Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Triennale in Ingegneria Informatica

Progettazione e realizzazione di un sistema per la gestione di articoli industriali a carattere tecnico-produttivo basato su architettura Hibernate

Laureando:
Daniel Zilio

Relatore:
Prof. Giorgio Maria
Di Nunzio

Anno accademico 2012/2013

Sommario

L'applicazione che verrà di seguito descritta riguarda il settore aziendale che si occupa della programmazione della produzione industriale, finalizzato cioè alla realizzazione fisica dell'articolo prodotto.

Il sistema è stato realizzato utilizzando come modello di riferimento una moderna occhialeria, e gli esempi inseriti si rivolgono a questo, ma è estendibile anche a quelle realtà aziendali che realizzano prodotti finiti e/o semilavorati.

Si è scelto per realizzare tale applicazione il noto linguaggio Java. In generale quando si usa realizzare un'applicazione che interagisce con un database i due moduli vengono realizzati separatamente e l'interazione avviene attraverso l'API JDBC. Qui invece si è deciso di utilizzare il *framework* Hibernate che unisce i due aspetti della programmazione nel modo, e coi vantaggi, che verranno in seguito descritti.

Indice

Capitolo 1

Introduzione

Stiamo vivendo in un'era in cui l'ambiente industriale è in continua evoluzione. Mai come ora le aziende sono costrette a perseguire un'opera di miglioramento continuo. Produttività, efficienza, innovazione sono le parole che più frequentemente riempiono il quotidiano, ed è in questa direzione che le imprese che vogliono essere competitive danno sempre più rilevanza alla funzione aziendale *sistemi informativi*, la quale, come è noto, ha il compito di analizzare e soddisfare il *fabbisogno di informazioni* e di rendere accessibile e utilizzabile al meglio tale bagaglio mediante il processo di informatizzazione dei dati che si realizza con la creazione e la gestione del *sistema informatico*.
Il sistema che si è scelto di realizzare ha lo scopo di esporre l'intero processo produttivo necessario per produrre un articolo industriale.
Esso mostra, a partire da un bene fabbricato, tutti i modelli che lo caratterizzano, siano essi commercializzati, in fase di prototipazione o ormai in obsolescenza. Per ognuno di questi è poi possibile vederne l'intero processo produttivo, ovvero tutte le lavorazioni fisiche, inserite nel loro contesto di centro di lavoro, che dovranno essere eseguite per avere l'articolo finito pronto per la commercializzazione. Per tutte le lavorazioni che si svolgono con l'ausilio di una macchina, l'applicazione provvede a fornire i relativi parametri di configurazione, che possono essere inerenti ad una delle quattro macro categorie principali (asportazione mediante utensili, trattamento galvanico, stampaggio, verniciatura) o caratteristici di una configurazione indipendente.
Tale sistema permette di ottenere numerosi vantaggi in termini di produttività in quanto riduce drasticamente le attività manuali di programmazione delle operazioni, velocizza il flusso delle informazioni, minimizza le possibilità di errore, facilità l'assistenza post vendita, permette una certa pianificazione delle materie da avere disponibili per le lavorazioni.
Tutto questo può essere integrato in una ancor più grande piattaforma che tenga conto anche degli altri aspetti che caratterizzano la produzione, quali ad esempio il magazzinaggio, la valutazione dei tempi e metodi, l'uso delle risorse umane.
Si è scelto per realizzare tale applicazione il *framework* Hibernate che unisce i due aspetti della programmazione nel modo, e coi vantaggi, che verranno in seguito descritti.

1.1 Presentazione del progetto

Lo sviluppo del sistema di gestione si articola in due parti principali. Nella prima si realizza la progettazione della base di dati in questione mediante una procedura standard definita da:

1. Analisi dei requisiti (cap. 2.1): ove vengono raccolte le informazioni dal committente e si definiscono le linee guida generali che la base di dati dovrà rispettare.

2. Progettazione concettuale (cap. 2.2): si rappresenta attraverso un modello formale ad alto livello i dati della realtà d'interesse analizzata. Si utilizzeranno i concetti di modellazione del modello *entity-relationship (ER)*.

3. Progettazione logica (cap. 2.3): si rappresentano i dati raccolti come una collezione di *relazioni*.

4. Progettazione fisica (cap. 2.4): i dati vengono descritti attraverso le strutture di un specifico *DBMS*.

Eseguiti i quattro punti precedenti si passa alla seconda fase, consistente nella realizzazione dell'applicazione attraverso i seguenti passaggi:

1. Definizione e realizzazione delle classi persistenti con relative mappature e realizzazioni fisiche (cap. 3.4).

2. Creazione di uno strato di accesso mediante pattern DAO (cap. 3.5).

1.2 Strumenti utilizzati

Di seguito verranno brevemente introdotti gli strumenti che sono stati utilizzati per la realizzazione di questo progetto.

- **Microsoft Visio 2010**[1]. Questo software è stato utilizzato per la realizzazione degli schemi della progettazione concettuale della base di dati. E' sotto licenza Microsoft per i sistemi operativi Windows.

