

1.1 Scopo del documento
1.2 Definizioni, acronimi ed abbreviazioni
1.3 Riferimenti
1.4 Introduzione del prodotto

2. DESCRIZIONE GENERALE DEL PRODOTTO

2.1 Ambito del prodotto
2.2 Funzionalità
2.3 Utenti destinatari
2.4 Vincoli
2.5 Assunzioni

3. SPECIFICA DEI REQUISITI

3.1 Organizzazione per classe di funzione
3.2 Organizzazione per classe di utenti
3.3 Organizzazione per classe di utenti e per classe di funzione

Figura 2.14: Scheletro delle specifiche dei requisiti

Lo scheletro seguito per la stesura del documento è mostrato in figura 2.14 e le parti che vanno a comporlo sono le seguenti:

- *scopo del documento* in cui va indicato il fine del documento (e non del prodotto di cui si stanno scrivendo le specifiche) ed i destinatari;
- *definizioni, acronimi ed abbreviazioni* in cui vanno riportate tutte le definizioni, acronimi od abbreviazioni utilizzate nel documento;
- *riferimenti* in cui vanno indicati gli altri documenti che sono stati utilizzati;
- *introduzione del prodotto* in cui va data una descrizione sintetica del prodotto di cui si sta scrivendo le specifiche. In particolare bisogna evidenziare il codice identificativo del prodotto (il nome con cui è univocamente riconoscibile sia dall' *IT Department* sia dagli utenti finali). Riportare quindi una breve descrizione di cosa il software farà (e se necessario anche cosa non farà). Infine specificare quale sarà l'utilizzo del software da parte degli utenti includendo benefici, obiettivi e risultati conseguiti;
- *ambito del prodotto* dove va descritto l'ambito di funzionamento del prodotto, ossia come esso si integra con gli altri sistemi aziendali. Se il prodotto è indipendente allora va chiaramente specificato. Vanno descritte in maniera sintetica le interfacce che il prodotto deve avere con il sistema (sia hardware che software) e con gli utenti;
- *funzionalità* in cui va riportata la descrizione funzionale del prodotto. Questa panoramica del prodotto non deve entrare troppo in dettaglio e serve per dare a chi legge il documento una visione generale delle funzionalità del sistema in modo da poter poi entrare nello specifico leggendo il capitolo successivo dove viene riportata la lista completa e dettagliata;
- *Utenti destinatari* il cui contenuto è una descrizione delle tipologie di utenti che utilizzeranno il prodotto. In particolare per ognuna di esse vanno indicate le competenze tecniche e quelle funzionali;
- *vincoli* dove fornisce la lista di tutti i vincoli progettuali e funzionali a cui il sistema è sottoposto. Le tipologie standard sono:
  - *policies* aziendali;
  - limitazioni hardware;
  - interfacce con altre applicazioni;

  - vincoli di programmazione (specifica di un determinato linguaggio di programmazione da utilizzare e/o particolari librerie o pacchetti software);
  - parametri di affidabilità da rispettare;
  - vincoli di sicurezza;
  - criticità delle applicazioni coinvolte;

- *assunzioni* in cui vanno indicate tutte le ipotesi ed assunzioni fatte nell'analisi del problema che il prodotto va a risolvere;

- *specifica dei requisiti* in cui va riportata la lista completa dei requisiti del prodotto. Ogni requisito va riportato con un grado di dettaglio tale da non dare dubbi di interpretazione e da permettere ai progettisti di realizzare il software ed ai tester di verificare il corretto funzionamento del modulo. Tuttavia non devono essere riportati dettagli di implementazione o progettazione del prodotto: deve essere scritto in maniera chiara cosa si deve fare ma non come il prodotto deve essere realizzato per farlo. La modalità di organizzazione dei requisiti in questo paragrafo dipende dalla tipologia di prodotto. Le principali tipologie di rappresentazione della lista dei requisiti sono essenzialmente due:
  - per classe di funzione in cui bisogna suddividere i requisiti funzionali per tipologia;
  - per classe di utente in cui i requisiti funzionali vengono suddivisi per tipologia di utenza;
  - combinazione dei due metodi precedenti, cioè all'interno di una suddivisione per classi di utenti si suddivide anche per classi di funzione, o il viceversa;

In tabella 2.2 sono presentate sinteticamente le descrizioni delle voci spiegate precedentemente per il progetto realizzato. E' possibile consultare le specifiche complete in appendice B.

Tabella 2.2: Tabella di sintesi delle specifiche del progetto

Voce	Descrizione
Scopo del documento	Lo scopo di questo documento è specificare i requisiti funzionali e tecnici del modulo del portale per la richiesta di validazione di una preserie o la richiesta di produzione di un prototipo. I destinatari della SRS sono i reparti IT, *operations*, qualità e sviluppo prodotto.
Definizioni, acronimi ed abbreviazioni	SRS *Software Requirement Specification* VSL *Value Stream Leader* PLCM *Product Life-Cycle Management* ODL Ordine Di Lavoro
Introduzione del prodotto	E' stata data un'introduzione del prodotto simile a quella nella sezione 2.1
Ambito del prodotto	Il modulo in oggetto è un'applicazione web *standalone* che verrà integrata all'interno del portale aziendale. La base di dati utilizzata è *Oracle*.
Funzionalità	Inserimento nel sistema tipologia di richiesta Inserimento nel sistema richiesta light Inserimento richiesta standard (parte richiedente) Inserimento richiesta standard (parte *owner*) Pianificazione Verifica della *check list* Realizzazione della preserie/prototipo in produzione Validazione di prodotto e validazione di processo Validazione finale Ricerca nello storico
Continua nella prossima pagina	

Continua dalla pagina precedente

Voce	**Descrizione**
Utenti destinatari	I profili sono: • VSL, personale interno *Carel* che validerà o non validerà la preserie richiesta; • Pianificazione, personale di *operations* addetto alla pianificazione degli ODL; • User utilizzato dal richiedente e dall'*owner*.
Vincoli	I vincoli progettuali, tecnologici e funzionali a cui il prodotto è sottoposto sono i seguenti: • La tecnologia utilizzata deve essere JSF (1.2) e EJB3; • Il modulo web deve utilizzare come web *container* GlassFish 2 (ultima release stabile disponibile per questa versione); • Il sorgente del modulo deve essere realizzato come progetto per *Oracle Jdeveloper 11g*; • Le e-mail di *alert* verranno inviate solo per le preserie che verranno prodotte in *Carel Industries*, *Carel* Asia e *Carel* USA poiché sono le uniche organizzazioni aventi il calendario nella base di dati *Oracle*.
Specifica dei requisiti	La lista completa dei requisiti del prodotto è stata suddivisa per tipologia utente e per classe funzionale.

Capitolo 3

Progettazione e implementazione della base di dati

In questo capitolo verrà spiegato come è stata progettata la basi di dati per la gestione delle richieste di produzione di preserie/prototipi. Dopo una breve premessa, si partirà con la costruzione dello schema concettuale, utilizzando la tecnica *Top-Down*, che verrà poi tradotto in schema logico e normalizzato. In finale verrà mostrata la fase di implementazione della base di dati in *Oracle*.

3.1 Premesse

La base di dati che verrà progettata farà parte di quella già esistente per il portale aziendale. Soprattutto si interfaccerà con l'entità già implementata "utente" la cui chiave primaria è "USER_ID". Gli scopi di questa base di dati sono di tenere traccia della fase del processo in cui si è arrivati e di fornire un'efficiente ricerca nello storico delle richieste di produzione di preserie/prototipi.

3.2 Schema concettuale

Si procede ora alla creazione di uno schema concettuale basato sul modello Entità-Relazione. La strategia utilizzata sarà di tipo *Top-Down*, cioè si partirà da un concetto più generale per andare poi a definire più concetti particolari che sono di interesse per l'applicazione[8]. Si è partiti infatti dal concetto generale, rappresentato in figura 3.1, che è la richiesta di produzione di preserie/prototipi

da parte di un utente, la quale può necessitare di materiale in stato *new beta* e del completamento di una *check list*.

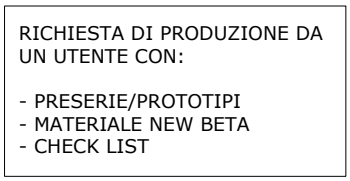


Figura 3.1: Entità che identifica il concetto generale

Questo concetto in realtà può essere rappresentato come la relazione fra i dettagli della richiesta e gli utenti che ne fanno parte, come si può notare in figura 3.2.

Figura 3.2: Prima trasformazione dell'entità generale

In realtà poi la relazione "in relazione" può essere ulteriormente specificata, come mostrato in figura 3.3, in quattro relazioni: "crea", "è responsabile", "pianifica" e "valida". Queste rappresentano le quattro operazioni eseguite rispettivamente dal richiedente, dall'*owner*, dal referente della pianificazione e dal VSL.

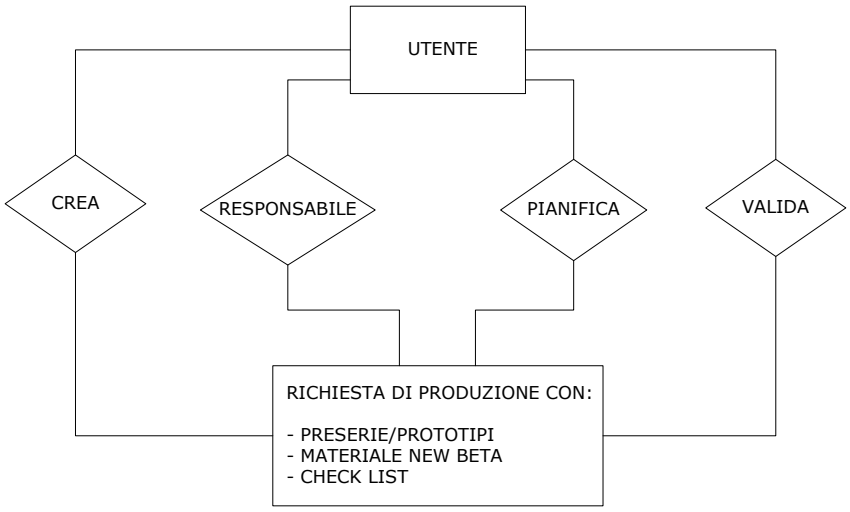


Figura 3.3: Trasformazione della relazione fra l'utente e la richiesta di produzione

In una richiesta di produzione sono contenuti anche i dati riguardanti le preserie/prototipi, si trasforma così l'entità precedente in una nuova relazione fra due entità, come rappresentata in figura 3.4.

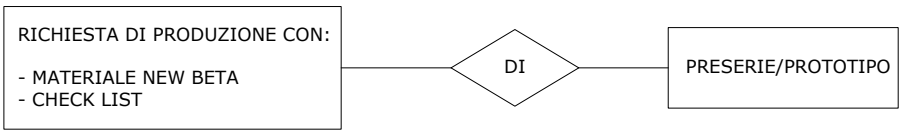


Figura 3.4: Ulteriore trasformazione della richiesta di produzione

Analogamente si può applicare lo stesso ragionamento per il materiale in stato *new beta* richiesto, si ottiene quindi la relazione in figura 3.5.

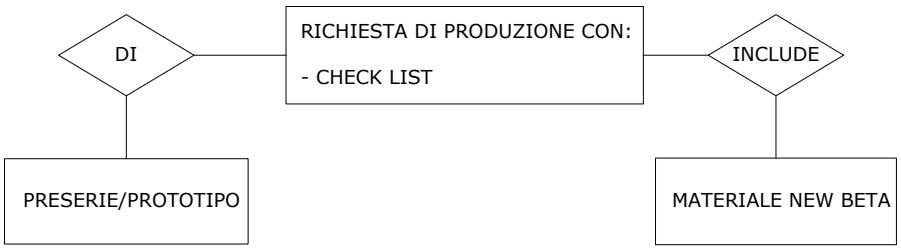


Figura 3.5: Ulteriore trasformazione della richiesta di produzione

In ultima, si applica lo stesso procedimento anche per le voci della *check list* che necessitano di essere completate per poter avviare la produzione. Il risultato è visibile in figura 3.6.

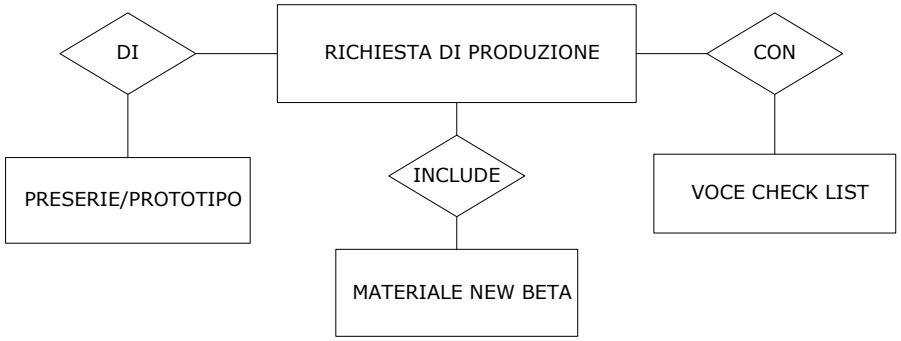


Figura 3.6: Ulteriore trasformazione della richiesta di produzione

In figura 3.7 è rappresentato lo schema concettuale completo, in cui sono stati inseriti anche i vincoli di cardinalità. Dato l'elevato numero di attributi per non

perdere in chiarezza, sono state indicate solo le chiavi primarie. Tutti gli attributi sono comunque descritti nelle tabelle seguenti:

- in tabella 3.1 sono indicati gli attributi dell'entità "richiesta di produzione";
- in tabella 3.2 sono indicati gli attributi dell'entità "preserie/prototipo";
- in tabella 3.3 sono indicati gli attributi dell'entità "materiale *new beta*";
- in tabella 3.4 sono indicati gli attributi dell'entità "voce *check list*";

.

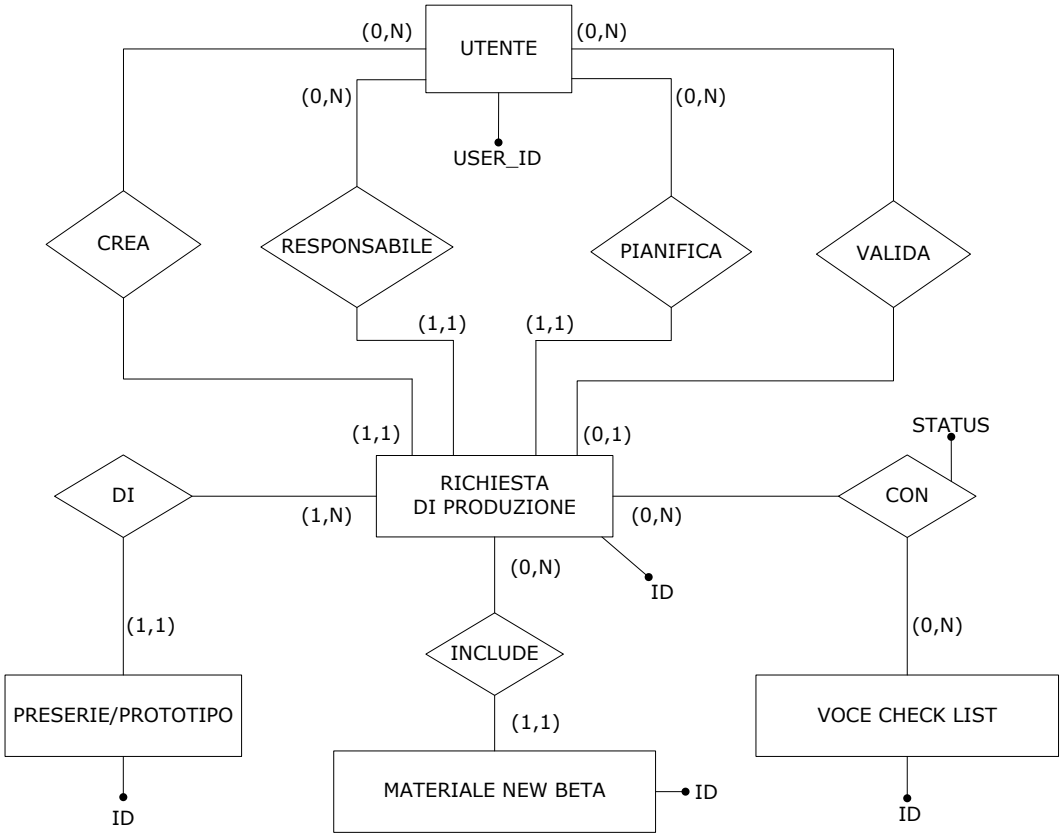


Figura 3.7: Schema concettuale completo

Tabella 3.1: Tabella degli attributi dell'entità "richiesta di produzione"

Attributo	Descrizione
ID	Chiave primaria dell'entità, identifica univocamente una richiesta di produzione di una o più preserie/-prototipo
CREATION_DATE	Data di creazione
ORG_ID	Codice che identifica da che stabilimento viene effettuata la richiesta
TYPE	Indica se è una preserie o un prototipo
PRESERIES_TYPE	Indica se è una preserie derivante da un nuovo progetto o una PLCM
PROJECT_CODE	Codice del progetto
PLCM_NUMBER	Numero della PLCM
PRODUCTION_SITE	Sito di produzione
CREATION_NOTE	Informazioni aggiuntive che vengono fornite dal richiedente al momento della creazione della richiesta di produzione
STATUS	Identifica la fase del processo in cui si trova la preserie/prototipo
REQUESTED_DATE	Data richiesta di completamento
PROCESS_VAL_REQ	Identifica se la validazione di processo è necessaria secondo l'*owner*
PROCESS_VAL_REQ_NOTE	Note che motivano il fatto che la validazione di processo non sia richiesta
PLANNING_PROCESS_VAL_REQ	Identifica se la validazione di processo è necessaria secondo la pianificazione

Continua nella prossima pagina

Continua dalla pagina precedente

Attributo	Descrizione
CHECKLIST_COMPLETION_DATE	Data di completamento della *check list*
REFERENCE_CONTROL_PLAN	Piano di controllo di riferimento
PRODUCT_VAL_RESULT	Risultato della validazione di prodotto
PRODUCT_VAL_NOTE	Note sulla validazione di prodotto
PRODUCT_VAL_DATE	Data della validazione di prodotto
PROCESS_VAL_RESULT	Risultato della validazione di processo
PROCESS_VAL_NOTE	Note sulla validazione di processo
PROCESS_VAL_DATE	Data della validazione di processo
PROCESS_VAL_EFF	Rendimento ottenuto in fase di produzione
LINE_OPERATORS_TRAINING	Indica se è stata effettuata la formazione agli operatori di linea
LINE_OPERATORS_TRAINING_DATE	Data di formazione degli operatori di linea
SERVICE_DEPT_TRAINING	Indica se è stata effettuata la formazione agli operatori del reparto riparazioni
SERVICE_DEPT_TRAINING_DATE	Data di formazione degli operatori del reparto riparazioni
FINAL_VAL	Esito della validazione finale
FINAL_VAL_AFTER_ACTION_TO_DO	Esito della validazione finale dopo eventuali attività correttive
FINAL_VAL_NOTE	Note sulla validazione finale
FINAL_VAL_DATE	Data di validazione finale
ACTION_TO_DO_REQUEST_DATE	Data massima entro cui eseguire le attività correttive

Continua nella prossima pagina

Continua dalla pagina precedente

Attributo	Descrizione
ACTION_TO_DO_NOTE	Attività correttive suggerite durante la validazione finale

Tabella 3.2: Tabella degli attributi dell'entità "preserie/prototipo"

Attributo	Descrizione
ID	Codice che identifica univocamente la richiesta di produzione di una certa preserie/prototipo
ITEM_CODE	Codice della preserie/prototipo
ITEM_DESCRIPTION	Descrizione della preserie/prototipo
ITEM_ID	ID della preserie/prototipo
QUANTITY	Quantità da produrre
ODL	Codice dell'ODL
OWNER_PRESENCE_REQUIRED	Identifica se è richiesta la presenza dell'*owner* durante la produzione
PLANNED_DATE	Data pianificata per la produzione
NEW_PLANNED_DATE	Data pianificata per la produzione, nel caso la *check list* non sia stata terminata prima della data stabilita
PRODUCTION_DATE	Data effettiva di produzione

Tabella 3.3: Tabella degli attributi dell'entità "materiale *new beta*"

Attributo	Descrizione
ID	Codice che identifica univocamente la richiesta di produzione di un certo materiale in stato *new beta*
ITEM_CODE	Codice del materiale
ITEM_DESCRIPTION	Descrizione del materiale
ITEM_ID	ID del materiale

Continua nella prossima pagina

Continua dalla pagina precedente

Attributo	Descrizione
AVAILABILITY_DATE	Data di disponibilità del materiale

Tabella 3.4: Tabella degli attributi dell'entità "voce *check list*"

Attributo	Descrizione
ID	Codice che identifica univocamente una voce della *check list* che si dovrà eseguire
TYPE	Codice della tipologia della voce
DESCRIPTION_IT	Descrizione in italiano della voce della *check list*
DESCRIPTION_EN	Descrizione in inglese della voce della *check list*
LINK	Link al file contenente le specifiche utili per il completamento della voce

3.3 Schema logico relazionale e le sue normalizzazioni

A seguito di *policies* aziendali, i nomi delle entità sono stati così tradotti:

- "richiesta di produzione" in "CAREL_PRESERIES_H";
- "preserie/prototipo" in "CAREL_PRESERIES_L";
- "materiale *new beta*" in "CAREL_PRESERIES_MATERIAL";
- "voce *check list*" in "CAREL_PRESERIES_CHECKLIST".

Si può notare in figura 3.8 che eseguendo la traduzione in schema logico è stata creato lo schema "CAREL_PRESERIES_CHECK_L", che ha lo scopo di gestire la relazione molti a molti presente fra la "richiesta di produzione" e la

"voce *check list*".
Inoltre, nello schema "CAREL_PRESERIES_H" sono apparse le chiavi esterne: "CREATED_BY", "OWNER", "VSL" e "PLANNING_REFERENT" che fanno riferimento alla tabella "JUSER". Queste rappresentano i quattro profili che entrano in gioco, rispettivamente il richiedente, l' *owner*, il VSL ed il referente della pianificazione. In ciascuna degli schemi "CAREL_PRESERIES_L", "CAREL_PRESERIES_MATERIAL" e "CAREL_PRESERIES_CHECK_L" è comparsa la chiave esterna "HEADER_ID" che fa riferimento alla tabella "CAREL_PRESERIES_H", che rappresenta la richiesta di produzione.
Tutti gli schemi di relazione sono in prima forma normale in quanto tutti gli attributi sono atomici. Sono anche in seconda forma normale perché la chiave dello schema è formata da un solo attributo, quindi non possono esserci dipendenze parziali dalla chiave. I seguenti schemi invece non sono in terza forma normale, poiché per ognuno di questi esiste una dipendenza funzionale $X \rightarrow Y$ in cui né X è una superchiave[1] dello schema in questione, né Y è un attributo primo[2], e sono:

- per lo schema "CAREL_PRESERIES_H" esiste la dipendenza funzionale "CREATED_BY"→ "ORG_ID";

- per lo schema "CAREL_PRESERIES_L" esiste la dipendenza funzionale "ITEM_ID"→"ITEM_DESCRIPTION";

- per lo schema "CAREL_PRESERIES_MATERIAL" esiste la dipendenza funzionale "ITEM_ID"→"ITEM_DESCRIPTION";

Anche se lo schema logico non si trova in terza forma normale non si sono eseguite scomposizioni perché le dipendenze funzionali sono causate dalla scelta di mantenere separata la parte di base di dati riguardanti l'inventario da quella riguardante il portale.

[1] K è superchiave per uno schema $R(X)$, dove X è l'insieme degli attributi, se e solo se R non contiene due n-uple distinte t_1 e t_2 tali che $t_1[K] = t_2[K]$

[2] Un attributo si dice primo se è parte di una qualche chiave candidata dello schema

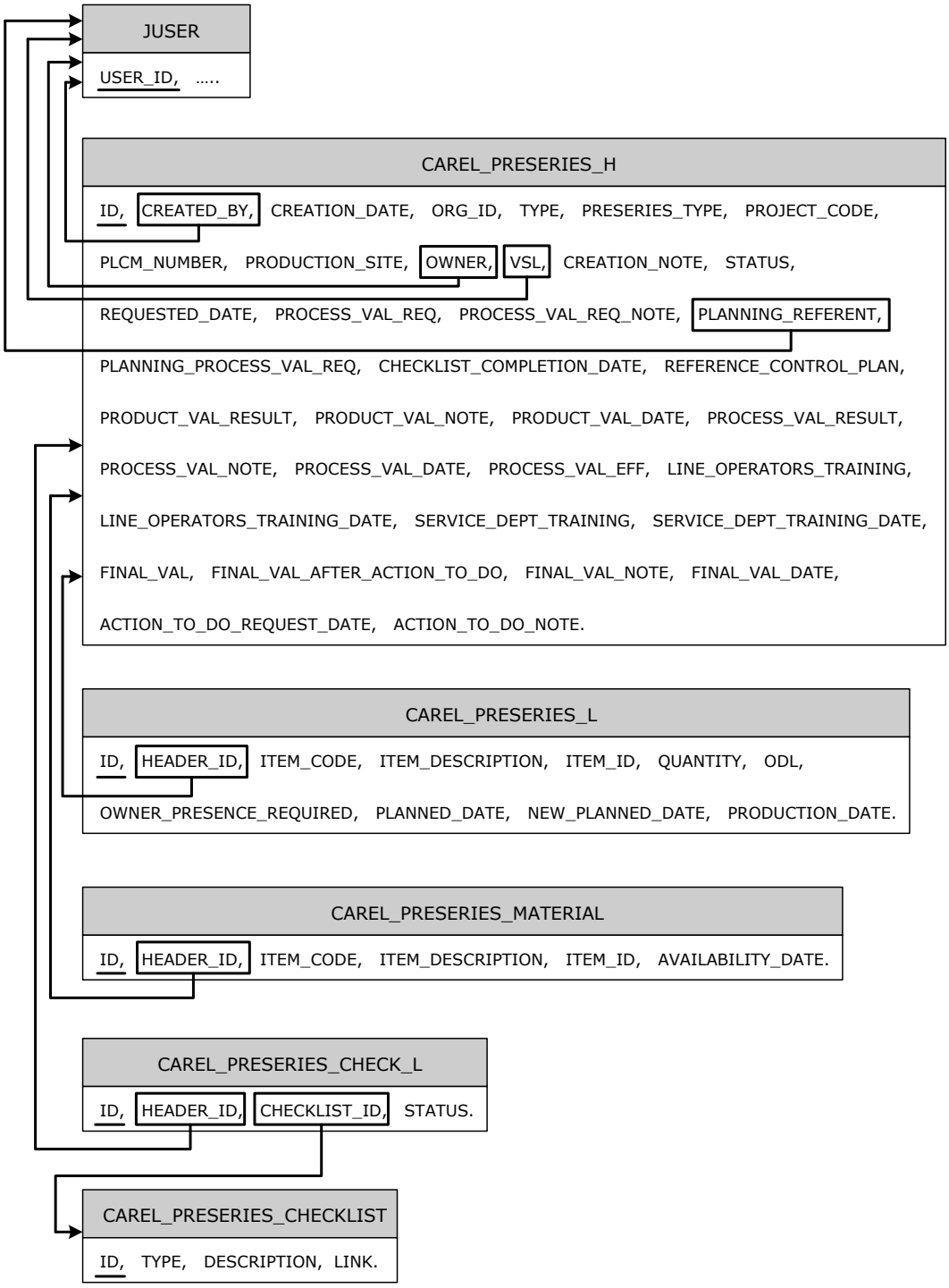


Figura 3.8: Schema logico della base di dati

3.4 Implementazione sulla base di dati *Oracle*

Qui di seguito è presentato il codice utilizzato per l'implementazione della base di dati su una base di dati *Oracle*.

Listing 3.1: Implementazione della basi di dati in *Oracle*

```
CREATE TABLE CAREL_PRESERIES_H  (
    -- Petitioner part
    ID                              NUMBER NOT NULL,
    CREATED_BY                      NUMBER NOT NULL,
    CREATION_DATE                   TIMESTAMP NOT NULL,
    ORG_ID                          NUMBER NOT NULL,
    TYPE                            NUMBER NOT NULL, -- preseries or prototype
    PRESERIES_TYPE                  NUMBER NOT NULL, -- new project or plcm
    PROJECT_CODE                    VARCHAR2(1000),
    PLCM_NUMBER                     NUMBER,
    PRODUCTION_SITE                 NUMBER,
    OWNER                           NUMBER,
    VSL                             NUMBER,
    CREATION_NOTE                   VARCHAR2(4000),
    STATUS                          NUMBER NOT NULL,
    -- Owner request part
    REQUESTED_DATE                  TIMESTAMP,
    PROCESS_VAL_REQ                 NUMBER,
    PROCESS_VAL_REQ_NOTE            VARCHAR2(4000),
    -- Planning part
    PLANNING_REFERENT               NUMBER,
    PLANNING_PROCESS_VAL_REQ        NUMBER,
    --Owner Confirmation
    CHECKLIST_COMPLETION_DATE       TIMESTAMP,
    -- Owner validation
    REFERENCE_CONTROL_PLAN          VARCHAR2(4000),
    PRODUCT_VAL_RESULT              NUMBER,
    PRODUCT_VAL_NOTE                VARCHAR2(4000),
    PRODUCT_VAL_DATE                TIMESTAMP,
    PROCESS_VAL_RESULT              NUMBER,
    PROCESS_VAL_NOTE                VARCHAR2(4000),
    PROCESS_VAL_DATE                TIMESTAMP,
    PROCESS_VAL_EFF                 NUMBER,
    LINE_OPERATORS_TRAINING         NUMBER,
    LINE_OPERATORS_TRAINING_DATE    TIMESTAMP,
```

```
36      SERVICE_DEPT_TRAINING          NUMBER,
37      SERVICE_DEPT_TRAINING_DATE     TIMESTAMP,
38      -- VSL part
39      FINAL_VAL                      NUMBER,
40      FINAL_VAL_AFTER_ACTION_TO_DO   NUMBER,
41      FINAL_VAL_NOTE                 VARCHAR2(4000),
42      FINAL_VAL_DATE                 TIMESTAMP,
43      ACTION_TO_DO_REQUEST_DATE      TIMESTAMP,
44      ACTION_TO_DO_NOTE              VARCHAR2(4000),
45      PRIMARY KEY (ID)
46    );
47  -- reference HEADER/PETITIONER
48  ALTER TABLE CAREL_PRESERIES_H ADD CONSTRAINT CAREL_PRESERIES_H_U_FK
49    FOREIGN KEY(CREATED_BY) REFERENCES JUSER(USER_ID) ON DELETE CASCADE;
50  -- reference HEADER/OWNER
51  ALTER TABLE CAREL_PRESERIES_H ADD CONSTRAINT CAREL_PRESERIES_H_O_FK
52    FOREIGN KEY (OWNER) REFERENCES JUSER(USER_ID) ON DELETE CASCADE;
53  -- reference HEADER/PLANNING
54  ALTER TABLE CAREL_PRESERIES_H ADD CONSTRAINT CAREL_PRESERIES_H_P_FK
55    FOREIGN KEY (PLANNING_REFERENT) REFERENCES JUSER(USER_ID) ON DELETE CASCADE;
56  -- reference HEADER/VSL
57  ALTER TABLE CAREL_PRESERIES_H ADD CONSTRAINT CAREL_PRESERIES_H_V_FK
58    FOREIGN KEY (VSL) REFERENCES JUSER(USER_ID) ON DELETE CASCADE;
59
60
61  CREATE TABLE CAREL_PRESERIES_L(
62      ID                       NUMBER NOT NULL,
63      HEADER_ID                NUMBER NOT NULL,
64      ITEM_CODE                VARCHAR2(40),
65      ITEM_DESCRIPTION         VARCHAR2(4000),
66      ITEM_ID                  NUMBER,
67      QUANTITY                 NUMBER,
68      ODL                      NUMBER,
69      OWNER_PRESENCE_REQUIRED  NUMBER,
70      PLANNED_DATE             TIMESTAMP,
71      NEW_PLANNED_DATE         TIMESTAMP,
72      PRODUCTION_DATE          TIMESTAMP,
73      PRIMARY KEY (ID)
74    );
75  -- reference LINES/HEADER
76  ALTER TABLE CAREL_PRESERIES_L ADD CONSTRAINT CAREL_PRESERIES_L_H_FK
```

```
77    FOREIGN KEY (HEADER_ID) REFERENCES CAREL_PRESERIES_H(ID) ON DELETE CASCADE;
78
79
80  CREATE TABLE CAREL_PRESERIES_MATERIAL(
81      ID                NUMBER NOT NULL,
82      HEADER_ID         NUMBER NOT NULL,
83      ITEM_CODE         VARCHAR2(40),
84      ITEM_DESCRIPTION  VARCHAR2(4000),
85      ITEM_ID           NUMBER,
86      AVAILABILITY_DATE TIMESTAMP,
87      PRIMARY KEY (ID)
88    );
89  -- reference MATERIAL/HEADER
90  ALTER TABLE CAREL_PRESERIES_MATERIAL ADD CONSTRAINT CAREL_PRESERIES_MATERIAL_H_FK
91    FOREIGN KEY (HEADER_ID) REFERENCES CAREL_PRESERIES_H(ID) ON DELETE CASCADE;
92
93
94  CREATE TABLE CAREL_PRESERIES_CHECK_L(
95      ID          NUMBER NOT NULL,
96      HEADER_ID   NUMBER NOT NULL,
97      CHECKLIST_ID   NUMBER NOT NULL,
98      STATUS      NUMBER NOT NULL,
99      PRIMARY KEY (ID)
100   );
101 -- reference CHECKLIST/HEADER
102 ALTER TABLE CAREL_PRESERIES_CHECK_L ADD CONSTRAINT CAREL_PRESERIES_CHECK_L_H_FK
103   FOREIGN KEY (HEADER_ID) REFERENCES CAREL_PRESERIES_H(ID) ON DELETE CASCADE;
104 ALTER TABLE CAREL_PRESERIES_CHECK_L ADD CONSTRAINT CAREL_PRESERIES_CHECK_L_CL_FK
105   FOREIGN KEY (CHECKLIST_ID) REFERENCES CAREL_PRESERIES_CHECKLIST(ID)
106   ON DELETE CASCADE;
107
108 CREATE TABLE CAREL_PRESERIES_CHECKLIST(
109     ID              NUMBER NOT NULL,
110     TYPE            NUMBER NOT NULL, -- MANDATORY OR NOT
111     DESCRIPTION_IT VARCHAR2(4000),
112     DESCRIPTION_EN VARCHAR2(4000),
113     LINK            VARCHAR2(1000),
114     PRIMARY KEY (ID)
115   );
```

Capitolo 4

La struttura dell'applicazione web e le attività preliminari

In questo capitolo verrà presentata com'è strutturata l'applicazione web e le attività che sono state eseguite preliminarmente rispetto all'implementazione delle classi componenti lo schema *Model-View-Controller* (MVC). Si partirà analizzando come sono stati gestiti i profili utente, per poi soffermarsi sulla gestione della navigazione fra le pagine componenti l'interfaccia. In finale verranno esaminati i file di configurazione necessari per il funzionamento dell'applicazione web.

4.1 Struttura dell'applicazione web

L'applicazione web è stata sviluppata utilizzando *JDeveloper*, che è un IDE (*Integrated Development Environment*) distribuito gratuitamente da *Oracle Corporation*. La struttura è organizzata secondo il *pattern* MVC. In figura 4.1 è visibile la struttura globale, che è così composta:

- nel *CommonsViewController* sono fornite una serie di classi basilari che coprono operazioni comuni ad ogni applicazione del portale. Le classi specifiche di una certa applicazione andranno quindi ad estendere quelle basilari;

- nel *Model* sono contenute le classi che si occupano della parte di business dell'applicazione;

- nel *ViewController* sono contenute le classi che svolgono l'operazione di presentazione dei dati all'utente e quelle che si occupano della comunicazione fra il *model* e la *view*.