- **PostgreSQL**. Questo software[2] nella versione 9.2 è stato utilizzato per la creazione della base di dati. E' un DBMS relazionale open-source.

- **PgAdmin 3**. Questa interfaccia[3] è stata utilizzata per verificare in corso d'opera la correttezza delle mappatture. E' un software open-source realizzato appositamente per la gestione dei database PostgreSQL.

[1] `http://it.wikipedia.org/wiki/Visio`
[2] `http://www.postgresql.org/`
[3] `http://www.pgadmin.org/`

- **Geany 0.21**. Editor di testo open-source utilizzato come supporto nella realizzazione del progetto[4].
- **Eclipse**. Ambiente di sviluppo multi-linguaggio e multi-piattaforma open-source[5]. E' stato utilizzato per la realizzazione delle classi persistenti e per i relativi file di mappatura. La versione utilizzata è la *Juno*.
- **Hibernate 4.1.8**. Piattaforma middleware open source per lo sviluppo di applicazioni Java atta a gestire la persistenza dei dati su database relazionali[6].
- **TeX Live 2012/Debian, Gostscript, TeXmaker 3.4**. Per la stesura della tesi è stato utilizzato LaTeX. In particolare la distribuzione usata è quella TeX Live 2012/Debian. Come editor è stato scelto Texmaker[7] nella sua versione 3.2.

[4] `http://www.geany.org/`
[5] `http://www.eclipse.org/juno/`
[6] `http://www.hibernate.org/`
[7] `http://en.wikipedia.org/wiki/Texmaker`

Capitolo 2

Progettazione della base di dati

2.1 Analisi dei requisiti

La progettazione di una base di dati comincia con la raccolta e l'analisi di tutte le informazioni necessarie per la completa e corretta realizzazione di ciò che il fruitore richiede, mediante un'interrogazione mirata a comprenderne tutte le sfaccettature. Il risultato è un'insieme di dati completi ma informali che andranno successivamente schematizzati. Di notevole importanza è anche la totale comprensione di come tali dati verranno elaborati, ovvero quali operazioni si intenderanno fare su di essi; tipicamente le principali sono quelle di interrogazione e aggiornamento.
Per la corretta riuscita dell'applicazione questa fase è essenziale in quanto permette di realizzare l'applicazione su misura rispondendo a tutte le esigenze dei futuri utenti.

2.1.1 Caratteristiche e requisiti

Alla base del sistema c'è l'articolo che rappresenta ciò che l'azienda produce da un punto di vista generico (es. *Occhiale*). A partire da un articolo poi si devono ricavare tutte le generiche istanze del medesimo ovvero i modelli. Per ogni modello è richiesto di memorizzare un identificatore dello status per indicarne la situazione progettuale (es. *Prototipo*, *In produzione*, ecc.) e delle immagini in due e tre dimensioni per un'identificazione visiva.
Scelto un modello si visualizzeranno a cascata tutte le lavorazioni necessarie alla sua realizzazione pratica, inserite nel loro contesto di centro di lavoro [1].

Una singola lavorazione rappresenta la reale esecuzione di una operazione fisica. Essa, oltre ad essere caratterizzata dal modello e dal centro di lavoro specifico, deve contenere un indicatore del tempo di ciclo che rappresenti la durata necessaria per la completa realizzazione dell'operazione e due immagini progettuali quotate, rispettivamente in due e tre dimensioni. Dovrà poi essere possibile elencare tutti

[1] Macro operazione dalla quale si possono evincere le informazioni relative alle postazioni fisiche nelle quali vengono effettuate le operazioni del processo produttivo (es *Finissaggio*).

i materiali utilizzati.
Per ogni macchina [2] si devono catalogare il codice cespite [3] il modello e la categoria di appartenenza (es. *Fresatrice*, *Impianto Galvanico*, ecc).
Una necessità molto importante che l'applicazione dovrà soddisfare è il reperimento delle informazioni relative a come una macchina dovrà essere configurata per la lavorazione. Una configurazione potrà essere a se stante o appartenente ad una di queste macro categorie:

1. Presettaggio: ove si rappresenta ciò che comunemente si identifica con la *scheda di presettaggio* ovvero la configurazione dell'*utensile* per le operazioni di asportazione. I parametri richiesti sono: l'utensile utilizzato (con in particolare le informazioni relative al numero di taglienti, la lunghezza del tagliente, il diametro, il materiale di costruzione e la configurazione DX-SX [4]), il cono porta-utensile e la relativa pinza, e uno schema visivo quotato in due dimensioni.

2. Trattamento galvanico: qui le informazioni richieste per ogni singolo bagno: la tensione elettrica di applicazione, il tempo di trattamento, l'intensità di corrente, la superficie da trattare, la temperatura.

3. Stampaggio: dovrà essere possibile rappresentare sia stampaggi che utilizzano materiali plastici via iniezione (di conseguenza specificando il tempo, la temperatura e la pressione di iniezione) sia quelli metallici specificando la pressione e il tempo. Inoltre andrà specificato lo stampo modello via via utilizzato.