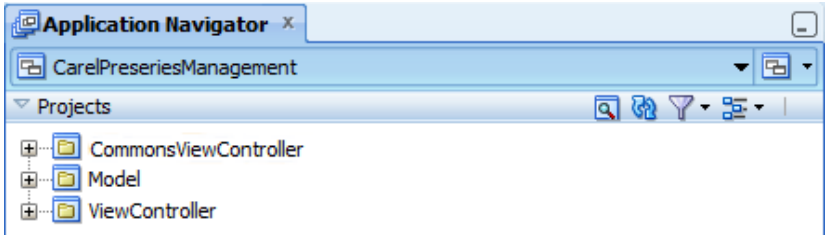


Figura 4.1: Struttura globale dell'applicazione web

In figura 4.2 sono raffigurate le classi che fanno parte del *model*, che sono:

- *CarelPreseriesSessioFacadeLocal* è l'interfaccia del *session bean*, gestisce la logica applicativa;
- *CarelPreseriesSessioFacadeBean* è la classe in cui è implementato il *session bean*;
- *CarelPreseriesCheckL* è un *entity bean* che si occupa della rappresentazione dei dati della tabella "CAREL_PRESERIES_CHECK_L";
- *CarelPreseriesChecklist* è un *entity bean* che si occupa della rappresentazione dei dati della tabella "CAREL_PRESERIES_CHECKLIST", questa è una tabella in cui sono state inserite tutte le possibili voci presentabili nella *check list*;
- *CarelPreseriesH* è un *entity bean* che si occupa della rappresentazione dei dati della tabella "CAREL_PRESERIES_H";
- *CarelPreseriesL* è un *entity bean* che si occupa della rappresentazione dei dati della tabella "CAREL_PRESERIES_L";
- *CarelPreseriesMaterial* è un *entity bean* che si occupa della rappresentazione dei dati della tabella "CAREL_PRESERIES_MATERIAL";

E' inoltre presente il file di configurazione *persistence.xml* in cui è stata configurata l'unità di persistenza.

In figura 4.3 sono presenti le classi che compongono la parte di *Controller*, che sono organizzate nei seguenti pacchetti:

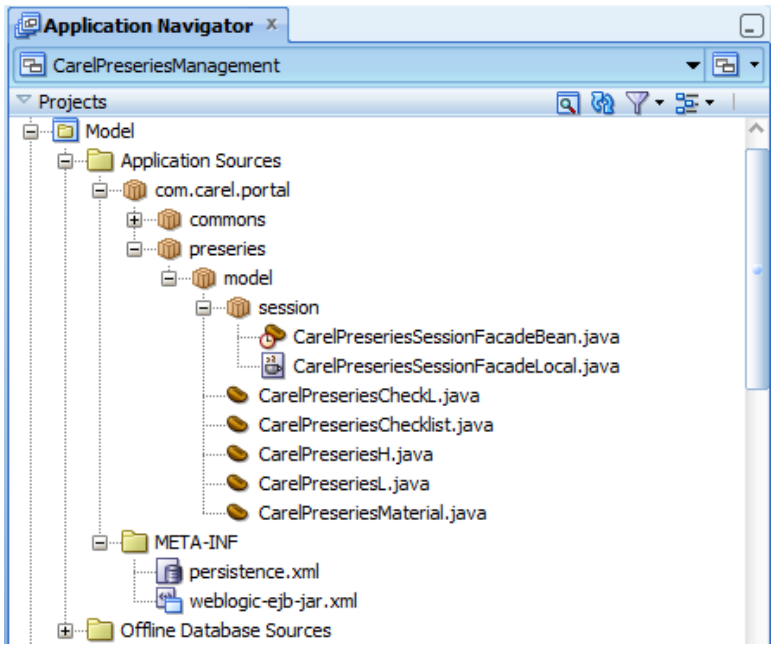


Figura 4.2: Composizione della parte *model*

- *comparators*, contiene classi che servono a confrontare due oggetti rendendone possibile l'ordinamento; i comparatori implementati hanno lo scopo di creare un ordinamento fra le richieste di produzione di preserie/prototipi basato sul loro *id*;
- *converters*, contiene le classi che convertono il valore di un componente UI in una stringa per poterla visualizzare;
- *navigation*, contiene le classi che si occupano della navigazione fra le pagine;
- *scheduledjobs*, contiene la classe che si occupa di schedulare l'invio delle e-mail di promemoria;
- *user*, contiene la classe che si occupa di raccogliere tutte le informazioni riguardanti l'utente, come ad esempio la lingua o il profilo di appartenenza;
- *util*, contiene le classi che si occupano di inviare le e-mail;
- *view*, contiene i *backing bean* delle pagine dell'applicazione web, una classe che si occupa di inizializzare tutte le liste di cui si necessiterà (per esempio la lista degli utenti aventi un certo profilo) e le due risorse contenenti una tutte le frasi in lingua italiana e l'altra le rispettive traduzioni in inglese;

- *wrapper*, contiene classi che servono a legare gli *entity bean* ad altre proprietà nel momento in cui sono visualizzati in forma tabellare.

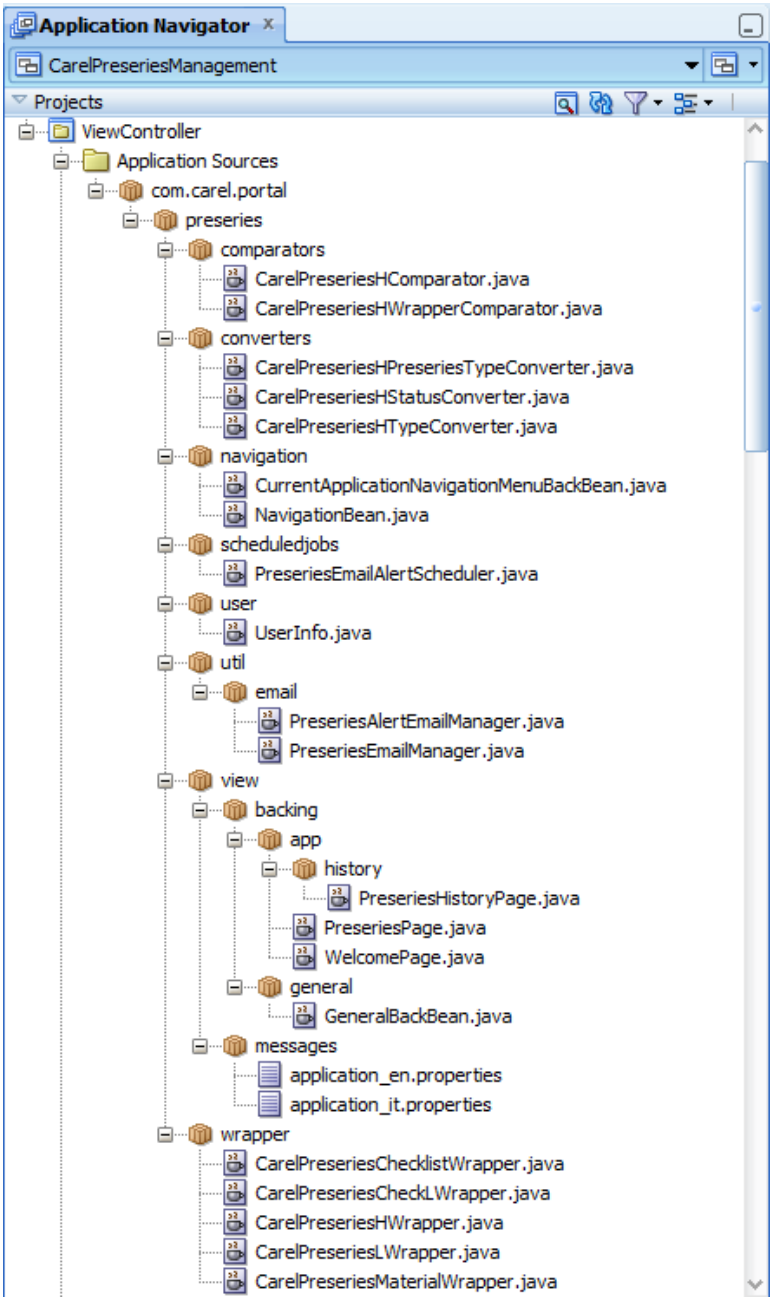


Figura 4.3: Composizione della parte *controller*

In figura 4.4 sono presenti le parti che compongono la *view*, d'interesse sono:

- le tre pagine *presenieshistory, preseries, welcome* che sono utilizzate dall'applicazione web;

- il foglio di stile *style.css*;
- i componenti personalizzati dell'azienda *Carel*;
- il file di configurazione dell'applicazione, *web.xml*;
- il file di configurazione del *framework* JSF, *faces-config.xml*.

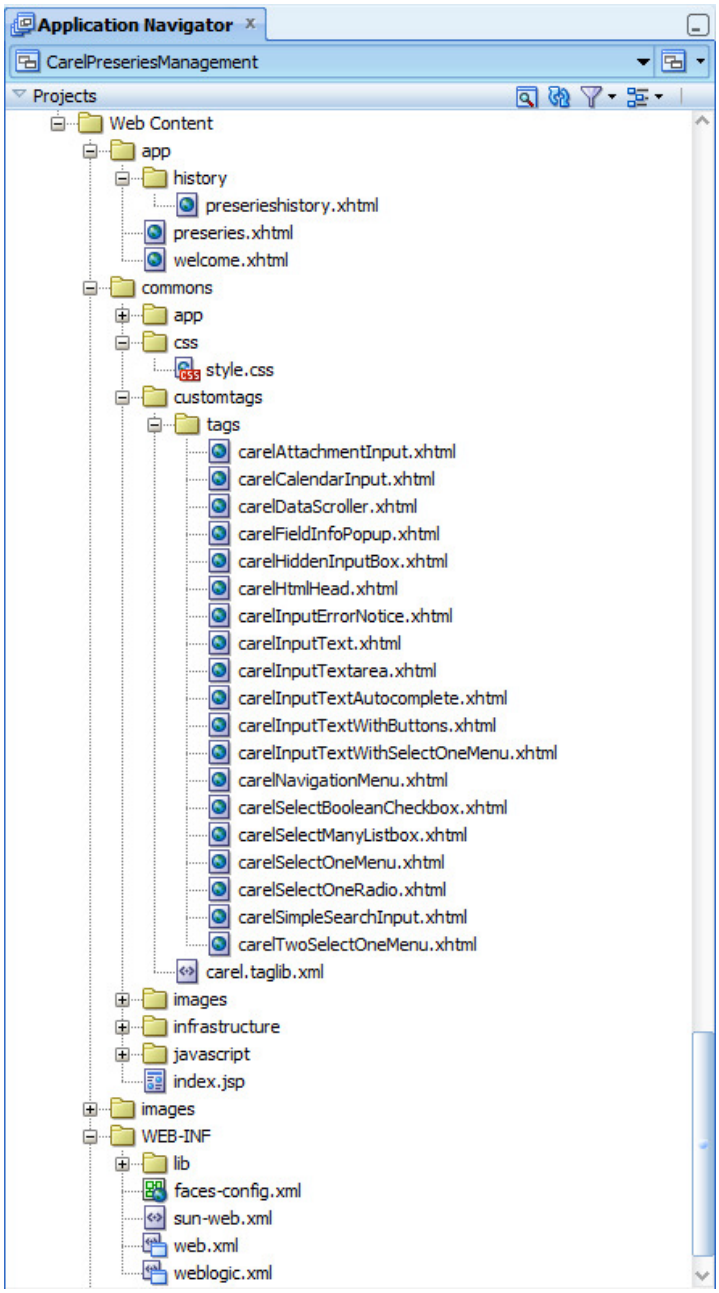


Figura 4.4: Composizione della parte *view*

4.2 La gestione dei profili utente

Sono stati utilizzati tre profili:

- *user*, utilizzato dal personale *Carel* designato per poter essere il richiedente del processo o l'*owner*;
- *planning*, utilizzato dal personale del reparto *Operations* addetto alla pianificazione degli ODL sulle linee produttive;
- *valueStreamLeader*, utilizzato dal personale del reparto *Plant* che si occupa delle linee produttive.

Questi sono stati dichiarati all'interno della classe *PreseriesManagementApplicationCostants* facente parte del *CommonsViewController*, che contiene le costanti dichiarate per l'applicazione: cioè un *id* che identifica univocamente l'applicazione all'interno del portale, la stringa utilizzata per identificare il *session bean* di questa applicazione e le stringhe utilizzate per identificare i profili utente.

Listing 4.1: Dichiarazioni delle costanti identificanti i profili utente

```
1    public static final String ROLE_USER = "user";
2    public static final String ROLE_V = "valueStreamLeader";
3    public static final String ROLE_PLANNING = "planning";
```

Il profilo *user* può eseguire le seguenti operazioni:

- creazione di una nuova richiesta di produzione di una preserie/prototipo;
- essere il responsabile di una richiesta;
- fare ricerche nello storico delle richieste.

Un utente avente profilo *planning* o *valueStreamLeader* può eseguire tutte le operazioni del profilo *user*. Inoltre il primo può inserire i dati riguardanti la pianificazione della produzione di una certa richiesta, mentre il secondo può eseguire la validazione finale di una preserie.

L'identificazione del profilo di appartenenza avviene all'interno della classe *UserInfo* ed il risultato è facilmente ottenibile attraverso dei metodi *get*. Per far ciò si sono utilizzate delle *query* fornite dal *session bean* del portale, dato che è un'operazione comune a tutte le applicazioni.

4.3 La gestione della navigazione

L'applicazione è composta dalle seguenti pagine:

- *welcome.xhtml*, è la pagina di benvenuto da cui è possibile accedere alle pagine responsabili della funzione desiderata;
- *preserieshistory.xhtml*, in questa pagina viene gestita sia la ricerca nello storico che la visualizzazione delle richieste pendenti. Entrambe le operazioni sono implementate in un unico *file* tramite l'utilizzo dei *rendered*. Questo ha lo scopo di utilizzare il minor numero di *file*, sui quali potrebbe essere necessario far manutenzione in un secondo momento;
- *preseries.xhtml*, in questa pagina è possibile creare una nuova richiesta, visualizzare le richieste (pendenti o concluse) ed avanzare nel flusso del processo. Anche questa pagina sfrutta molto l'utilizzo dei *rendered*.

Come mostrato in figura 4.5, dalla pagina di benvenuto *welcome.xhtml* è possibile accedere ad entrambe le altre pagine. Questo utilizzando il menù presente in alto a tutte le pagine, rappresentato in figura 4.6. Questo permette inoltre un facile spostamento tra le pagine *preserieshistory.xhtml* e *preseries.xhtml*.

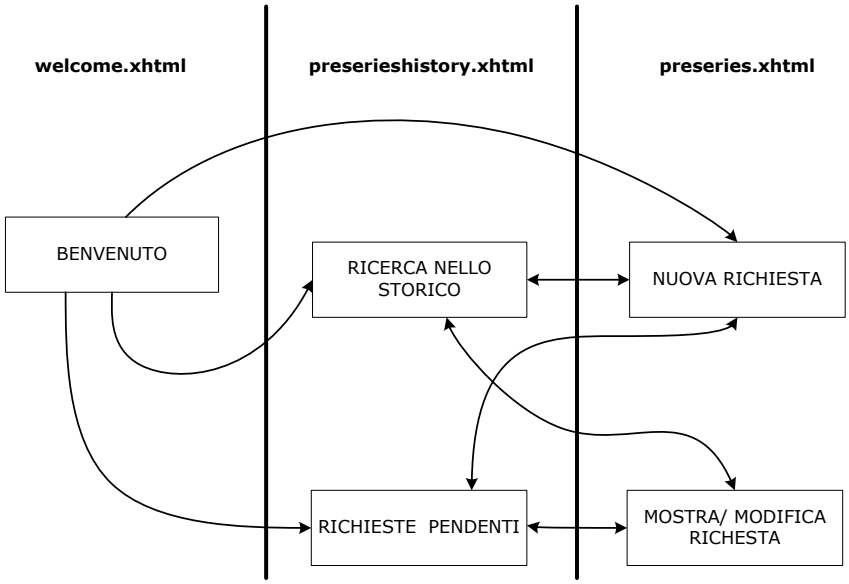


Figura 4.5: Suddivisione delle operazioni fra le pagine e flusso della navigazione

Figura 4.6: Menù di navigazione

La gestione della navigazione in JSF avviene tramite eventi. Nell'estratto del *file* di configurazione *faces-config.xml* presente nel listato 4.2 sono dichiarati gli eventi che comportano il cambiamento di pagina e la pagina di destinazione. Più precisamente, ogni qualvolta venga lanciato l'evento *preseries* si finirà in *preseries.xhtml*, mentre, se viene lanciato l'evento *preserieshistory* si finirà in *preserieshistory.xhtml*.

Listing 4.2: Regole di navigazione dichiarate nel *file faces-config.xml*

```
1  <navigation-rule>
2    <navigation-case>
3      <from-outcome>preseries</from-outcome>
4      <to-view-id>/app/preseries.xhtml</to-view-id>
5      <redirect/>
6    </navigation-case>
7    <navigation-case>
8      <from-outcome>preserieshistory</from-outcome>
9      <to-view-id>/app/history/preserieshistory.xhtml</to-view-id>
10     <redirect/>
11   </navigation-case>
12 </navigation-rule>
```

Questo però non è sufficiente. Infatti ogni pagina può avere comportamenti diversi grazie all'uso dei *rendered*. Per la gestione di questi comportamenti sono state utilizzate due classi, contenute nel pacchetto *navigation*:

- *NavigationBean* si occupa di inizializzare la pagina per la funzione desiderata e restituire la stringa corrispondente all'evento da lanciare per la sua visualizzazione;
- *CurrentApplicationNavigationMenùBackBean* si occupa di aggiornare le voci del menù principale;

Nella classe *NavigationBean* sono state dichiarate le seguenti quattro costanti:

Listing 4.3: Dichiarazione delle costanti rappresentanti la pagina da visualizzare

```
1 public static final String NAVIGATION_ACTION_PRESERIES = "preseriesAction";
```

```
2 public static final String NAVIGATION_ACTION_NEW_PRESERIES =
3         "newPreseriesAction";
4 public static final String NAVIGATION_ACTION_PRESERIES_HISTORY =
5         "preseriesHistoryAction";
6 public static final String NAVIGATION_ACTION_MY_PENDING_PRESERIES =
7         "myPendingPreseriesAction";
```

Queste costanti rappresentano le quattro modalità di visualizzazione delle due pagine *preseries.xhtml* e *presereishistory.xhtml*, corrispondenti alle operazioni mostrate in figura 4.5.

Come detto precedentemente per spostarsi da una pagina all'altra si utilizza il menù in figura 4.6, che ad esempio nella pagina *preseries.xhtml* è così dichiarato[1]:

Listing 4.4: Dichiarazione del menù nella pagina *preseries.xhtml*

```
1 <carel:navigationMenu rendered="#{PreseriesPage.showMainMenu}"
2         binding="#{PreseriesPage.mainMenuBackBean}"/>
```

Il menù viene creato utilizzando il componente personalizzato *navigationMenu*, il cui scopo è visualizzare le voci presenti nella lista *mainMenuBackBean* sottoforma di menù e reindirizzare la navigazione alla pagina prescelta, lanciando l'evento necessario. Per far ciò ogni volta che viene visualizzato attraverso la proprietà *rendered* invoca il metodo *getShowMainMenu()* del *back bean PreseriesPage*, che fa eseguire un aggiornamento come mostrato nel listato seguente.