4. Verniciatura: le informazioni richieste sono i componenti utilizzati (in generale aerografi e ugelli), e tre tempi di lavoro: di avanzamento, di passivazione, di essicazione.

E' importante precisare che una lavorazione può essere realizzata utilizzando macchine diverse con configurazioni diverse ma sempre in singole occorrenze, con l'unica eccezione del trattamento galvanico che può presentare più configurazioni per la stessa coppia lavorazione-macchina (una per ogni singolo bagno appartenente all'impianto).

Per tutti gli strumenti utilizzati nelle varie categorie dovrà essere presente l'eventuale cespite e la categoria di appartenenza.
Per fornire supporto alle attività di assistenza post-vendita dovrà essere possibile reperire per un singolo semi-lavorato di un particolare modello le lavorazioni necessarie a realizzarlo (in caso di avvenuta obsolescenza del modello in questione).

[2] per la definizione si veda la Nuova Direttiva Macchine 2006/42/CE Articolo 2 punto a): `http://www.ispesl.it/linee_guida/tecniche/LGDirettivaMacchine.pdf`

[3] `http://it.wikipedia.org/wiki/Cespite`

[4] ovvero con taglienti destrosi o sinistrorsi.

Si è scelto inoltre di fornire la possibilità di inserire per le lavorazioni e le configurazioni ulteriori parametri che verranno utilizzati a seconda della situazione specifica d'uso, questo per fornire una maggiore fruibilità del prodotto.

Inoltre per tutte le componenti prese in esame vi è la possibilità di compilare delle note testuali esplicative e di utilizzare come indicatore univoco un codice libero.

2.1.2 Operazioni di inserimento

Le operazioni di inserimento riguardano tutte le entità e tutti i loro attributi. Esse avranno la loro massima frequenza in fase di inizializzazione della base di dati e ogniqualvolta che verrà inserito nel processo produttivo un nuovo modello, situazione fortemente dipendente dalla tipologia di azienda. Ulteriore inserimenti si verificheranno all'atto di acquisto di nuove macchine e nuovi relativi componenti d'uso, situazione che comporta inoltre la stesura di nuove configurazioni macchina.

2.1.3 Operazioni di interrogazione

La frequenza delle operazioni sotto riportate è indicativa e stimata sulla base di informazioni ricavate da diversi possibili utilizzatori.
Legenda: MF = Molto frequente, F = Frequente, R = Raro, MR = Molto raro.

Operazioni relative all'articolo prodotto

Operazione	**Freq.**
Tutti gli attributi di un articolo	F
Tutti i modelli che caratterizzano un articolo	MF

Operazioni relative al modello di articolo

Operazione	**Freq.**
Tutti gli attributi di un modello	MF
Tutte le lavorazioni che realizzano un modello	MF
Conteggio del tempo teorico relativo per la realizzazione di un articolo	MF
Conteggio del numero totale di lavorazioni che realizzano un modello	F

Operazioni relative al centro di lavoro

Operazione	Freq.
Tutti gli attributi di un centro di lavoro	MF
Tutte le lavorazioni appartenenti ad un centro di lavoro	MF

Operazioni relative alla lavorazione

Operazione	Freq.
Tutti gli attributi di una lavorazione	MF
Tutti i parametri aggiuntivi di una lavorazione	MF
La macchina e la relativa configurazione inerente ad un lavorazione	MF
Tutti i materiali utilizzati in una lavorazione	MF

Operazioni relative al materiale

Operazione	Freq.
Visualizzazione della categoria di un materiale	R
Visualizzazione delle eventuali lavorazioni che producono il materiale come semilavorato	F
Tutte le lavorazione che utilizzano dato un materiale	F

Operazioni relative alla macchina

Operazione	Freq.
Tutti gli attributi di una macchina	R
Tutte le lavorazione che utilizzano una determinata macchina	R
Conteggio di tutte le macchine per categoria	MR

Operazioni relative alla configurazione

Operazione	Freq.
Visualizzazione di tutti gli attributi di una configurazione	MF
Visualizzazione di tutti i componenti di supporto di una configurazione	MF

Operazioni relative ai componenti

Operazione	Freq.
Tutti gli attributi di un componente	R
Tutte le configurazioni che utilizzano un determinato componente	R
Conteggio di tutti i componenti per categoria	MR

2.2 Progettazione dello schema relazionale

Una volta terminata l'analisi dei requisiti il passo successivo consiste nel rappresentare i dati raccolti attraverso un *modello concettuale*, che per definizione è formale, ad alto livello di astrazione e indipendente dal DMBS che si andrà ad utilizzare in seguito.
Nel caso in esame il modello concettuale utilizzato è lo schema *Entity-Relationship* [5].
Esso è caratterizzato dai seguenti costrutti:

1. **Entità**: rappresentazione di una classe di oggetti con caratteristiche comuni ai fini dell'applicazione di interesse. Nello schema concettuale l'entità viene indicata in un rettangolo con all'interno il suo nome univoco.

2. **Associazione**: rappresentazione di un legame tra una o più entità (possono sussistere legami ad anello). La sua rappresentazione grafica è data da un rombo contenente il proprio nome, che in generale è un verbo, un sostantivo o una sigla.

3. **Attributo**: le entità e le associazioni possono essere descritte usando una serie di attributi. Tutti gli oggetti della stessa classe, entità o associazione, hanno gli stessi attributi. Gli attributi vengono indicati con un pallino collegato all'entità o all'associazione a cui appartengono. Il pallino può essere vuoto se indica un attributo con valori che possono ripetersi tra un'istanza ed un'altra di una classe, oppure pieno in caso contrario.

[5] `http://it.wikipedia.org/wiki/Modello_ER`

2.2.1 Entità

Di seguito verranno descritte le entità definite nel modello in esame.

Nome	Descrizione	Attributi	Ident.
Articolo	Il bene che l'azienda produce da un punto di vista generico	Codice, Descrizione	Codice
Modello	Istanza di un articolo	Codice, Nome, Status, Descrizione, Immagine 2D, Immagine 3D	Codice
Centro di Lavoro	Rappresenta un insieme contenente una o più lavorazioni	Codice, Manodopera impiegata, Livello	Codice
Lavorazione	Rappresenta la singola operazione fisica	Codice, Tempo ciclo, Immagine 2D, Immagine 3D, Note	Codice
Materiale	Ha il duplice significato di materia primaria per la lavorazione e semilavorato	Codice, Categoria	Codice
Macchina	Rappresenta l'unità meccanica di supporto alla lavorazione	Codice, Cespite, Modello, Categoria	Codice
Configurazione	Rappresenta la generica configurazione macchina per lo svolgimento di una lavorazione	Codice, Categoria, Note	Codice
Parametro	Rappresenta un'entità di supporto per eventuali parametri specifici inerenti alle lavorazioni, alle configurazioni o ai componenti di supporto	Codice, Unità misura, Categoria	Codice
Parametro di Lavorazione	Rappresenta un'entità di supporto per eventuali parametri specifici inerenti alle lavorazioni	*Nessuno*	*Nessuno*
Parametro di Configurazione	Rappresenta un'entità di supporto per eventuali parametri specifici inerenti alle configurazioni	*Nessuno*	*Nessuno*
Presettaggio	Rappresenta una scheda di presettaggio utensile	Numero taglienti, Diametro tagliente, Lunghezza tagliente, Configurazione Dx-Sx, Materiale, Immagine 2D	*Nessuno*
Trattamento galvanico	Rappresenta la configurazione necessaria per l'uso di una cella galvanica per un singolo trattamento galvanico	Tensione, Tempo, Intensità corrente, Superficie, Temperatura	*Nessuno*

Stampaggio	Rappresenta la configurazione per una macchina che utilizza stampi per iniezione e/o modellazione	Tempo, Temperatura iniezione, Pressione	*Nessuno*
Verniciatura	Rappresenta la configurazione per una cabina di verniciatura	Tempo avanzamento, Tempo essiccazione, Tempo passivazione	*Nessuno*
Componenti	Rappresentano i generici componenti utilizzati per le lavorazioni mediante macchina	Codice, Descrizione, Cespite	Codice

2.2.2 Associazioni

Di seguito verranno elencate e descritte le associazioni presenti nel modello.

Nome	**Entità coinvolte**	**Descrizione**	**Attrib.**
Realizzato	Articolo (0,N), Modello (0,1)	Questa associazione esprime che un articolo può avere più modelli che lo caratterizzano mentre un modello è relativo ad un solo eventuale articolo	*Nessuno*
Realizza	Modello (0,N), Centro di Lavoro (0,N), Lavorazione (1,1)	Questa associazione ternaria sta ad indicare che una lavorazione non è definita al di fuori del contesto di un modello e/o di un centro di lavoro	*Nessuno*
Adopera	Lavorazione (0,N), Materiale (0,N)	Questa associazione rappresenta i materiali utilizzati in una lavorazione. La duplice molteplicità indica che è possibile che un materiale sia utilizzato in differenti lavorazioni	Quantità
Prodotto	Materiale (0,N), Lavorazione (0,N)	Sebbene questa associazione coinvolga le stesse entità della precedente in questo caso si vuole rappresentare il materiale come un semi lavorato che può essere realizzato attraverso delle lavorazioni	*Nessuno*
Segue	Lavorazione (0,N), Lavorazione (0,N)	Questa associazione serve a rappresentare un ordine di esecuzione tra lavorazioni differenti	Note

Utilizza	Lavorazione (0,N) Macchina (0,N), Configurazione (0,N)	Questa associazione ternaria indica per ogni lavorazione che ne necessiti, quali macchine e quali configurazioni relative sono necessarie per il corretto svolgimento della medesima	Software, Note
Descritta	Lavorazione (0,N), Parametri di Lavorazione (0,N)	Questa associazione serve per indicare dei parametri aggiuntivi per una lavorazione	Valore
Possiede	Configurazione(0,N), Parametri di Configurazione (0,N)	Questa associazione serve per indicare dei parametri aggiuntivi per una configurazione macchina	Valore, Tolleranze
Usa	Configurazione (0,N), Componente(0,N)	Questa associazione rappresenta i componenti utilizzati in una configurazione macchina	Numero, Descrizione, Valore

2.2.3 Scelte progettuali e regole di vincolo

La scelta progettuale più in evidenza è quella di aver reso l'entità Lavorazione debole rispetto a Modello e Centro di Lavoro: questo deriva dal fatto che ogni lavorazione è a se stante e che, presa singolarmente, non ha un preciso significato ma va vista all'interno di una visione globale. La scelta poi di avere una classe che rappresenti gli attributi di alcune delle entità è conseguenza della scelta di rendere la struttura il più flessibile possibile permettendo così di adattarsi ad un ampio ventaglio di produzioni industriali.