Listing 4.5: Metodi responsabili dell'aggiornamento delle voci del menù

```
1  public boolean getShowMainMenu() {
2      updateMainMenuBackBean();
3      return true;
4  }
5
6  public void updateMainMenuBackBean() {
7      if (userSessionBean == null)
8          userSessionBean =
9                  (UserInfo)getSessionBean(PortalConstants.USER_INFO_SESSION_BEAN);
10     mainMenuBackBean.clearMenuBackBean();
11     mainMenuBackBean.addMenuItem(
12           NavigationBean.NAVIGATION_ACTION_PRESERIES_HISTORY);
13     mainMenuBackBean.addMenuItem(
```

[1]La dichiarazione nella pagina *presereishistory.xhtml* è analoga, farà però riferimento ai metodi aventi la stessa firma ma appartenenti al *back bean PreseriesHistoryPage*

```
14              NavigationBean.NAVIGATION_ACTION_MY_PENDING_PRESERIES);
15      mainMenuBackBean.addMenuItem(
16              NavigationBean.NAVIGATION_ACTION_NEW_PRESERIES);
17  }
```

Per eseguire l'aggiornamento si fa uso della classe *CurrentApplicationNavigationMenùBackBean* attraverso la sua istanza chiamata *mainMenuBackBean*. Che aggiungerà la voce al menù soltanto se l'utente loggato ha i privilegi necessari per poterla visualizzare, cioè se l'utente possiede almeno uno dei tre profili disponibili per quest'applicazione.

Per quanto riguarda invece il reindirizzamento il componente personalizzato *navigationMenu* invoca il metodo *redirect()* del *back bean PreseriesPage* per ottenere la stringa corrispondente all'evento da lanciare. Per far ciò, una volta che l'utente preme una voce del menù, il componente esegue il salvataggio del corrispondente parametro, il quale potrà essere una delle quattro costanti presenti nel listato 4.3. Il metodo *redirect()* si occupa quindi di recuperare il parametro per poi invocare il metodo *getNavigation(String action)* della classe *NavigationBean*.

Listing 4.6: Metodo responsabile di ottenere la stringa rappresentante l'evento da invocare

```
1  public String redirect() {
2      clearBackBean();
3      //Get the navigation type parameter
4      String redirectTo =
5        (String)getContextParameter(PortalConstants.REDIRECT_TO_CONTEXT_PARAMETER);
6      return (new NavigationBean()).getNavigation(redirectTo.trim());
7  }
```

Il metodo *getNavigation(String action)* ha il compito di identificare la funzione alla quale si vuole accedere e di conseguenza smistare la richiesta.

Listing 4.7: Smistamento della richiesta a seconda della funzione richiesta

```
1  public String getNavigation(String action) {
2      if (isNavigationBase(action))
3          return getNavigationBase(action);
4      else if (action.equals(NAVIGATION_ACTION_PRESERIES))
5          return preseriesAction();
6      else if (action.equals(NAVIGATION_ACTION_PRESERIES_HISTORY))
7          return preseriesHistoryAction();
8      else if (action.equals(NAVIGATION_ACTION_NEW_PRESERIES))
```

```
9           return newPreseriesAction();
10      else if (action.equals(NAVIGATION_ACTION_MY_PENDING_PRESERIES))
11          return myPendingPreseriesAction();
12      else
13          return "";
14  }
```

Nel listato 4.8 sono mostrati i metodi che si occupano di effettuare le inizializzazioni necessarie e poi restituire la stringa rappresentante l'evento da lanciare per poter andare nella pagina desiderata.

Listing 4.8: Dichiarazione del menù nella pagina *preseries.xhtml*

```
1  public String newPreseriesAction() {
2    PreseriesPage page =
3        (PreseriesPage)PageBackBeanBase.getSessionBean("PreseriesPage");
4    page.clearBackBean();
5    return "preseries";
6  }
7
8  public String preseriesAction() {
9    return "preseries";
10 }
11
12 public String preseriesHistoryAction() {
13   PreseriesHistoryPage page =
14    (PreseriesHistoryPage)PageBackBeanBase.getSessionBean("PreseriesHistoryPage");
15   page.clearBackBean();
16   page.setViewType(PreseriesHistoryPage.VIEW_TYPE_HISTORY);
17   return "preserieshistory";
18 }
19
20 public String myPendingPreseriesAction() {
21   PreseriesHistoryPage page =
22    (PreseriesHistoryPage)PageBackBeanBase.getSessionBean("PreseriesHistoryPage");
23   page.init();
24   page.setViewType(PreseriesHistoryPage.VIEW_TYPE_PENDING);
25   return "preserieshistory";
26 }
```

4.4 I file di configurazione

Nell'applicazione sono presenti tre file di configurazione importanti:

- *web.xml* è il file di configurazione dell'applicazione web;
- *persistence.xml* in cui è definita l'unità di persistenza;
- *faces-config.xml* è il file di configurazione del *framework* JSF.

Nel prima parte del file *web.xml*, presente nel listato seguente, è stato definito il nome dell'applicazione ed il metodo di salvataggio delle informazioni riguardanti lo stato, che verranno salvate al lato *client*. Questa scelta è dovuta alle limitate capacità del server che si occupa della gestione del portale aziendale.

Listing 4.9: Parte iniziale del file di configurazione *web.xml*

```
1     <description>Preseries Management</description>
2     <context-param>
3         <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
4         <param-value>client</param-value>
5     </context-param>
```

A seguito sono mostrati i parametri di configurazione della tecnologia di visualizzazione *Facelets* che sono: il tipo di *file* utilizzato, l'indicazione a non riportare i commenti, il *file carel.taglib.xml* dove sono indicati tutti i componenti personalizzati utilizzati, la classe in cui è implementato il *FacesServlet* e l'URL su cui mappare le pagine.

Listing 4.10: Parte riguardante *Facelets* del file di configurazione *web.xml*

```
6     <context-param>
7         <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
8         <param-value>.xhtml</param-value>
9     </context-param>
10    <context-param>
11        <param-name>facelets.SKIP_COMMENTS</param-name>
12        <param-value>true</param-value>
13    </context-param>
14    <context-param>
15        <param-name>facelets.LIBRARIES</param-name>
16        <param-value>/commons/customtags/carel.taglib.xml</param-value>
17    </context-param>
```

```
18      <servlet>
19          <servlet-name>Faces Servlet</servlet-name>
20          <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
21          <load-on-startup>1</load-on-startup>
22      </servlet>
23      <servlet-mapping>
24          <servlet-name>Faces Servlet</servlet-name>
25          <url-pattern>/faces/*</url-pattern>
26      </servlet-mapping>
```

Nella parte seguente è invece stato configurato il *timeout* di una sessione e la pagina iniziale, che è quella in cui avviene il login al portale.

Listing 4.11: Parte riguardante JSF del file di configurazione *web.xml*

```
6       <session-config>
7           <session-timeout>30</session-timeout>
8       </session-config>
9       <welcome-file-list>
10          <welcome-file>/commons/index.jsp</welcome-file>
11      </welcome-file-list>
```

Di seguito sono presenti le configurazioni dei *session bean* utilizzati. Sono tutti locali, cioè risiedono all'interno dello stesso *container* degli altri *bean* facenti parte dell'applicazione. I primi due sono implementati nella parte comune a tutte le applicazioni: il primo si occupa di recuperare informazioni riguardanti l'utente mentre il secondo quelle riguardanti l'inventario. In ultima la *CarelPreseriesSessionFacade* è quella utilizzata per le operazioni riguardanti l'applicazione implementata.

Listing 4.12: Configurazione dei *session bean* nel file di configurazione *web.xml*

```
42      <ejb-local-ref>
43          <ejb-ref-name>ejb/JUserInfoSessionFacade</ejb-ref-name>
44          <ejb-ref-type>Session</ejb-ref-type>
45          <local>
46      com.carel.portal.commons.model.portalinfo.session.JUserInfoSessionFacadeLocal
47          </local>
48          <ejb-link>JUserInfoSessionFacadeBean</ejb-link>
49      </ejb-local-ref>
50
51      <ejb-local-ref>
52          <ejb-ref-name>ejb/OappsSessionFacade</ejb-ref-name>
```

```
53          <ejb-ref-type>Session</ejb-ref-type>
54          <local>
55      com.carel.portal.commons.model.oapps.session.OappsSessionFacadeLocal
56          </local>
57          <ejb-link>OappsSessionFacadeBean</ejb-link>
58      </ejb-local-ref>
59
60      <ejb-local-ref>
61          <ejb-ref-name>ejb/CarelPreseriesSessionFacade</ejb-ref-name>
62          <ejb-ref-type>Session</ejb-ref-type>
63          <local>
64      com.carel.portal.preseries.model.session.CarelPreseriesSessionFacadeLocal
65          </local>
66          <ejb-link>CarelPreseriesSessionFacadeBean</ejb-link>
67      </ejb-local-ref>
```

Nel file sono presenti anche altre configurazioni che non sono state riportate riguardanti il caricamento dei file tramite l'applicazione, il metodo di autenticazione e le configurazioni necessarie per l'invio delle e-mail.

Il listato seguente contiene il file di configurazione *persistence.xml*. Un'unità di persistenza (*persistence unit*) definisce l'insieme di tutte le classi *Entity* gestite dall'*EntityManager* in un'applicazione. In altre parole, rappresenta l'insieme dei dati, di interesse per quell'applicazione, contenuto in un *data-store* singolo. L'elemento "jta-data-source" specifica il nome JNDI (*Java Naming and Directory Interface*) globale della sorgente dati che deve essere utilizzata dal container. *EclipseLink* è l'implementazioni della specifica JPA utilizzata dal portale, è un progetto *open source* della *Eclipse Foundation*.

Listing 4.13: File di configurazione *persistence.xml*

```
1  <persistence>
2    <persistence-unit name="Portal">
3    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
4      <jta-data-source>java:/app/jdbc/jdbc/XXCAR_PORTAL_RACTDS</jta-data-source>
5      <properties>
6        <property name="eclipselink.target-server" value="WebLogic_10"/>
7        <property name="javax.persistence.jtaDataSource"
8      value="java:/app/jdbc/jdbc/XXCAR_PORTAL_RACTDS"/>
9      </properties>
10   </persistence-unit>
11 </persistence>
```

Si esaminerà ora il file di configurazione *faces-config.xml*, il cui scopo è di configurare il *framework* JSF. Nel listato seguente è dichiarato l'utilizzo della tecnologia di visualizzazione *Facelets*.

Listing 4.14: Estratto del file di configurazione *faces-config.xml*

```
1    <view-handler>com.sun.facelets.FaceletViewHandler</view-handler>
```

L'elemento "managed-bean" definisce la configurazione di un *bean* che ha il nome definito dalla proprietà "managed-bean-name", è istanza della classe definita dalla proprietà "managed-bean-class" e ha *scope* (visibilità) definito dalla proprietà "managed-bean-scope".

Listing 4.15: Estratto del file di configurazione *faces-config.xml*

```
2   <managed-bean>
3     <managed-bean-name>WelcomePage</managed-bean-name>
4     <managed-bean-class>com.carel.portal.preseries.view.backing.app.WelcomePage
5     </managed-bean-class>
6     <managed-bean-scope>request</managed-bean-scope>
7   </managed-bean>
```

Tutti i *managed bean* dichiarati all'interno del file di configurazione e le loro visibilità sono riportati in tabella 4.1.

Tabella 4.1: Tabella dei *backing bean*

Nome	**Visibilità**
WelcomePage	*request*
PopUpPage	*request*
PreseriesPage	*session*
PreseriesHistoryPage	*session*
InitialPage	*request*
UserInfo	*session*
GeneralBackBean	*session*
PreseriesEmailAlertScheduler	*application*

Nel file di configurazione sono presenti anche la parte di configurazione della navigazione affrontata nella sezione 4.3, la parte di configurazione delle risorse

che verrà affrontata nella sezione 5.3.1 e la parte di dichiarazione dei convertitori che verrà affrontata nella sezione 5.3.2.

Capitolo 5

Implementazione dei componenti dello schema MVC

In questo capitolo si andranno ad affrontare singolarmente le classi ed i *file* che compongono lo schema MVC dell'applicazione web. Il capitolo è suddiviso in modo tale da trattare uno alla volta il *model*, la *view* ed il *controller*.

5.1 Il *Model*

In questa sezione verranno affrontate tutti i componenti del *Model*, cioè le classi che si occupano della parte di business dell'applicazione.

5.1.1 Gli *entity bean*

Come già visto precedentemente il *Model* contiene i seguenti *entity bean*: *CarelPreseriesCheckL*, *CarelPreseriesChecklist*, *CarelPreseriesH*, *CarelPreseriesL* e *CarelPreseriesMaterial*. Ognuno di essi rappresenta una tabella della base di dati relazionale e una loro istanza corrisponde a una riga in tale tabella.
L'ORM è stato effettuato con l'utilizzo delle annotazioni all'interno delle classi. I requisiti di una classe *entity* sono:

- avere l'annotazione "javax.persistence.Entity";
- avere un costruttore senza argomenti, *public* o *protected*;
- nessun metodo o variabile di istanza persistente deve essere dichiarata costante;

- ogni entità deve avere un identificatore unico di oggetto, la chiave primaria di una entità può essere semplice o composta;
- la chiave primaria semplice deve essere indicata attraverso l'annotazione "javax.persistence.Id".

Ora si andrà ad analizzare uno degli *entity bean* creati: *CarelPreseriesL*. Come prima cosa è stata inserita l'annotazione "javax.persistence.Entity" per indicare che la classe rappresenta un *entity bean*. Di seguito sono state utilizzate le JPA *Query* API, che supportano due tipi di *query*:

- *named*, sono tipicamente utilizzate per eseguire le operazioni più comuni, vengono infatti memorizzate e riutilizzate quando è necessario;
- *dynamic*, sono create secondo le necessità applicative del momento.

Quella presente nel seguente listato è una *named* query che ha lo scopo di recuperare tutte i record della tabella "CAREL_PRESERIES_L". In finale è indicato il nome della tabella di riferimento dato che non è uguale a quello dell'*entity bean*, a causa del carattere trattino basso che non è utilizzato nel nome della classe.

Listing 5.1: Classe *CarelPreseriesL*

```
1 @Entity
2 @NamedQueries({
3   @NamedQuery(name = "CarelPreseriesL.findAll",
4           query = "select o from CarelPreseriesL o")
5 })
6 @Table(name = "CAREL_PRESERIES_L")
```

Successivamente sono state dichiarate le variabili, ognuna delle quali rappresenta un attributo della tabella. Per ogni variabile è presente un metodo *get* e *set*;

Listing 5.2: Classe *CarelPreseriesL*

```
1 public class CarelPreseriesL implements Serializable {
2   private Long id;
3   private String itemCode;
4   private String itemDescription;
5   private Long itemId;
6   private Long odl;
7   private Long ownerPresenceRequired;
8   private Long quantity;
```

```
9   private Timestamp plannedDate;
10  private Timestamp newPlannedDate;
11  private Timestamp productionDate;
12  private CarelPreseriesH carelPreseriesH;
```

Poi si è definita una costante che verrà utilizzata per eseguire controlli prima della memorizzazione dell'*entity bean*.

Listing 5.3: Classe *CarelPreseriesL*

```
1   public static final long MAX_QUANTITY = 50;
```

E' stato creato il costruttore pubblico senza argomenti.

Listing 5.4: Classe *CarelPreseriesL*

```
1   public CarelPreseriesL() {
2   }
```

E' stata specificata la chiave primaria della tabella e il vincolo che non può essere uguale a *NULL*.

Listing 5.5: Classe *CarelPreseriesL*

```
1   @Id
2   @Column(nullable = false)
3   public Long getId() {
4     return id;
5   }
```

Tramite l'annotazione "import javax.persistence.ManyToOne" è possibile legare questo *entity bean* al *CarelPreseriesH*.

Listing 5.6: Classe *CarelPreseriesL*

```
1   @ManyToOne
2   @JoinColumn(name = "HEADER_ID")
```

Tutti gli altri *entity bean* sono stati creati in maniera analoga.

5.1.2 L'interfaccia del *session bean*

Per quanto riguarda il *session bean* l'interfaccia è stata creata nella classe *CarelPreseriesSessioFacadeLocal*. L'interfaccia è di tipo locale, come visibile nel seguente listato.

Listing 5.7: Interfaccia del *session bean CarelPreseriesSessioFacadeLocal*

```
1 @Local
2 public interface CarelPreseriesSessionFacadeLocal extends Serializable {
```

Nel listato seguente sono mostrate le dichiarazione dei metodi fondamentali da implementare:

- il metodo *mergeEntity(Object entity)* aggiorna il database con i dati memorizzati nell'entità;
- il metodo *refreshEntity(Object entity)* aggiorna l'entità con i dati memorizzati nel database;
- Il metodo *persistEntity(Object entity)* crea un nuovo record nel database per l'entità;
- il metodo *removeEntity(Object entity)* rimuove un'entità dal database.

Listing 5.8: Metodi dichiarati nell'interfaccia *CarelPreseriesSessioFacadeLocal*

```
3     Object mergeEntity(Object entity);
4     Object persistEntity(Object entity);
5     Object refreshEntity(Object entity);
6     void removeEntity(Object entity);
```

Sono poi presenti le dichiarazione dei metodi il cui è scopo è ottenere il prossimo *id* disponibile per poter creare un nuovo record nel *database*, come mostrato qui di seguito.

Listing 5.9: Metodi dichiarati nell'interfaccia *CarelPreseriesSessioFacadeLocal*

```
7     long getNextAvailableCarelPreseriesCheckLId();
8     long getNextAvailableCarelPreseriesChecklistId();
9     long getNextAvailableCarelPreseriesHId();
10    long getNextAvailableCarelPreseriesLId();
11    long getNextAvailableCarelPreseriesMaterialId();
```

In finale sono mostrate le dichiarazioni degli altri metodi facenti parte dell'interfaccia *CarelPreseriesSessioFacadeLocal*. Il primo serve a recuperare tutti i record presenti in tabella "CAREL_PRESERIES_H" aventi un certo *id*. Il secondo serve a recuperare tutti i record presenti in tabella "CAREL_PRESERIES_CHECK_LIST". Il terzo ed il quarto metodo servono a recuperare le richieste pendenti a seconda

del profilo dell'utente loggato. L'ultimo metodo è quello utilizzato per effettuare ricerche nello storico, secondo parametri configurati dall'utente.