E' necessario introdurre dei vincoli inter-relazionali per garantire l'integrità dei dati:

- La possibilità di associare più macchine ad una stessa lavorazione è sinonimo del fatto che tale lavorazione può utilizzare disgiuntamente tali mezzi.

- La possibilità di associare più configurazioni ad una stessa coppia macchina - lavorazione è sinonimo del fatto che tale coppia può utilizzare disgiuntamente tale configurazione.

- Una lavorazione facente uso di un impianto galvanico può avere diverse configurazioni di tipo *trattamento galvanico*.

- Una configurazione non può appartenere ad una categoria di macchina diversa da quella prevista.

- Una configurazione non può essere associata ad un componente appartenente ad una categoria diversa.

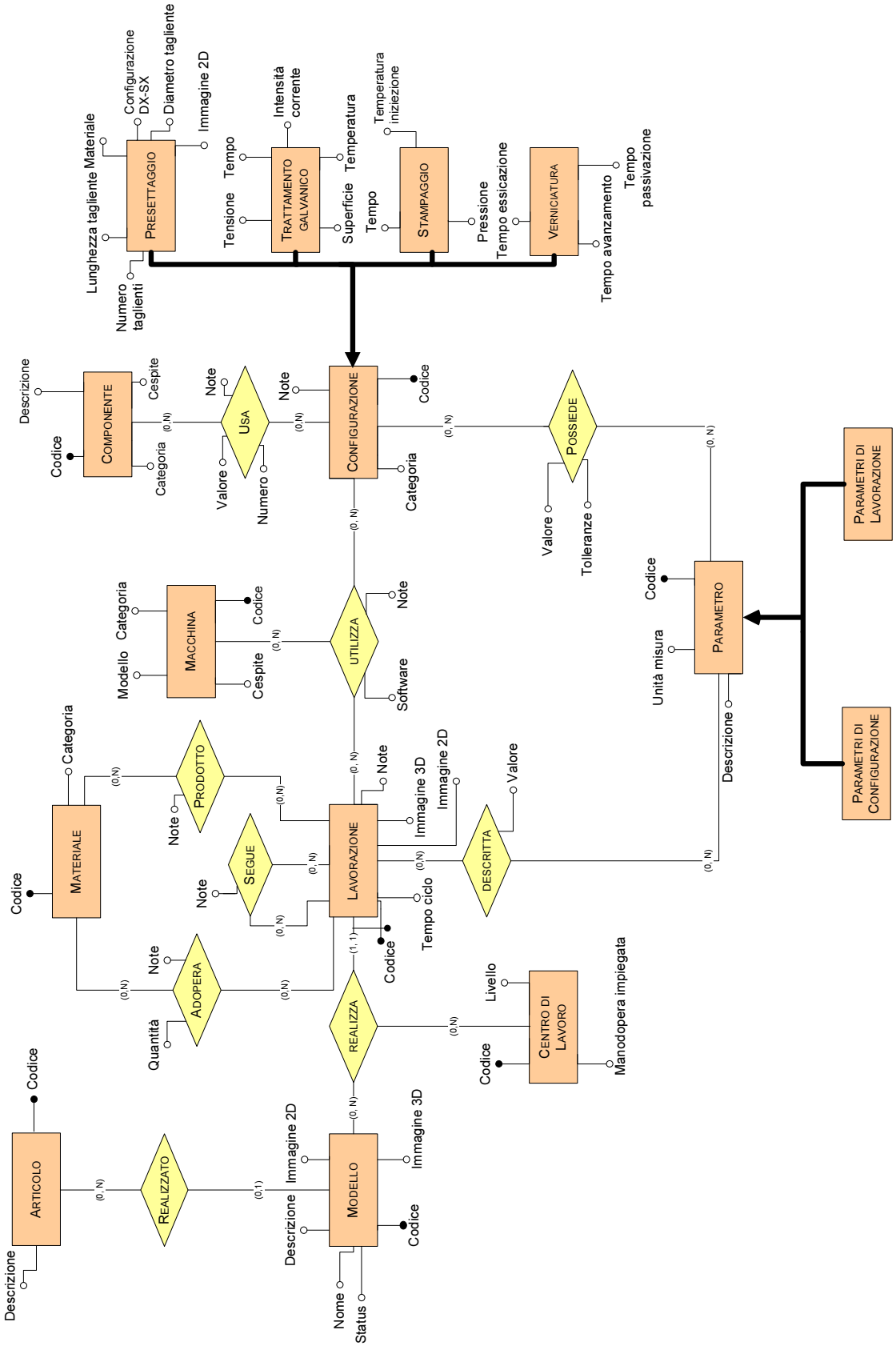

Figura 2.1: Schema ER Generalizzato

2.3 Progettazione logica

La progettazione logica ha come obiettivo la definizione di uno schema logico che rappresenti in modo fedele ed efficiente le informazioni contenute nel modello concettuale.
Tali dati sono rappresentati facendo uso del concetto matematico di *relazione* (o *tabella*) e come tali manipolati con gli operatori dell'algebra relazionale [6].
La prima operazione da eseguire consiste nella ristrutturazione dello schema ER, ove le generalizzazioni/specializzazioni e gli attributi multi-valore vengono eliminati (in quanto non rappresentabili direttamente) e riprodotti mediante altre entità o associazioni. La seconda invece consiste nella effettiva creazione dello *schema logico* equivalente, ovvero la rappresentazione senza semplificazioni dello schema concettuale ristrutturato.

2.3.1 ER ristrutturato

A seguito di una analisi che coglieva più fattori è stato scelto di risolvere la specializzazione che legava un Parametro con tutte le sue sotto-entità figlie accorpandole all'interno dell'identità padre, inserendo un attributo che ne distinguesse la categoria di appartenenza. Questa operazione non è stata possibile invece per quanto riguarda la specializzazione che considera le diverse possibilità di configurazione macchina (Presettaggio, Trattamento Galvanico, Stampaggio, Verniciatura) perché ognuna di queste presenta degli attributi unici caratteristici per ogni tipologia di configurazione. Si è scelto quindi di mantenere tali entità rendendole però deboli rispetto all'entità padre Configurazione.