Listing 5.10: Metodi dichiarati nell'interfaccia *CarelPreseriesSessioFacadeLocal*

```
12      List<CarelPreseriesH> findAllCarelPreseriesHById(Long id);
13      List<CarelPreseriesChecklist> findAllCarelPreseriesChecklist();
14      List<CarelPreseriesH> findAllCarelPreseriesHPendingForPetitionerOrOwnerOrVsl(
15                              Long userId);
16      List<CarelPreseriesH> findAllCarelPreseriesHPendingForPlanning(Long userId);
17      List<CarelPreseriesH> findAllCarelPreseriesHByCustomQuery(String sqlQuery,
18                              List<String> parametersList);
```

5.1.3 Implementazione del *session bean*

Il *session bean* è implementato nella classe *CarelPreseriesSessioFacadeBean*. Come si può vedere nel listato seguente, nelle prime righe attraverso l'annotazione "javax.ejb.Stateless" è specificato che non mantiene informazioni riguardanti lo stato. L'annotazione "javax.ejb.TransactionManagement;" specifica che è il *container* ad occuparsi di iniziare la transazione ed effettuarne il *commit* ed il *rollback* al posto dello sviluppatore. Con l'annotazione "javax.ejb.Local" viene specificata la classe in cui è descritta l'interfaccia locale che verrà implementata.

Listing 5.11: Implementazione del *session bean CarelPreseriesSessioFacadeBean*

```
1  @Stateless
2  @TransactionManagement(value = TransactionManagementType.CONTAINER)
3  @Local(value = { CarelPreseriesSessionFacadeLocal.class })
4  public class CarelPreseriesSessionFacadeBean
5                              implements CarelPreseriesSessionFacadeLocal {
```

L'*EntityManager* è un'interfaccia delle JPA con il compito di costituire un ponte tra il mondo *Object-Oriented* e quello relazionale. L'utilizzo dell'annotazione "javax.persistence.PersistenceContext" consente di ottenere l'istanza di un *EntityManager container-managed*. Con l'elemento *unitName* si specifica il nome dell'unità di persistenza, quella definita nel file di configurazione *persistence.xml*.

Listing 5.12: Implementazione del *session bean CarelPreseriesSessioFacadeBean*

```
6   @PersistenceContext(unitName = "Portal")
7   private EntityManager em;
```

Come mostrato qui di seguito, attraverso l'*EntityManager* è possibile implementare semplicemente i quattro metodi fondamentali dichiarati nell'interfaccia.

Listing 5.13: Implementazione del *session bean CarelPreseriesSessioFacadeBean*

```
11 public Object mergeEntity(Object entity) {
12   return em.merge(entity);
13 }
14
15 public Object persistEntity(Object entity) {
16   em.persist(entity);
17   return entity;
18 }
19
20 public Object refreshEntity(Object entity) {
21   em.refresh(entity);
22   return entity;
23 }
24
25 public void removeEntity(Object entity) {
26   em.remove(em.merge(entity));
27 }
```

Nel listato seguente è mostrata una *dynamic query* che è utilizzata per ottenere il prossimo *id* disponibile. Questo utilizzando le sequenze fornite da *Oracle* per ottenere identificatori univoci.
Per fare ciò è creata un'istanza dell'oggetto *Query* mediante il metodo dell'*EntityManager createNativeQuery(String sqlString, Class resultClass)*. Questo crea un'istanza dell'oggetto *Query* a partire dalla stringa *sqlString* passata e dal tipo di classe risultato *resultClass*. Infine si ottiene il risultato mediante il metodo *getSingleResult()* della classe *Query*.

Listing 5.14: Implementazione del *session bean CarelPreseriesSessioFacadeBean*

```
52 public long getNextAvailableCarelPreseriesHId() {
53   Query query =
54    em.createNativeQuery("SELECT CAREL_PRESERIES_H_S.NEXTVAL FROM DUAL",
55                         SequenceResult.class);
56   return ((SequenceResult)query.getSingleResult()).getNextval();
57 }
```

I seguenti due metodi sono utilizzati per estrarre le richieste pendenti. Ce ne sono due poiché ciò avviene a seconda del profilo dell'utente: il primo metodo viene invocato nel caso l'utente non possieda il profilo *planning*, il secondo se lo possiede. Nel primo si ricercano le richieste ancora non completate, in cui l'utente loggato è il richiedente o l'*owner* o il VSL. Nel secondo si effettua la stessa *query* eseguita precedentemente a cui però vanno unite tutte le richieste che sono in attesa di essere pianificate.

Listing 5.15: Implementazione del *session bean CarelPreseriesSessioFacadeBean*

```
91  public List<CarelPreseriesH>
92              findAllCarelPreseriesHPendingForPetitionerOrOwnerOrVsl(Long userId) {
93    return em.createNativeQuery("select h.* from carel_preseries_h h
94           where (h.created_by = " + userId +
95           " or h.OWNER = " + userId +
96           " or h.VSL = " + userId +
97           ") and h.status <> " + CarelPreseriesH.STATUS_CLOSED,
98           CarelPreseriesH.class).getResultList();
99  }
100
101 public List<CarelPreseriesH> findAllCarelPreseriesHPendingForPlanning(
102                                                                  Long userId) {
103   return em.createNativeQuery("select h.* from carel_preseries_h h
104          where (h.created_by = " + userId +
105          " or h.OWNER = " + userId +
106          " or h.VSL = " + userId +
107          ") and h.status <> " + CarelPreseriesH.STATUS_CLOSED +
108          " union select h.* from carel_preseries_h h where
109          h.status = "+ CarelPreseriesH.STATUS_WAITING_PLANNING_APPROVAL,
110          CarelPreseriesH.class).getResultList();
111 }
```

L'ultimo metodo implementato è quello che si occupa della ricerca nello storico, secondo parametri specificati dall'utente. La *query string* viene creata all'interno del *backing bean* "PreseriesHistory" ed utilizza i parametri posizionali che sono indicati con il punto interrogativo seguito da un numero. Viene inoltre passata al metodo la lista, ordinata correttamente, dei parametri settati dall'utente. Quindi utilizzando il metodo *setParameter(int pos, Object value)* i parametri vengono settati all'interno della *query string*, permettendo di evitare attacchi di tipo SQL "Injection"[2].

Listing 5.16: Implementazione del *session bean CarelPreseriesSessioFacadeBean*

```
113 public List<CarelPreseriesH> findAllCarelPreseriesHByCustomQuery(String sqlQuery,
114                                                     List<String> parametersList) {
115   Query q = em.createNativeQuery(sqlQuery, CarelPreseriesH.class);
116   for (int i = 0; i < parametersList.size(); i++) {
117     // Positional parameters are numbered starting from 1
118     q.setParameter(i + 1, parametersList.get(i));
119   }
120   return q.getResultList();
121 }
```

5.2 La *View*

In questa sezione verranno esaminati i componenti della vista. Si partirà dando una breve spiegazione su cos'è un foglio di stile e perché è stato utilizzato. Si introdurranno poi i componenti personalizzati *Carel* utilizzati. In finale si andranno ad esaminare le tre pagine componenti l'applicazione web.

5.2.1 I fogli di stile

I CSS (*Cascading Style Sheet* - Fogli di stile a cascata) sono utilizzati per definire come una pagina deve essere visualizzata; mentre il codice HTML descrive il contenuto della pagina, i fogli di stile a cascata si occupano della sua formattazione (colori, caratteri, bordi, ...).
Esistono diversi vantaggi dovuti al loro utilizzo, principalmente:

- riutilizzo dello stesso tema per più pagine web senza bisogno di aggiunta di codice;
- separazione del codice relativo al contenuto da quello relativo alla rappresentazione grafica, facilitando quindi la manutenzione.

E' possibile creare stili a livello dell'intero documento o a livello di specifici *tag* dello stesso. Nel primo caso lo stile è definito all'interno dell'intestazione del documento (*tag head*), nel secondo ricorrendo all'attributo *style* per i singoli *tag*. E' importante considerare che il comportamento degli stili è "a cascata". In un documento possono essere utilizzati più stili: se un secondo stile viene definito

per un blocco di documento già trattato da un primo stile, tale porzione di pagina verrà visualizzata utilizzando il secondo. Lo stesso si verificherebbe per un terzo stile rispetto al secondo e via dicendo.
Le istruzioni usate nei fogli di stile sono dette regole di stile. Le regole sono scritte indicando il nome del *tag* seguito dall'elenco delle proprietà dello stile incluse tra parentesi graffe. Ogni proprietà di stile è individuato con il nome della proprietà, i due punti e il valore da attribuire alla proprietà. Nel caso di più proprietà le stesse sono separate dal carattere punto e virgola.
Le classi di stile sono un insieme di istruzioni di formattazione che possono essere applicate ad una pluralità di *tag* HTML. L'applicazione delle classi avviene aggiungendo esplicitamente la clausola *class* al *tag* di apertura dell'elemento HTML al quale applicarla e pertanto le classi non vengono applicate in modo automatico per ogni ricorrenza del *tag*.
Un'altra caratteristica delle classi è quella di poter essere anonime. Ciò significa che l'insieme degli elementi da alterare, previsti dallo stile, possono essere usati per alterare qualsiasi elemento (non si riferiscono ad un elemento particolare). In questo caso la sintassi non prevede l'indicazione di un *tag* ma semplicemente di un punto seguito dal nome della classe. Queste sono quelle che sono state utilizzate all'interno del file *style.css*, come è possibile osservare nell'estratto seguente.

Listing 5.17: Estratto dal foglio di stile *style.css*

```
1  .bodyEncloser {
2    width: 99.5%;
3    font-family: Verdana, Arial, Helvetica, sans-serif;
4    color: #707070;
5    font-size: 12px !important;
6    line-height: 15px !important;
7    margin-left: 0.25% !important;
8    margin-right:0.25% !important;
9    margin-top: 3px !important;
10 }
11
12 .alignLeft {
13   text-align: left;
14 }
15
16 .alignRight {
17   text-align: right;
```

```
18 }
19
20 .alignCenter {
21   text-align: center;
22   vertical-align: middle;
23 }
```

Per ottenere una grafica più accattivante è stato utilizzato anche *Twitter Bootstrap*[1] che è una collezione di strumenti per la creazione di siti e applicazione web.

5.2.2 I componenti personalizzati

L'utilizzo dei componenti personalizzati semplifica la creazione dell'interfaccia utente, dato che è possibile creare componenti complessi che poi possono essere riutilizzati. Inoltre rende più facile la manutenzione.
Nella tabella di seguito sono elencati i componenti personalizzati *Carel* e la loro funzione.

Tabella 5.1: Tabella dei componenti personalizzati *Carel*

Componente	Descrizione
selectOneMenu	Permette la scelta attraverso un menù a tendina di un elemento
selectBooleanCheckbox	Corrisponde ad una *checkbox* il cui valore può essere collegato alla proprietà booleana di un *bean*
simpleSearchInput	Ricerca nell'inventario un codice inserito
calendarInput	Permette la selezione di una data
inputText	Permette l'inserimento di un testo breve
inputTextarea	Permette l'inserimento di un testo anche lungo
inputErrorNotice	Visualizza un messaggio di errore nella pagina
navigationMenu	Ha lo scopo di visualizzare le voci fornitegli sottoforma di menù e reindirizzare la navigazione alla pagina prescelta
Continua nella prossima pagina	

[1]Per ulteriori informazioni si rimanda al sito `http://twitter.github.com/bootstrap/`

Continua dalla pagina precedente

Componente	Descrizione
dataScroller	Suddivide gli elementi di una tabella in blocchi e permette di navigare da un blocco ad un altro
htmlHead	Aggiunge l'*header* standard per un'applicazione web *Carel*
paragraphTitle	Inserisce il titolo di un paragrafo

5.2.3 La pagina *welcome*

La pagina di benvenuto è molto semplice. Come mostrato in figura 5.1, contiene il menù nella parte superiore ed un'immagine seguita da una scritta di benvenuto al di sotto.

Figura 5.1: La pagina di benvenuto

5.2.4 La pagina *preseries*

La pagina *preseries* si occupa della creazione di una nuova richiesta, in questo caso la pagina appare come in figura 5.2. I campi creatore e data di creazione sono autocompilati, al di sotto vanno indicati i dati che dovranno essere inseriti a cura del richiedente.

Una seconda funzione svolta dalla pagina è la visualizzazione o la modifica, cioè l'avanzamento, di una richiesta. In questo caso la pagina si presenta come in figura 5.3, dove in alto è presente un riepilogo dei dati inseriti e al di sotto, al solo responsabile della fase in corso, è data la possibilità di inserire i dati richiesti.

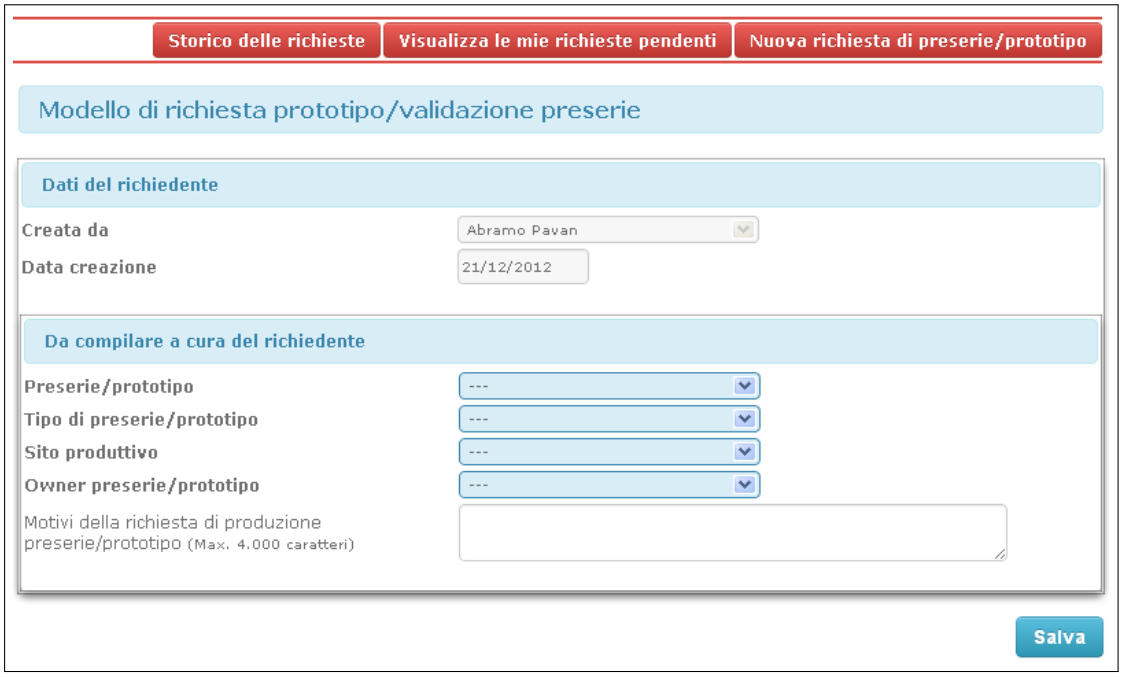


Figura 5.2: La pagina *preseries*, creazione di una nuova richiesta

La pagina è organizzata in una vista e la possibilità di eseguire diverse funzioni è gestita attraverso i *rendered*. Ciascun componente della pagina è legato ad una variabile del *back bean PreseriesPage* che contiene inoltre anche i metodi utilizzati dai *rendered*. Questi ritornano valori booleani e servono a discriminare ad esempio se la richiesta in questione è ad una certa fase o se l'utente loggato possiede i privilegi per avanzare nel flusso.

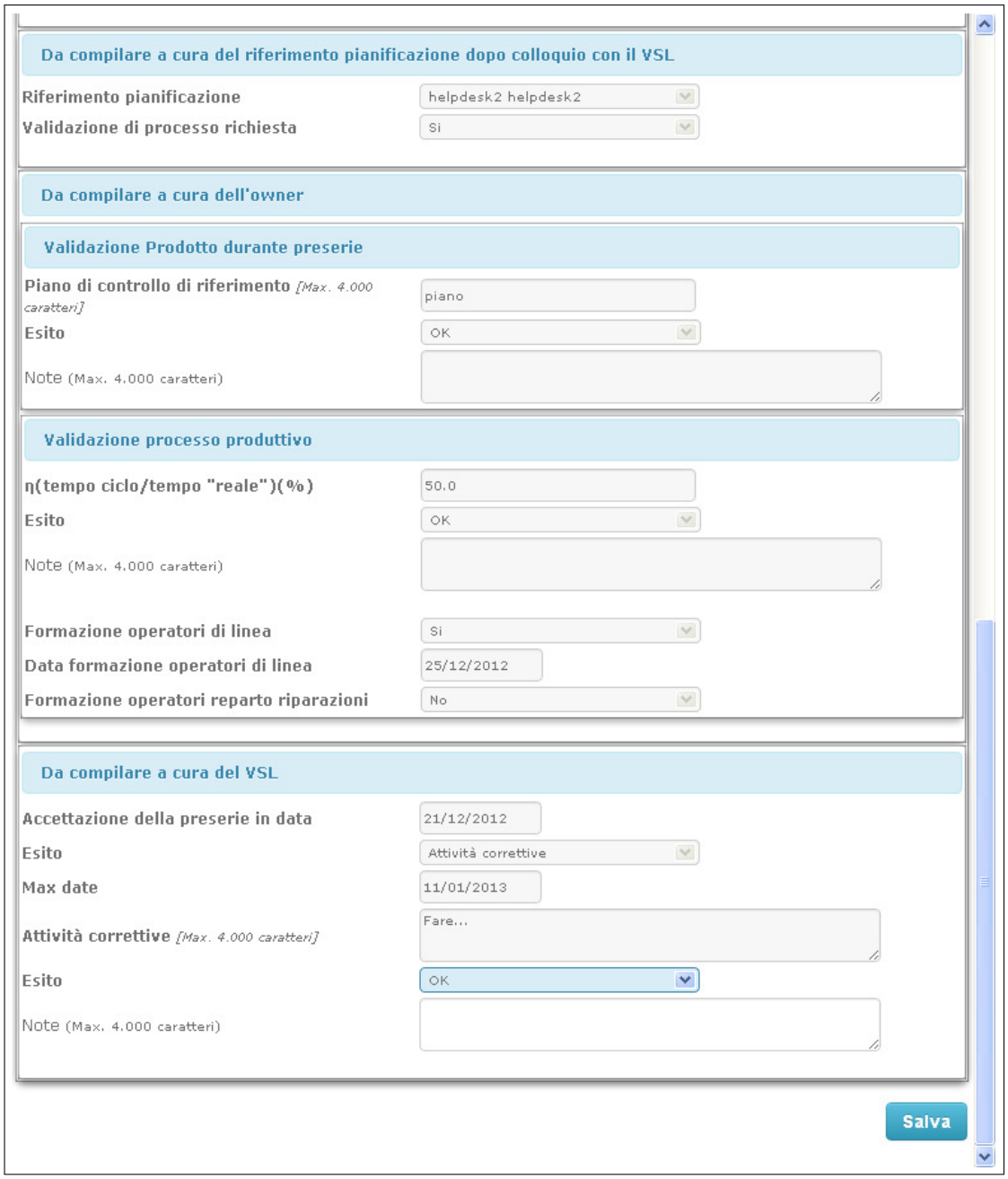


Figura 5.3: La pagina *preseries*, visualizzazione di una richiesta

5.2.5 La pagina *preserieshistory*

La pagina *preserieshistory* si occupa di eseguire le ricerche nello storico delle richieste completate. Appare come mostrato in figura 5.4. L'utente può settare tutta una serie di parametri, scelti in fase di analisi, per poi andare a premere il pulsante "ricerca".