[6] `http://it.wikipedia.org/wiki/Modello_relazionale`

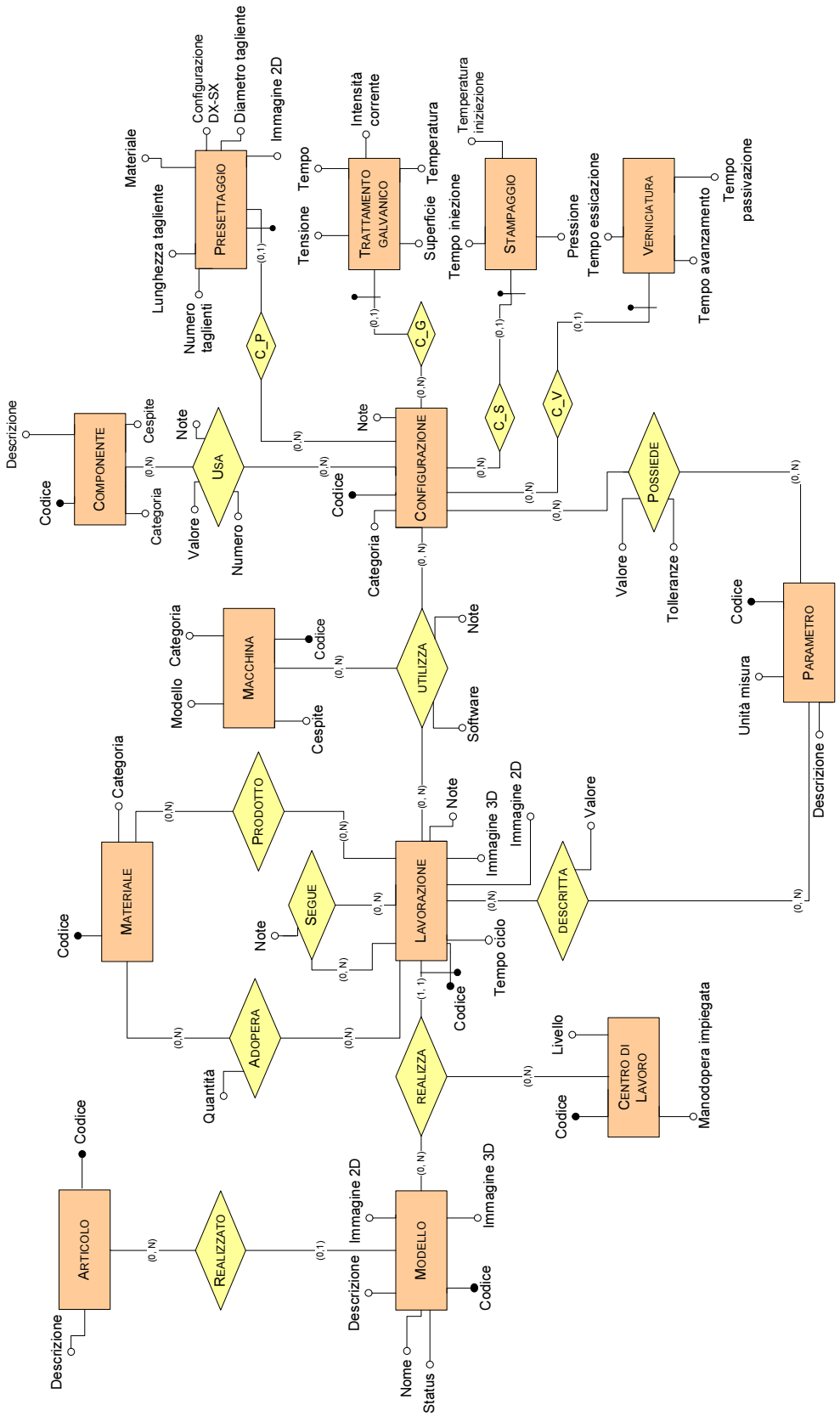

Figura 2.2: Schema ER Ristrutturato

2.3.2 Regole di vincolo aggiuntive

A seguito della ristrutturazione bisogna tener conto del seguente vincolo:

- Un PARAMETRO deve sempre essere associato in base alla categoria di appartenenza.

2.3.3 Schema Relazionale

Di seguito è riportato lo schema relazionale prodotto della traduzione dello schema ER generalizzato.

Legenda:
NOME = indica il nome della tabella
attributo = indica la chiave primaria della tabella (se più attributi sono sottolineati si tratta di una chiave composita)
attributo = indica un vincolo inter-referenziale

ARTICOLO (codice, descrizione)

MODELLO (codice, nome, status, descrizione, immagine_2D, immagine_3D, *articolo*)

CENTRO_DI_LAVORO (codice, livello, manodopera_impiegata)

LAVORAZIONE (codice, *modello*, *centro_di_lavoro*, tempo_ciclo, note, immagine_2D, immagine_3D)

MATERIALE (codice, categoria)

ADOPERA (*lavorazione*, *materiale*, note)

PRODOTTO (*materiale*, *lavorazione*, note)

SEGUE (*lavorazione*, *lavorazione*, note)

MACCHINA (codice, modello, categoria, cespite)

CONFIGURAZIONE (codice, note, categoria)

PRESETTAGGIO(, numero_taglienti, lunghezza_tagliente, diametro_tagliente, configurazione_DxSx, materiale, immagine_2d)

TRATTAMENTO_GALVANICO (*configurazione*, tensione, intesita_corrente, tempo, superficie, temperatura)

STAMPAGGIO (*configurazione*, tempo, temperatura_iniezione, pressione)

VERNICIATURA (*configurazione*, tempo_avanzamento, tempo_essicazione, tempo_passivazione)

PARAMETRO (codice, unita_misura, descrizione)

UTILIZZA (*lavorazione*, *macchina*, *configurazione*, software, note)

COMPONENTE (codice, descrizione, categoria, cespite)

USA (*configurazione*, *componente*, descrizione, numero, note)

POSSIEDE (*configurazione*, *parametro*, valore, tolleranze)

DESCRITTA (*lavorazione*, *parametro*, valore)

2.4 Progettazione fisica

L'ultima fase di progettazione della base di dati è quella fisica. A partire dallo schema logico e dai vincoli richiesti si produce lo schema fisico utilizzando il linguaggio SQL.
I passi da seguire per implementare ogni singola relazione dello schema logico sono i seguenti:

1. Definizione di una tabella avente la stessa nomenclatura della relazione che la rappresenta.
2. Elencazione dei rispettivi attributi assegnando a ciascuno il rispettivo nome e un dominio idoneo.
3. Specificazione della chiave primaria .
4. Definizione dei vincoli intra-relazionali (ad esempio vincoli *NOT NULL*, *UNIQUE*, ecc).
5. Definizione dei vincoli inter-relazionali, ovvero i vincoli di integrità referenziale *FOREIGN KEY*.

Il codice SQL prodotto da questa fase non verrà direttamente implementato nel DBMS ma servirà come strumento di paragone rispetto a quello che verrà autogenerato dall'apposito strumento.
In appendice il codice relativo (si veda 5.1).

Capitolo 3

Architettura Hibernate

3.1 Introduzione

Nell'ambito informatico il termine *persistenza* viene utilizzato per indicare la possibilità dei dati di sopravvivere anche dopo la cessazione del programma che li ha creati, proprietà non immediata in quanto all'atto dell'esecuzione vengono salvati nella memoria RAM, che per definizione è volatile. Il salvataggio dei dati si può ottenere utilizzando principalmente due metodi, ovvero mediante file system oppure attraverso un DBMS (Database Management System). Il secondo mezzo è considerato migliore per applicazioni che lavorano con grandi moli di dati in quanto fornisce un buon controllo sulla ridondanza, una riduzione dei tempi per lo sviluppo di applicazioni, flessibilità d'uso, funzioni di backup, assistenza e ripristino, e molti altri vantaggi.

Come precedentemente indicato nel Sommario, nel linguaggio Java per accedere ai database (e di conseguenza permettere il recupero e il salvataggio degli oggetti) si usa ricorrere alle API [1] JDBC (Java DataBase Connectivity)[2], che svolgono il compito di collegare l'applicazione alla base di dati. Per fare ciò il programmatore deve aggiungere alle classi dei metodi che implementano le cosiddette operazioni CRUD (Create, Read, Update, Delete). Questa tecnica occupa mediamente però un terzo del tempo totale per lo sviluppo dell'intera applicazione, dovuto alla scrittura del codice che implementa il database ma soprattutto per il mantenimento delle connessioni.
Un'ulteriore punto di debolezza è dato dalla differenza tra la rappresentazione dei dati nel mondo *object-oriented* e quella del mondo relazionale.