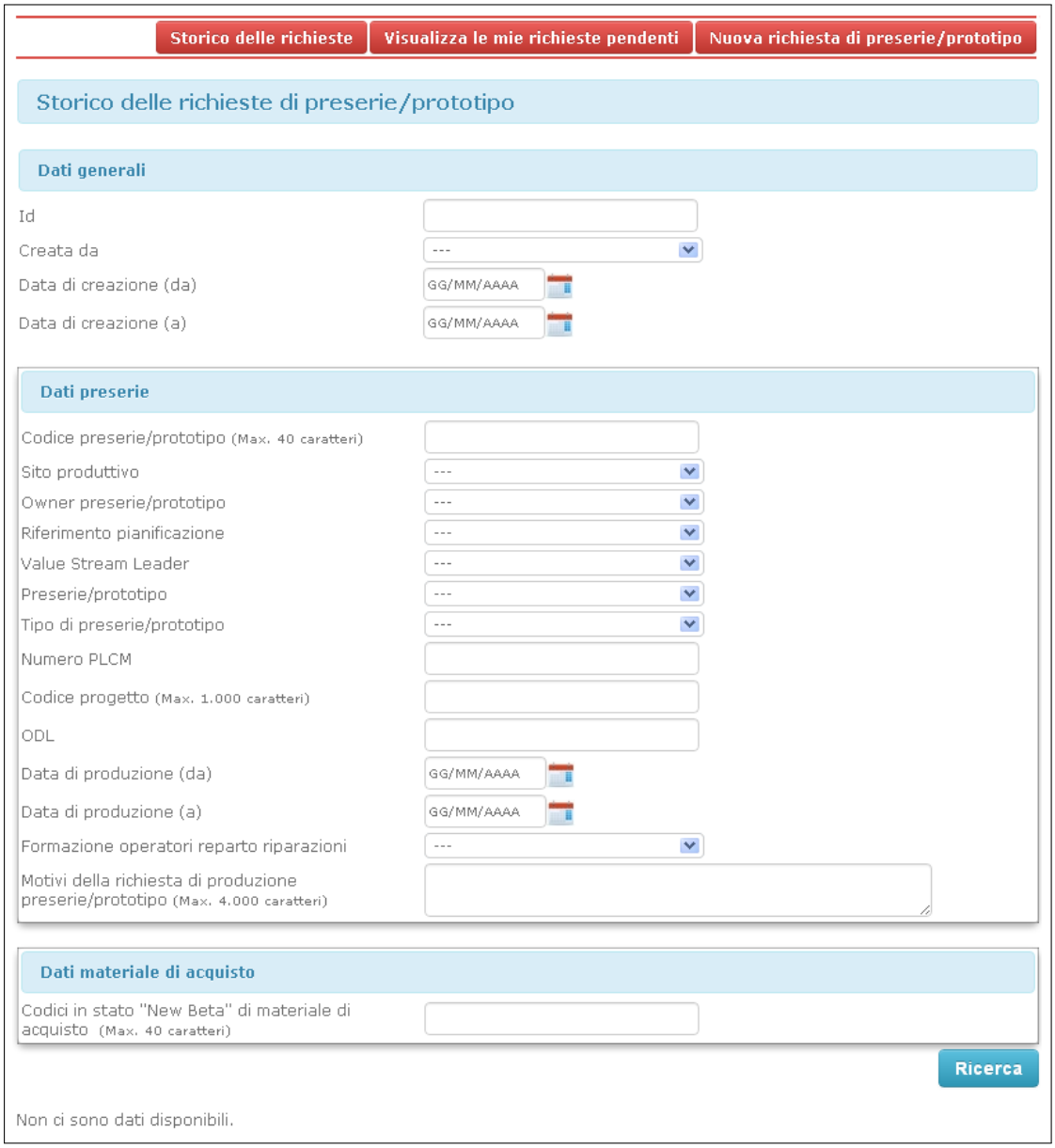


Figura 5.4: Ricerca nella storico delle richieste

I risultati verranno proposti a fondo pagina in una tabella, come mostrato in figura 5.5. La prima colonna è l'*id* della richiesta seguita da ulteriori informazioni

riguardanti tale richiesta. Utilizzando l'icona "Mostra", in ultima colonna, sarà possibile accedere a tutti i dati riguardanti la richiesta.

Figura 5.5: Risultati ricerca nello storico

E' possibile inoltre accedere ad alcuni dettagli andando a premere sull'*id* desiderato. Questo aprirà al di sotto una finestra riportante ulteriori informazioni riguardanti la preserie ed il materiale utilizzato, come mostrato in figura 5.6.

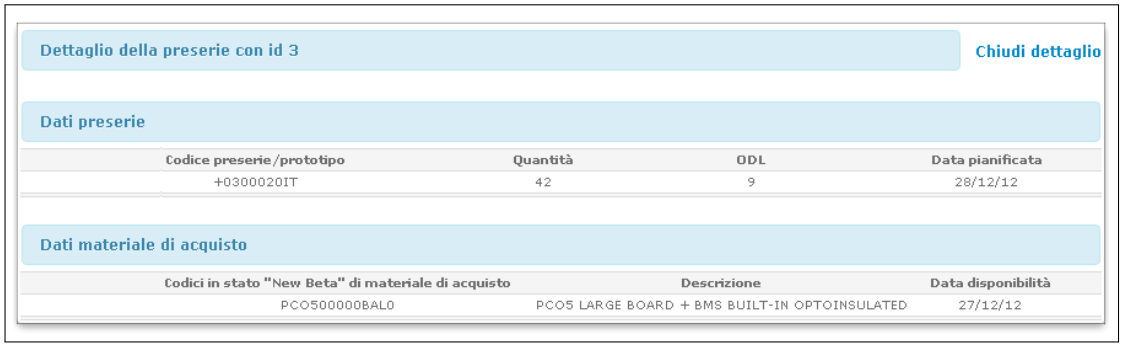


Figura 5.6: Dettagli di una preserie ricercata

Questa pagina serve inoltre alla visualizzazione delle richieste pendenti dell'utente loggato ed in questo caso appare come mostrato in figura 5.7.

Figura 5.7: Visualizzazione delle richieste pendenti

Le richieste pendenti di un utente sono quelle che attendono l'inserimento di alcuni dati da parte di tale utente. Attraverso questa pagina è possibile accedere all'inserimento di tali dati attraverso la colonna "Modifica". Tramite la colonna

"Mostra" è possibile visionare i dati finora inseriti, mentre tramite la colonna "Elimina" è possibile rimuovere una data richiesta. La rimozione è effettuabile solo da parte del richiedente e se la richiesta è al suo primo stadio.

5.3 Il *Controller*

In questa sezione verranno esaminati i componenti del *Controller*. Si partirà analizzando le risorse, cioè i file in cui sono inserite tutte le etichette utilizzate all'interno del programma. Si passerà poi ai convertitori, ad una breve spiegazione di come avviene la schedulazione di un promemoria e la gestione delle mail. In finale si analizzeranno singolarmente ognuno dei *backing bean* componenti l'applicazione.

5.3.1 Le risorse

Con risorse si intendono i file contenenti tutte le etichette utilizzate nella creazione delle pagine web e delle e-mail. Il listato seguente contiene parte del file di configurazione *faces-config.xml*. In questa parte sono dichiarate le lingue supportate, italiano ed inglese, di cui l'italiano è la lingua di default. Sono poi configurate le risorse.

Listing 5.18: Configurazione della lingua e delle risorse

```
1   <locale-config>
2     <default-locale>it</default-locale>
3     <supported-locale>en</supported-locale>
4   </locale-config>
5   <message-bundle>com.carel.portal.preseries.view.messages.application
6   </message-bundle>
7   <resource-bundle>
8     <base-name>com.carel.portal.preseries.view.messages.application</base-name>
9     <var>res</var>
10  </resource-bundle>
11  <resource-bundle>
12   <base-name>com.carel.portal.commons.view.messages.general.general</base-name>
13   <var>general</var>
14  </resource-bundle>
```

Da riga 7 a 10 è definito dove si trova il file contenente le etichette create per questa applicazione e la variabile da utilizzare per invocare, mentre da riga 11 a 14 avviene lo stesso per le etichette più generali, cioè quelle che vengono utilizzate in più applicazioni.
L'inserimento in una pagina avviene passando all'attributo *label* di un certo componente personalizzato Carel attraverso un'espressione, come mostrato nei due listati seguenti. Il primo utilizza la variabile "res" per invocare l'etichetta, del file contenente quelle create appositamente per quest'applicazione, che di nome fa "preseries.result".

Listing 5.19: Estrazione di un'etichetta da quelle relative all'applicazione

```
1 label="#{res['preseries.result']}"
```

Il secondo invece recupera l'etichetta di nome "general.note" dal file contenente quelle generali.

Listing 5.20: Estrazione di un etichetta da quelle generali

```
1 label="#{general['general.note']}"
```

Infatti tutte le etichette sono dichiarate utilizzando un nome seguito dal carattere uguale e poi dalla stringa da far apparire. Come mostrato qui di seguito.

Listing 5.21: Dichiarazione di una voce

```
1 preseries.result=Esito
```

5.3.2 I convertitori

All'interno della base di dati sono state eseguite delle scelte progettuali tali per cui il tipo di richiesta, lo stato ed altri attributi sono numerici. Presentare però un numero all'utente non lo renderebbe chiaro. Si utilizzano quindi i convertitori per convertire un certo valore, presente in un campo, in una stringa facilmente comprensibile all'utente.
I convertitori vanno dichiarati all'interno del file *faces-config.xml* indicando il file in cui è presente la loro implementazione, come mostrato nel listato seguente.

Listing 5.22: File di configurazione *faces-config.xml*

```
1   <converter>
2     <converter-id>UserConverter</converter-id>
```

```
3      <converter-class>com.carel.portal.commons.converters.UserConverter
4      </converter-class>
5    </converter>
```

I convertitori utilizzati ed il loro scopo sono riassunti nella seguente tabella.

Tabella 5.2: Tabella dei convertitori

Nome	Descrizione
UserConverter	Converte l'*id* di un utente nel suo nominativo
YesOrNoConverter	Converte i valori numerici 0 e 1 rispettivamente in "No" e "Si"
	Converte i valori numerici 0 e 1 rispettivamente in "Preserie" e "Prototipo"
CarelPreseriesHPreseriesTypeConverter	Converte i valori numerici 0 e 1 rispettivamente in "Nuovo progetto" e "PLCM"
CarelPreseriesHStatusConverter	Converte i valori numerici in una stringa rappresentante lo stato in cui si trova la preserie

5.3.3 La gestione delle e-mail

Le e-mail vengono inviate ogniqualvolta venga superato una fase all'interno del flusso per avvisare l'utente responsabile della fase successiva. La creazione delle e-mail avviene all'interno della classe *PreseriesEmailManager*. Ognuna riporta un riassunto dei dati precedentemente inseriti. Le e-mail sono create utilizzando codice HTML.

5.3.4 La schedulazione dei *job*

I *job* da schedulare sono le e-mail da inviare come promemoria. All'interno del flusso ne sono presenti due:

- promemoria di completamento della *check list*;

- promemoria di completamento delle attività correttive per ottenere la validazione della preserie.

Queste e-mail devono essere inviate come promemoria in una precisa data. Per far questo si è utilizzato *Quartz*[2] un *job scheduler*. Attraverso il *bean PreseriesEmailAlertScheduler* ogni giorno vengono ricercati i promemoria da inviare, andando ad interrogare la base di dati, e nel caso ci sia bisogno vengono inviati i promemoria che sono creati nella classe *PreseriesAlertEmailManager*. La creazione dei promemoria avviene analogamente a quanto già visto per le altre e-mail.

5.3.5 Il *GeneralBackBean*

In questo *bean* vengono create le liste che verranno poi utilizzate dall'utente nel momento in cui dovrà scegliere una risposta all'interno di alcune possibilità, come ad esempio nei menù a comparsa. Le liste che sono state create sono riassunte nella seguente tabella.

Tabella 5.3: Tabella delle liste utilizzate nell'applicazione

Nome	Descrizione
preseriesUsersList	Lista di tutti gli utenti aventi uno almeno uno dei profili compatibile con quelli dell'applicazione
preseriesPlanningList	Lista degli utenti aventi il profilo *planning*
preseriesVSLList	Lista degli utenti aventi il profilo *valueStreamLeader*
typeList	Lista contenente due possibilità di scelta: "Preserie" e "Prototipo"
preseriesTypeList	Lista contenente due possibilità di scelta: "Nuovo progetto" e "PLCM"
resultList	Lista contenente due possibilità di scelta: "Ok" e "Not Ok"
resultFinalValidationList	Lista delle possibili scelte di risultato della validazione
globalChecklist	Lista contenente tutte le possibili voci inseribili in una *check list*

[2]Per ulteriori informazioni si rimanda al sito ufficiale `http://quartz-scheduler.org/`

5.3.6 Il *WelcomePage bean*

E' il *backing bean* della pagina di benvenuto. Contiene, come tutte gli altri *bean* di una pagina, la parte riguardante la navigazione analizzata in sezione 4.3. Esegue inoltre l'inizializzazione della sessione dell'utente.

5.3.7 Il *PreseriesPage bean*

E' il *backing bean* della pagina *preseries*. Contiene una variabile per ogni componente della pagina web e per ognuna di queste variabile deve essere implementato un metodo *get* e *set*. I metodi principali implementati sono i seguenti:

- *clearBackBean()*, viene invocato quando è necessario resettare tutte le parti della pagina web, per preparare la pagina alla creazione di una nuova richiesta, e più precisamente esegue le seguenti operazioni:
  - resetta tutti i componenti;
  - resetta le variabile booleane utilizzate come *rendered*;
  - resetta la variabile corrispondente all'istanza di *CarelPreseriesH* attualmente in visualizzazione;
  - svuota la lista di preserie/prototipi da produrre per questa richiesta;
  - svuota la lista del materiale in stato *new beta* necessario;
  - svuota la lista delle voci della *check list*.
- *setSelectedPreseries(CarelPreseriesH preseries)*, questo metodo viene invocato quando bisogna settare tutti i componenti in modo tale da visualizzare correttamente la richiesta *preseries*, passata come argomento al metodo.
- *checkHeaderBackBean()*, questo metodo controlla, a seconda dello stato in cui la richiesta è arrivata, se sono stati inseriti dati corretti e se i campi necessari sono stati inseriti.
- *savePreseries()*, viene invocato nel momento in cui viene premuto il pulsante di salvataggio della richiesta. Come prima cosa questo metodo invoca il metodo *checkHeaderBackBean()* appena descritto e se i dati inseriti sono corretti esegue il salvataggio della richiesta sulla base dei dati e l'invio delle e-mail di notifica.

Oltre a quelli visti, sono implementati una serie di metodi che vengono invocati se il valore di un certo componente cambia. Ad esempio esiste un menù a tendina in cui è possibile scegliere se la preserie è una PLCM o un nuovo progetto. In base al valore selezionato in questo menù bisogna quindi richiedere il numero della PLCM o il codice del progetto. Se quindi il *listener* segnala che il valore di quel componente è cambiato viene invocato un metodo che analizza qual è il nuovo valore e decide qual è il componente da visualizzare di conseguenza.

Come già visto con una richiesta è possibile produrre più preserie/prototipi, queste verranno visualizzate in una tabella, come mostrato in figura 5.8.

Figura 5.8: Porzione delle pagina in cui si andranno ad inserire i dati riguardanti le preserie/prototipi

Sono presenti una serie di metodi per gestire la fase in cui si debbono inserire le preserie/prototipi che comporranno questa tabella. L'utente andrà a premere sul pulsante "Crea" che aprirà una finestra, visibile in figura 5.9.

Figura 5.9: Finestra in cui è possibile inserire i dati riguardanti la preserie(prototipo

Sarà quindi possibile inserire i dati necessari e premendo il pulsante "Salva" verrà verificata la correttezza dei dati e verrà creato il *CarelPreseriesL* necessario per il salvataggio. La tabella verrà quindi popolata come mostrato in figura 5.10.

Figura 5.10: Visualizzazione dopo l'inserimento

A questo è possibile ripetere l'operazione per inserire altre preserie/prototipi. Il salvataggio sul database però avverrà soltanto al momento in cui verranno inseriti anche gli altri dati necessari per la fase in cui si trova e verrà premuto il pulsante "Salva" della richiesta.
In modo analogo è gestita anche la tabella dei materiali in stato *new beta*.

Naturalmente anche questo *bean* contiene i metodi necessari per gestire la navigazione fra le pagine.

5.3.8 Il *PreseriesHistory bean*

E' il *backing bean* della pagina *preseriesHistory*. Anche questo *bean* contiene una variabile per ogni componente della pagina web e per ognuna di queste variabile è implementato un metodo *get* e *set*. I metodi principali implementati sono i seguenti:

- *clearBackBean()*, viene invocato quando è necessario resettare tutte le parti della pagina web. E' utilizzato prima che la pagina venga usata per eseguire una ricerca nello storico.

- *init()*, questo metodo viene invocato quando la pagina deve essere utilizzata per visualizzare le richieste pendenti. Esegue una *query* ricercando quelle dell'utente loggato e le visualizza in forma tabellare.

- *searchAction()*, questo metodo viene invocato quando viene premuto il tasto di ricerca nello storico. Si occupa di costruire la *query string* e la lista di parametri inseriti che verranno utilizzati dal *session bean* per ricercare le richieste nello storico che soddisfano i parametri desiderati. Si occupa inoltre anche di visualizzare i risultati ottenuti.

Anche questo *bean* contiene i metodi necessari per gestire la navigazione fra le pagine e metodi *listener*.

5.4 Test e manuale utente

Una volta conclusa l'implementazione sono stati eseguiti una serie di test in cui si sono provate tutte le funzionalità offerte dall'applicazione. Questo ha permesso di eseguire piccoli accorgimenti prima del rilascio dell'applicazione. E' stato inoltre scritto un manuale utente con lo scopo di fornire, agli utenti che andranno ad utilizzare questo modulo, le istruzioni per lavorare correttamente. Queste sono suddivise a seconda del profilo dell'utente. Nel manuale sono fornite le istruzioni passo passo per eseguire ognuna delle seguenti operazioni:

- utente *user*:
  - creazione di una nuova richiesta preserie/prototipo (parte richiedente);
  - nuova richiesta preserie/prototipo (parte dell'*owner*);
  - verifica del completamento della *check list*;
  - inserimento di una nuova data pianificata;
  - validazione da parte dell'*owner*;
  - ricerca nello storico delle richieste concluse;
  - visualizzazione delle richieste pendenti a proprio carico.
- utente *pianificazione*:
  - pianificazione della produzione.
- utente *value stream leader*:
  - validazione finale (Ok/ not Ok);
  - validazione finale (Attività correttive).

Capitolo 6

Conclusioni e sviluppi futuri

L'obiettivo di questo progetto era l'ottimizzazione del processo industriale di richiesta di produzione di un prototipo o validazione di una preserie. Questo è stato ottenuto attraverso l'informatizzazione del flusso industriale in un'applicazione web, da inserire all'interno del portale aziendale.

Dopo un'ampia fase di analisi del processo, svolta seguendo le tecniche di produzione *lean*, si sono stese le specifiche secondo lo standard "830-1998, *IEEE Recommended Practice for Software Requirements Specification*". E' stata poi progettata ed implementata una base di dati il cui scopo è contenere le informazioni necessarie per questo flusso. Si è quindi passati alla progettazione e implementazione dell'applicazione web, basandosi sulla piattaforma Java EE. Questo ha permesso di approfondire il funzionamento dei *framework*:

- EJB, comprendente *Entity* e *Session Beans*, per la gestione della parte di business di un'applicazione;
- JSF, per la realizzazione dell'interfaccia utente;

L'applicazione è stata successivamente testata e si è realizzato un manuale utente da distribuire al personale che dovrà poi utilizzare questo modulo.

I vantaggi che si sono ottenuti attraverso l'informatizzazione di questo processo aziendale sono:

- il flusso delle operazioni è stato reso più snello, eliminando durante la fase di analisi l'inserimento di informazioni che non sono risultate utili al fine di questo processo produttivo;

- si è creato uno storico in formato digitale che permette l'interrogazione rapida per il reperimento delle informazioni necessarie al suo interno;
- si è velocizzato il passaggio da una fase all'altra grazie all'invio automatico di e-mail, che vanno ad informare l'utente successivo che la fase precedente è stata conclusa.

L'informatizzazione di questo processo ha fornito inoltre la base da ampliare per eseguire lo stesso procedimento per quelle che vengono definite preserie *light*. Queste hanno un flusso diverso dalle preserie studiate per questo progetto, dato che non richiedono la loro produzione. Sarà comunque possibile utilizzare la stessa applicazione con l'inserimento di un'altra pagina e di un'altra voce al menù per l'implementazione di questo flusso, che dovrà essere analizzato in modo da essere reso più snello.

Appendice A

A.1 Modulo cartaceo richiesta di produzione prototipo/validazione preserie

CAREL

Mod. RICHIESTA PROTOTIPO / VALIDAZIONE PRESERIE

PROTOTYPE REQUEST / PRE-SERIES VALIDATION Form

PARTE A / *PART A* - Da compilare a cura del richiedente /
To be filled in by the petitioner

MODULO N. / *FORM N.:*	Data (dd/mm/yy) / *Date*:
☐ PRESERIE / *PRE-SERIES*	☐ PROTOTIPI / *PROTOTYPES*
COD. PROGETTO / *PROJECT CODE*: C	PROJECT LEADER:
NUMERO PLCM / *PLCM NR.*:	PLCM OWNER:
SITO PRODUTTIVO / *PRODUCTION SITE:*	REF. DEL SITO / *SITE REF.*:
Data fine Preserie - Prototipo richiesta / *Requested Pre-series - Prototype ending date:*	

Livello Distinta Base/ ***B.O.M. Level***	***Codice/*** ***Code***	***Descrizione/*** ***Description***	***n°pezzi/*** ***n°pieces***[1]	***multipli/*** ***multiples***	***ODL* a cura di Pianificazione/** ***to be filled in by Planning Dept.***

Codici in stato "New Beta" di materiale di acquisto / ***codes in "New Beta" status of purchased material***	***Descrizione/*** ***Description***	***Data disponibilità /*** ***Availability date***	***Responsabili disponibilità /*** ***Availability supervisors***

Checklist		**Data di disponibilità /** ***Availability date***	
Ciclo di lavoro in Oracle (con linea e battente) / *Oracle work instructions / routings*	☐	Distinta Base / *B.O.M.*	☐
Ciclo di lavoro cartaceo / *Paper work instrution / routings*			
Schema elettrico / *Electrical drawings*	☐	Istruzioni di assemb. e imball. / *Assembling & packaging instructions*	☐
Planimetrie / *Electronic hardware layout*	☐	Istruzioni di resinatura / *Coating instructions*	☐
Istruzioni di collaudo / *Test instructions*	☐	Lista attrezzature / *Equipments list*	☐
Lamine per serigrafia SMD / *SMD screen printing frame*	☐	Programma per ispezione ottica / *Optical check software*	☐
Programma per serigrafia SMD / *SMD screen printing software*	☐	Programma per stampigliatura laser / *Laser printing software*	☐
Programma Pick & Place / *Pick & Place software*	☐	Attrezzature di produzione e collaudo / *Production and Tests equip.*	☐
Disponibilità materiale / *Material availability*	☐	Firmware & Software / *Firmware & Software*	☐
Attività/ Controlli supplementari / *Additional activities/checks*	☐	Piano di controllo / *Control plan*	☐
Etichette prodotto / *product labels*	☐	Set-up di linea / *Line set-up*	☐
Note / *Notes*			

Specificare Magazzino di destinazione/ *To be specify products destination*

☐ Magazzino Oracle 1/ *Oracle warehouse 1*	☐ Magazzino Oracle 13/ *Oracle warehouse 13*

Nel caso di Prototipi indicare dove saranno realizzati / In case of Prototypes indicate where they will be made:

☐ Utilizzando risorse di produzione (risorse uomo/ tools)/ *Using production resources (human resources / tools)*	
☐ Produzione/ *Production Dept.*	☐ Out Sourcing/ *Out Sourcing*
Note/ *Note*	

PARTE B / *PART B* - Da compilare a cura della Pianificazione /
To be filled in by Planning Dept.

Data fine confermata/ *ending confirmed date:*	*riferimento Pianificazione/ Planning reference:*
Note/ *Note*	

PARTE C / *PART C* - Da compilare a cura della Produzione /
To be filled in by Production Dept.

Approvazione data fine confermata/ *ending confirmed date approval:*	
Note/ *Note*	

[1] Il numero pz. preserie deve essere inferiore o uguale a **50** / *The pre-series pieces must be less or equal to* ***50***

1/2 *Prototype_Request/ Pre-series_Validation form R.7.00 30/06/10*

CAREL

Mod. RICHIESTA PROTOTIPO / VALIDAZIONE PRESERIE

PROTOTYPE REQUEST / PRE-SERIES VALIDATION Form

PARTE D / *PART D* - Da compilare a cura del richiedente /
To be filled in by petitioner

Validazione Prodotto durante preserie / ***product validation during pre-series***	***Piano di controllo di riferimento*** **(Sampling Features o altro documento) /** ***Reference Control plan*** ***(Sampling Feature or other document)***	***Esito/ Result*** N.A.	OK	NOT OK	***Note***
		☐	☐	☐	

Validazione processo produttivo / ***producttion process validation***	***Obiettivi Q, C di riferimento /*** ***Reference Q, C targets***	***Esito/ Result*** N.A.	OK	NOT OK	***Valori rilevati***	***Note***
	Q test funzionale: difettosità ≤ 5 %	☐	☐	☐		
	Q test in circuit: difettosità ≤ 2 x OMI plant (Prod. Elettr.)	☐	☐	☐		
	η (tempo ciclo / tempo "reale") (%) > 50%	☐	☐	☐		

Formazione operatori di linea / *Line operators training*:	NO ☐	SI/*YES* ☐	Se SI, data/If YES, date:
Formazione operatori reparto riparazioni / *Service dept. training*:	NO ☐	SI/*YES* ☐	Se SI, data/If YES, date:

Peso lordo unitario (Kg) / *Unit gross weight (Kg)*	
Volume unitario (m^3) / *Unit volume (m^3)*	
Dimensioni lunghezza/larghezza/altezza (cm) / *Dimensions lenght/width/height (cm)*	

PARTE E / *PART E* – Nel caso di sito produttivo esterno da compilare a cura del project leader/
In case of external productive site to be filled in by project leader

Validazione Prodotto a fine preserie / ***product validation at the end of pre-series***	***Sampling Features di riferimento /*** ***Sampling Feature reference***	***Esito/ Result*** N.A.	OK	NOT OK	***Note***
		☐	☐	☐	

PARTE F / *PART F* – Da compilare a cura della Produzione /
To be filled in by Production Dept.

ACCETTAZIONE DELLA PRESERIE E AVVIO DELLA PRODUZIONE DI SERIE/ ***PRE-SERIES ACCEPTANCE AND START OF PRODUCTION***	
Data accettazione e presa in carico da parte della produzione / *Acceptance and takeover by Production*:	
Value Stream Leader o Responsabile di Stabilimento / *Value Stream Leader or Plant Manager (Production Dept. reference)*:	

A.2 Modulo cartaceo richiesta light

CAREL Mod. VALIDAZIONE PRESERIE LIGHT PJEZ FW/Parametri

PARTE A - Da compilare a cura dell' owner della preserie

Data (gg/mm/aa)**:**	
COD. PROGETTO :	OWNER PLCM:
NUMERO PLCM:	PRESERIE OWNER:
SITO PRODUTTIVO: *CID*	RIF. PRODUZIONE:

Codice Padre	***Descrizione***	
Codice Nuovo	***Descrizione***	***Codice a barre***

Checklist		Etichette prodotto:			
Aggiornamento linee	☐				
Presenza Firmware & Software	☐				
Assenza errore EEP	☐				
Invio pacchetto ai referenti in Cina e Brasile	☐				
Aggiornamento documentazione test da validare	☐				
Allegati					
Ciclo di lavoro in Oracle/Disegno (padre/figlio)	☐	***Validazione Prodotto:***	***Esito***		
Distinta Base	☐	***Piano di controllo di riferimento*** **(PLCM o altro documento)**	N.A.	OK	NOT OK
Report di programmazione remota	☐				
Confronto distinte (padre/figlio)	☐		☐	☐	☐
Padre Active	☐		Firma		
Note:					

PARTE B– Da compilare a cura della Produzione / Value Stream Leader

Verifica corrispondenza caratteri HW padre figlio: (es.PZPU***COMB0***2K cod.padre – PZPU***COMB0***6K cod.figlio)	☐
Verifica lettura codice a barre nella macchina di test	☐
Verifica corrispondenza num. ciclo, cod. disegno, presenza linea e lotto	☐
Verifica report programma off-line (Presenza file su tutte le linee)	☐
Verifica presenza etichetta	☐
Verifica padre Active	☐
Verifica distinta base Active e confronto	☐
Data presa in carico da parte della produzione:	
Firma Value Stream Leader:	

1/1

Mod. Validazione preserie light PJEZ_ R.1.00

Appendice B

B.1 Carel Preserie Portal – Documento di specifica requisiti

SOFTWARE REQUIREMENT SPECIFICATION

CAREL

Carel S.p.A.

Carel Preserie Portal – Documento di specifica requisiti

Versione	Autore	Note	Data	Approvatore
1.0	Nicola Celli	Stesura iniziale	24/10/2012	

Sommario

1. Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è specificare i requisiti funzionali e tecnici del modulo del portale per la richiesta di validazione di una preserie o la richiesta di produzione di un prototipo.
I destinatari della SRS sono i reparti IT, operations, qualità e sviluppo prodotto.

1.2 Definizioni, acronimi ed abbreviazioni

SRS Software Requirement Specification, documento di specifica requisiti
VSL Value Stream Leader
PLCM Product Life-Cycle Management
ODL Ordine Di Lavoro

1.3 Introduzione del prodotto

Il progetto ha lo scopo di creare una web application per permettere ad un dipendente Carel, che chiameremo richiedente, di produrre un prototipo o di far validare una preserie. Entrambe possono essere di due tipi:

- un nuovo progetto;
- una PLCM, cioè la modifica di un prodotto già in vendita;

Le PLCM della famiglia PJEZ in cui avvengono solo modifiche firmware o parametriche (sono prodotti in cui certi parametri sono personalizzabili per il cliente) non necessitano di essere prodotte per passare la validazione, il richiedente eseguirà quindi una richiesta light. Questa tipo di richiesta potrà in seguito essere adottata da altre famiglie di prodotto, ma il flusso della web-application rimarrà invariato. L'implementazione della richiesta light non avverrà nella prima versione del software.

Gli step saranno:

1. Il richiedente indica se è una richiesta light o una richiesta standard;

Se è una richiesta standard:

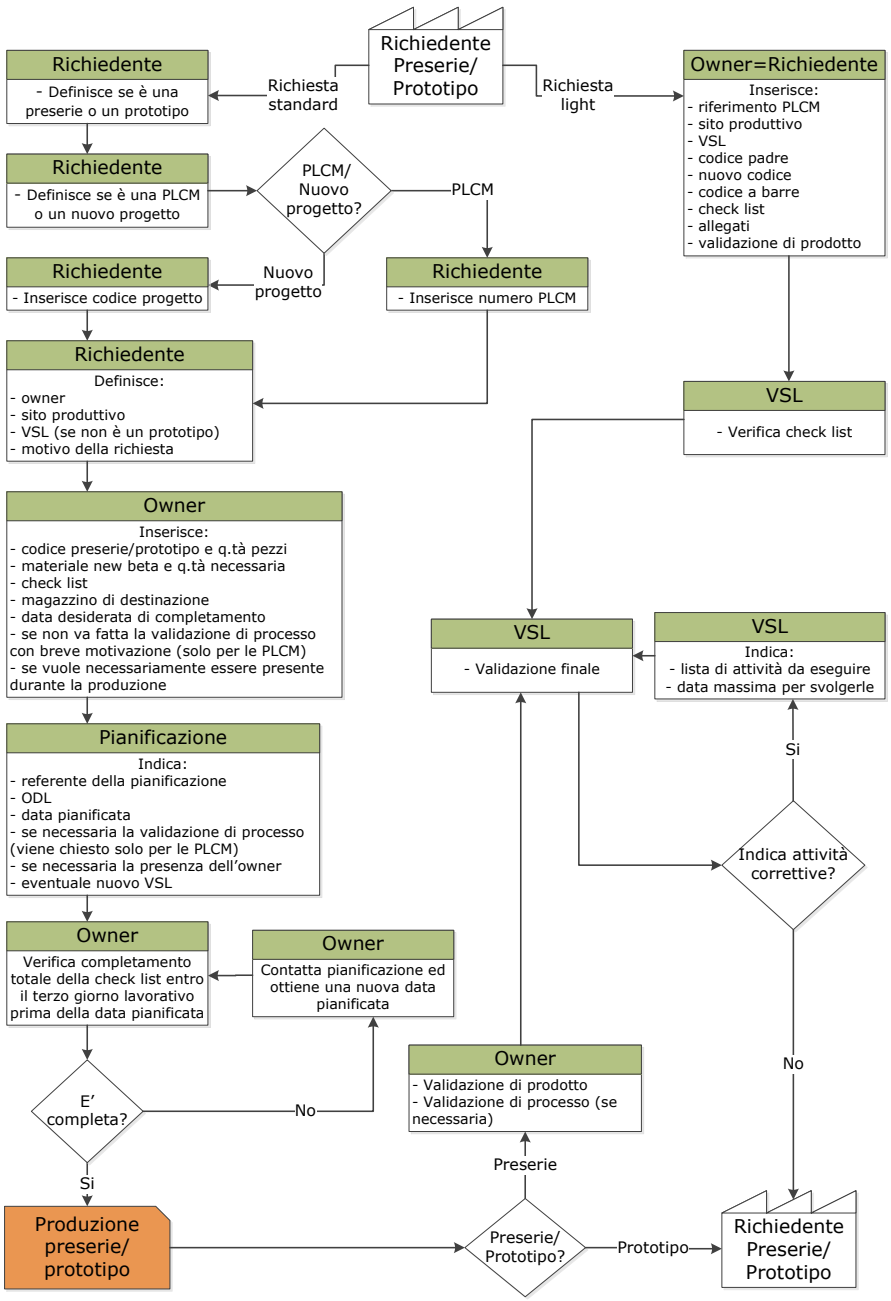
2. Il richiedente crea una richiesta di produzione preserie/prototipo indicando se è una preserie o un prototipo, la tipologia, il sito produttivo, l'owner e il VSL;
3. L'owner indica i codici della preserie/prototipo, i materiali necessari, compila una check list delle operazioni che dovrà eseguire, richiede una data entro cui il processo sia finito, indica se vuole necessariamente essere presente al momento della produzione e se è necessaria la validazione di processo;
4. La pianificazione in accordo con il VSL fissa la data di produzione, se è necessario la presenza dell'owner al momento della produzione e se è necessaria la validazione di processo;
5. L'owner, prima della data pianificata, verifica il completamento totale della check list;
6. Avviene la produzione della preserie/prototipo; nel caso del prototipo il flusso dell'applicazione termina;
7. L'owner compila i form di validazione prodotto e di validazione processo;

Se è una richiesta light:

2. L'owner, che coincide con il richiedente, compila la form di richiesta;
3. L'owner compila una check list ed allega alcuni file necessari;
4. L'owner compila il form di validazione prodotto;
5. Il VSL esegue la verifica della check list e dei file allegati;

Entrambe le richieste confluiscono all'ultimo passo nel quale:

- Il VSL esegue la validazione finale;


Richiedente Preserie/ Prototipo
Richiesta standard
Richiesta light
Richiedente
- Definisce se è una preserie o un prototipo
Owner=Richiedente
Inserisce:
- riferimento PLCM
- sito produttivo
- VSL
- codice padre
- nuovo codice
- codice a barre
- check list
- allegati
- validazione di prodotto
Richiedente
- Definisce se è una PLCM o un nuovo progetto
PLCM/ Nuovo progetto?
PLCM
Nuovo progetto
Richiedente
- Inserisce codice progetto
Richiedente
- Inserisce numero PLCM
Richiedente
Definisce:
- owner
- sito produttivo
- VSL (se non è un prototipo)
- motivo della richiesta
VSL
- Verifica check list
Owner
Inserisce:
- codice preserie/prototipo e q.tà pezzi
- materiale new beta e q.tà necessaria
- check list
- magazzino di destinazione
- data desiderata di completamento
- se non va fatta la validazione di processo con breve motivazione (solo per le PLCM)
- se vuole necessariamente essere presente durante la produzione
VSL
- Validazione finale
VSL
Indica:
- lista di attività da eseguire
- data massima per svolgerle
Si
Pianificazione
Indica:
- referente della pianificazione
- ODL
- data pianificata
- se necessaria la validazione di processo (viene chiesto solo per le PLCM)
- se necessaria la presenza dell'owner
- eventuale nuovo VSL
Indica attività correttive?
Owner
Verifica completamento totale della check list entro il terzo giorno lavorativo prima della data pianificata
Owner
Contatta pianificazione ed ottiene una nuova data pianificata
No
Owner
- Validazione di prodotto
- Validazione di processo (se necessaria)
E' completa?
No
Si
Preserie
Produzione preserie/ prototipo
Preserie/ Prototipo?
Prototipo
Richiedente Preserie/ Prototipo

2. Descrizione generale del prodotto

2.1 Ambito del prodotto

Il modulo in oggetto è un'applicazione web stand-alone che verrà integrata all'interno del portale aziendale.
La base di dati utilizzata è Oracle.

2.2 Funzionalità

L'applicazione web deve servire da supporto al processo di validazione di una preserie o di produzione di un prototipo.
L'applicazione deve essere multilingua. La lingua deve essere impostata in base al profilo dell'utente: se non è una delle lingue supportate deve essere impostata nella lingua inglese. Per i campi di non immediata comprensione, a fianco della label del campo deve essere disponibile un pulsante di help che apre un pop-up con la spiegazione del campo. La lista di questi campi verrà concordata in fase di progettazione del software.

2.2.1 Inserimento nel sistema tipologia di richiesta

Il richiedente, previa autenticazione, indica in un form se deve eseguire una richiesta standard o una richiesta light.

2.2.2 Inserimento nel sistema richiesta standard (parte richiedente)

Il richiedente compila un form nel quale va ad indicare: se è una preserie o un prototipo, se è una PLCM o un nuovo progetto e rispettivamente il numero di PLCM o il codice del progetto, se è una preserie indica il VSL(se il sito produttivo non è Carel Industries il VSL è il plant manager), l'owner, il sito produttivo e una breve motivazione della richiesta.
All'atto del salvataggio la richiesta verrà memorizzata nel database e verrà inviata una mail al richiedente, all'owner e al VSL per notifica.

2.2.3 Inserimento nel sistema richiesta standard (parte owner)

L'owner compila un form nel quale indica: il codice della preserie/prototipo e la quantità da produrre, il materiale new beta di cui necessita e la data di disponibilità di tale materiale, compila una check list di operazioni che deve aver completato prima della data pianificata, specifica un magazzino di destinazione, richiede una data di conclusione, indica se la validazione di processo non è necessaria (tale opzione sarà possibile solo per le PLCM) e se vuole necessariamente essere presente al momento della produzione.
All'atto del salvataggio la richiesta verrà memorizzata nel database e verrà inviata una mail alla pianificazione per richiedere la programmazione della produzione della preserie/prototipo.

2.2.4 Pianificazione

La pianificazione dopo essersi incontrata con il VSL compila un form nel quale indica: l'ODL, la data di produzione prevista, se è necessaria la presenza dell'owner al momento della produzione e se è necessaria la validazione di processo. La pianificazione avrà inoltre la possibilità di indicare un nuovo VSL. In questo caso la modifica verrà comunica tramite mail all'owner, al nuovo VSL e al vecchio VSL.
All'atto del salvataggio la richiesta verrà memorizzata sul database e verrà inviata una mail all'owner per notificare la pianificazione dell'ODL.

2.2.5 Verifica della check list

Tre giorni lavorativi prima della data pianificata viene inviata una mail all'owner come promemoria di verifica della check list. All'owner verrà presentata la check list precedentemente immessa nella fase di richiesta ed avrà la possibilità indicare quali voci ha completato.
Nel caso tutta la check list sia stata completata verrà inviata una mail alla pianificazione per segnalare che sono state eseguite tutte le operazioni necessarie per la produzione.
Nel caso non sia stato completato tutto è necessario ripianificare la produzione della preserie/prototipo, l'owner dovrà autonomamente contattare la pianificazione per richiedere una nuova data di produzione della preserie/prototipo, che andrà ad inserire.

2.2.6 Realizzazione della preserie/prototipo in produzione

Nel momento in cui viene prelevato il materiale dell'ODL dal magazzino, viene inviata una mail dell'owner in cui viene riportato se è necessario la sua presenza al momento della produzione e viene memorizzato nel database la data di effettiva produzione.

2.2.7 Validazione di prodotto e validazione di processo

L'owner dovrà compilare due form:

1. Il primo form riguarda la validazione di prodotto, l'owner dovrà inserire le specifiche di riferimento, eventuali note e se il prodotto ha passato tale validazione;
2. Il secondo form riguarda la validazione di processo, questo form sarà facoltativo e l'owner dovrà inserire il rendimento globale, eventuali informazioni aggiuntive e se il prodotto ha passato questa validazione. Andrà anche inserito se è stata eseguita la formazione degli operatori di linea e se è stata eseguita la formazione degli operatori del reparto riparazioni.

Al momento del salvataggio i risultati vengono memorizzati. Viene inviata una mail al VSL per segnalare che è necessaria la sua validazione finale. La validazione finale è richiesta anche nel caso la validazione di processo e di prodotto non vengano superate perché è possibile per l'owner giustificare i problemi incontrati utilizzando il campo note.

2.2.8 Inserimento nel sistema richiesta light

L'owner, che in questo tipo di richiesta coincide con il richiedente, compila un form inserendo: il riferimento PLCM, il sito produttivo, il VSL, il codice padre, il nuovo codice, il codice a barre (in allegato). Inoltre compila una check list, allega altri file necessari (da definire) e compila il form di validazione di prodotto.
Se è stata superata la validazione di prodotto viene inviata una mail al VSL per segnalare che è necessaria la sua validazione finale.

2.2.9 Validazione finale

Il VSL avrà a disposizione un form per approvare/non approvare/indicare delle attività correttive. Nel caso il VSL indichi delle attività correttive dovrà indicare anche una data massima entro la quale dovranno essere portate a termine. Questa opzione offre la possibilità al VSL di comunicare all'owner di effettuare delle modifiche per poter superare la validazione finale. Al secondo tentativo di validazione sarà possibile soltanto validare/non validare.
Nel caso vengano indicate delle attività correttive viene inviata una mail all'owner in cui sono inserite le richieste del VSL e la data entro cui eseguirle. Negli altri due casi viene inviata una mail al richiedente, all'owner e al VSL per segnalare il superamento/non superamento della validazione finale.

2.2.10 Ricerca nello storico

Un utente avente la possibilità di accedere all'applicazione avrà a disposizione un form apposito per la ricerca nello storico, secondo parametri da lui settati.

2.3 Utenti destinatari

Il modulo è utilizzabile:

- da personale Carel autorizzato.

I profili sono:

- VSL, personale interno Carel che validerà o non validerà la preserie richiesta;
- Pianificazione, personale di Operations addetto alla pianificazione degli ODL
- USER utilizzato dal richiedente e dall'owner.

2.4 Vincoli

I vincoli progettuali, tecnologici e funzionali a cui il prodotto è sottoposto sono i seguenti:

- La tecnologia utilizzata deve essere JSF (1.2) e EJB3;
- Il modulo web deve utilizzare come web container GlassFish 2 (ultima release stabile disponibile per questa versione);
- Il sorgente del modulo deve essere realizzato come progetto per Oracle Jdeveloper 11g;
- Le mail di alert verranno inviate solo per le preserie che verranno prodotte in Carel Industries, Carel Asia e Carel USA poiché sono le uniche organizzazioni aventi il calendario nella base di dati Oracle.

3. Specifica dei requisiti

In questo capitolo viene riportata la lista completa dei requisiti del prodotto suddivisa per tipologia utente e per classe funzionale.

3.1 Richiedente

3.1.1 Inserimento nel sistema tipologia di richiesta

Id	3.1.1
Titolo	Inserimento nel sistema della tipologia di richiesta: standard o light
Descr.	Attraverso un menù sarà possibile scegliere tra richiesta standard e richiesta light, a seconda della scelta si accederà al rispettivo form di creazione di nuova richiesta.
Versione	

3.1.2 Inserimento nel sistema richiesta standard

Id	3.1.2
Titolo	Inserimento nel sistema di richiesta standard da parte del richiedente
Descr.	Il form di inserimento richiede: 1. Di selezionare attraverso un menù se vuole inserire una preserie o un prototipo; 2. Scegliere se è una PLCM o un nuovo progetto; 3. La terza parte dipende da quanto inserito precedentemente, per le PLCM si dovrà inserire il numero di PLCM mentre per i nuovi progetti si dovrà inserire il codice del progetto; 4. Indicare obbligatoriamente l'owner ed il sito produttivo; 5. Se è una preserie va indicato obbligatoriamente il VSL; 6. Facoltativamente è possibile inserire una breve motivazione della richiesta. Verrà inviata una mail al richiedente, all'owner e al VSL per notifica.
Versione	

3.1.3 Ricerca nello storico

Id	3.1.3
Titolo	Ricerca nello storico
Descr.	Attraverso un form apposito sarà possibile effettuare una ricerca nello storico, secondo parametri impostabili. Al di sotto appariranno i risultati e sarà possibile accedere alla scheda della preserie/prototipo selezionata.
Versione	

3.2 Owner

3.2.1 Inserimento nel sistema richiesta standard

Id	3.2.1
Titolo	Inserimento della richiesta standard da parte dell'owner
Descr.	Il form è composto da più parti: 1. Nella prima viene inserito il codice della preserie/prototipo, la quantità da produrre e se vuole necessariamente essere presente al momento della produzione; Viene eseguito un controllo che la quantità da produrre sia inferiore di una quantità massima, inizialmente fissata a 50 unità; 2. Nella seconda va inserito il materiale new beta di cui si necessita e la data di

	disponibilità di tale materiale; 3. Nella terza specifica un magazzino di destinazione (sarà possibile scegliere tra magazzino 1 e 13), indica se la validazione di processo non è necessaria (tale opzione sarà possibile solo per le PLCM) e richiede una data di completamento della preserie. Viene eseguito un controllo per cui tale data sia successiva alla data di disponibilità del materiale new beta; 4. Nella seconda compila una check list delle operazioni che l'owner dovrà eseguire ed ha a disposizione anche un campo note per aggiungere altri voci; Verrà inviata una mail alla pianificazione per richiedere la programmazione della produzione della preserie.
Versione	

3.2.2 Inserimento nel sistema richiesta light

Id	3.2.2
Titolo	Inserimento nel sistema della richiesta light da parte dell'owner
Descr.	Il form è composto da più sezioni: 1. Nella prima sezione i campi da inserire sono: il riferimento PLCM, il sito produttivo, il VSL, il codice padre, il nuovo codice ed il codice a barre in allegato; 2. Nella seconda parte compila una check list, ma ha a disposizione anche un campo note per aggiungere altri voci, ed allega i file necessari; 3. Nella terza parte vengono inseriti i campi necessari per la validazione di prodotto: le specifiche di riferimento, eventuali note e se è stata superata/non superata. Se è stata superata la validazione di prodotto viene inviata una mail al VSL per segnalare che è necessaria la sua validazione finale.
Versione	

3.2.3 Verifica della check list

Id	3.2.3
Titolo	Verifica della check list prima di permettere la produzione
Descr.	La verifica della check list potrà essere effettuata in qualsiasi istante successivo all'inserimento della richiesta da parte dell'owner fino al massimo tre giorni lavorativi prima della data pianificata. Tre giorni lavorativi prima della data pianificata, nel caso la check list non risulti ancora completata, verrà inviata un'alert all'owner e alla pianificazione. Nel form sarà presentata la check list e l'owner avrà la possibilità di spuntare le operazioni effettuate. Nel caso tutto sia stato completato e si sta richiedendo la produzione di una preserie viene inviata una mail all'owner in cui se ne richiede la validazione. Nel caso tutto sia stato completato e si sta richiedendo la produzione di un prototipo viene inviata una mail al richiedente e all'owner in cui si notifica la chiusura della richiesta. Se l'owner non ha completato tutte le operazioni necessarie dovrà contattare la pianificazione per ottenere una nuova data di produzione che dovrà inserire nel form.
Versione	

3.2.4 Validazione di prodotto e validazione di processo

Id	3.2.4
Titolo	Validazione di prodotto e validazione di processo
Descr.	L'owner dovrà compilare due form: 1. Il primo form riguarda la validazione di prodotto, l'owner dovrà inserire le specifiche di riferimento, eventuali note ed un menù in cui è possibile scegliere tra superato e non superato; 3. Il secondo form riguarda la validazione di processo, questo form sarà facoltativo; nel caso fosse necessario compilarlo si dovrà inserire il rendimento globale ed eventuali informazioni aggiuntive e scegliere da un menù tra superato e non superato; Andrà anche inserito se è stata eseguita la formazione degli operatori di linea e se è stata eseguita la formazione degli operatori del reparto riparazioni. In

	caso positivo va indicata anche la data in cui sono state eseguite. Verrà inviata una mail agli operatori del reparto riparazioni nel caso venga indicato che è stata eseguita loro la formazione. Viene inviata una mail al VSL per segnalare che è necessaria la sua validazione finale.
Versione	

3.2.5 Ricerca nello storico

Id	3.2.5
Titolo	Ricerca nello storico
Descr.	Attraverso un form apposito sarà possibile effettuare una ricerca nello storico, secondo parametri impostabili. Al di sotto appariranno i risultati e sarà possibile accedere alla scheda della preserie/prototipo selezionata.
Versione	

3.3 Pianificazione

3.3.1 Pianificazione

Id	3.3.1
Titolo	Pianificazione della preserie/prototipo
Descr.	Nel form andrà inserito obbligatoriamente il referente della pianificazione, l'ODL e la data di produzione prevista. Verrà verificato che la data inserita sia successiva alla data di disponibilità del materiale. Dovrà inoltre indicare se è necessaria la presenza dell'owner al momento della produzione e se è necessaria la validazione di processo. La pianificazione avrà inoltre la possibilità di indicare un nuovo VSL che andrà a sostituire quello precedentemente indicato. In questo caso verrà inviata una mail all'owner, al vecchio ed al nuovo VSL per notificare la modifica. In ogni caso verrà inviata una mail all'owner per notificare la pianificazione dell'ODL.
Versione	

3.3.2 Ricerca nello storico

Id	3.3.2
Titolo	Ricerca nello storico
Descr.	Attraverso un form apposito sarà possibile effettuare una ricerca nello storico, secondo parametri impostabili. Al di sotto appariranno i risultati e sarà possibile accedere alla scheda della preserie/prototipo selezionata.
Versione	

3.4 VSL

3.4.1 Validazione finale

Id	3.4.1
Titolo	Validazione finale della preserie
Descr.	Sarà presente un menù in cui sarà possibile scegliere fra valida/non valida/attività correttive ed un campo note in cui sarà possibile giustificare la scelta. In caso il VSL indichi delle attività correttive indicherà anche una data massima entro cui portarle a termine. In caso di attività correttive viene inviata una mail all'owner in cui sono inserite le richieste del VSL e la data entro cui eseguirle. Prima di tale data verrà inviato un reminder all'owner e al VSL. Al secondo tentativo di validazione sarà possibile soltanto validare/non validare.

Bibliografia

[1] 830-1998 - ieee recommended practice for software requirements specifications. http://standards.ieee.org/findstds/standard/830-1998.html.

[2] Pagina rigurdante le sql *Injection* dell'*OWASP*. https://www.owasp.org/index.php/Guide_to_SQL_Injection.

[3] Pagina wikipedia di outsourcing. http://it.wikipedia.org/wiki/Esternalizzazione.

[4] Pagina wikipedia di prototipo. http://it.wikipedia.org/wiki/Prototipo.

[5] Sito della lean enterprise institute. http://www.lean.org/.

[6] Sito di carel industries s.p.a. http://www.carel.com/.

[7] Sito di leannovator. http://www.leannovator.com/.

[8] Ramez A. Elmasri and Shamkant B. Navathe. *Sistemi di basi di dati*. Pearson, 2007.

[9] Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Devika Gollapudi, Kim Haase, William Markito Oliveira, Chinmayee, and Srivathsa. *The Java EE 6 Tutorial*, 2012.

[10] Kito D. Mann. *JavaServer Faces in Action*. Manning, 2005.

[11] Debu Panda, Reza Rahman, and Derek Lane. *EJB 3 in Action*. Manning, 2006.

[12] Mike Rother and John Shook. *Learning to See*. Lean Enterprise Institute, 1998.

[13] Peter Walsh. Value-stream mapping for the office and services.

Elenco delle figure

Elenco delle tabelle

Ringraziamenti

Ringrazio l'azienda *Carel* per avermi dato l'opportunità di svolgere questa tesi. In modo particolare il mio tutor aziendale Abramo per il suo continuo supporto e l'abilità con cui è riuscito ad introdurmi ad argomenti per lo più nuovi e che, insieme gli altri compagni d'"isola" Federico e Patrick, ha reso la mia esperienza in *Carel* una "scuola di vita". Ci tengo a ringraziare anche gli altri compagni d'ufficio Beppe, Federica, Lisa, Marco e Mauro che hanno reso liete le mie giornate.

Ringrazio il mio relatore Prof. Carlo Ferrari per la grande libertà datami e per gli utili suggerimenti.

Un immenso grazie ai miei genitori che mi hanno sempre sostenuto nell'arco di questa mia esperienza, senza di loro non avrei potuto raggiungere questo obiettivo. Ringrazio anche Deborah per la sua pazienza e comprensione anche nei periodo in cui ero "pesante" e per aver scritto sotto mia dettatura parte di questo elaborato...maledetta spalla.

Un grazie ai miei compagni di università Zan, Gobbo, Dalla, Boc, Chessa e Franz per aver reso i miei giorni a Padova sempre gioiosi e mai scontati. Senza di voi non sarebbe stato lo stesso. Non vedo già l'ora che Zan organizzi la prossima reunion.

Un ringraziamento speciale a tutti i miei amici che sono stati ad ascoltarmi anche quando magari li assillavo con questioni universitarie, che mi hanno accompagnato nelle ore di studio in biblioteca e che per fortuna mi ricordano sempre quanto è bello stare insieme e divertirsi.

Infine, ma non da meno, ringrazio i miei parenti per l'affetto che mi hanno sempre dimostrato.