[1] `http://it.wikipedia.org/wiki/Application_programming_interface`
[2] `http://it.wikipedia.org/wiki/JDBC`

3.2 Object-relational mapping

Per cercare di ridurre tale inefficienza sono state create delle tecniche di programmazione atte a fornire delle interfacce per implementare i dati dal modello relazionale al modello ad oggetti e viceversa, snellendo di fatto notevolmente il lavoro svolto dal programmatore per rendere gli oggetti persistenti. La principale di queste tecniche è la Object Relation Mapping (ORM).
Attraverso questo mezzo ogni oggetto viene reso persistente nel database tramite l'inserimento di nuovi record i cui campi contengono i valori degli attributi dell'oggetto stesso. Si viene quindi a creare la relazione tra oggetto Java e tabella SQL.
Una soluzione ORM consiste di quattro componenti caratteristici [3]:

1. Una API per eseguire le operazioni CRUD sugli oggetti delle classi del modello.

2. Un linguaggio o una API per specificare query che fanno riferimento alle classi e alle proprietà delle classi.

3. Uno strumento per la specifica del mapping mediante metadata.

4. Una tecnica per l'implementazione di ORM per interagire con oggetti. transazionali al fine di eseguire funzioni di ottimizzazione.

L'associazione fisica tra la classe e la tabella viene ottenuta mediante l'utilizzo di file di descrizione (detti file di mapping), in cui si specificano le modalità di conversione tra gli attributi dell'oggetto e i campi della tabella.
In definitiva i principali vantaggi derivanti dall'uso di una soluzione ORM sono:

- Snellimento della parte di codice riservata alla persistenza dei dati.

- Maggiore facilità per quanto riguarda la manutenzione del codice grazie alla separazione tra il modello ad oggetti e il modello relazionale.

- Performance più efficienti grazie alle numerose opzioni di ottimizzazione presenti.

- Elevata portabilità rispetto alla tecnologia DBMS utilizzata.

L'esempio più noto di ORM per il linguaggio Java è Hibernate i cui dettagli saranno descritti nella sezione successiva.

3.3 Il framework Hibernate

Il progetto è stato sviluppato nel 2001 da Gaving King e Christian Bauer e sono rispettivamente del 2003 e 2010 le versioni Hibernate 2 e Hibernate 3.x. Attualmente l'ultima versione rilasciata è la 4.2.4 disponibile da Agosto 2013. Esso è

[3] `http://www-db.deis.unibo.it/courses/TW/PDF/5.02.DAO-ORM-Hibernate.pdf`

inserito all'interno del contesto dell'application server open source Jboss [4].
Hibernate è una piattaforma middleware open source distribuito sotto licenza LGPL[5]. Fornisce supporto non solo per quanto riguarda la mappatura di un modello di dominio orientato agli oggetti in un classico database relazionale, ma anche per quanto riguarda il reperimento dei dati da esso, ovvero la gestione delle *query*.

3.3.1 Approcci di sviluppo

Essenzialmente l'utilizzo di Hibernate permette tre approcci di programmazione:

- **Top Down**. Si parte da un diagramma delle classi di dominio e dalla sua implementazione in Java e si ha completa libertà rispetto allo schema della base di dati. Si specificano poi i file XML con i mapping ed, eventualmente, si può usare in particolare lo strumento di Hibernate *hbm2ddl*, per generare lo schema della base di dati.
  Questa metodologia si applica in quei contesti ove non vi è già presente un database definito.

- **Meet in the middle**. Questo metodologia unisce, attraverso una completa mappatura, una base di dati esistente e un diagramma delle classi di dominio dell'applicazione.

- **Bottom up**. Si parte da un base di dati esistente e si ha completa libertà rispetto al dominio dell'applicazione. Si usa una serie di strumenti per generare lo schema di base del codice Java del modello a oggetti per la gestione della persistenza, ovvero il codice delle classi persistenti.

3.3.2 Metodologie di rappresentazione

Hibernate può essere utilizzato principalmente in modalità *Native* o *Provider*. Nel primo caso si utilizzano dei file XML di configurazione per realizzare la mappatura tra le classi da rendere persistenti con le tabelle del database: precisamente ad ogni classe Java si associa il suo file di mappatura XML. Questo rende il codice molto leggibile e facilmente modificabile.
La seconda modalità prevede l'inserimento diretto all'interno delle classi del codice inerente alla mappatura utilizzando lo stile *Java Annotation*[6]; Questo ha il vantaggio di ridurre i file utilizzati ma rende il codice molto meno chiaro, di conseguenza difficilmente modificabile. In questo progetto è stata utilizzata la metodologia *Native*.

[4] `http://www.jboss.org/overview/`
[5] `http://it.wikipedia.org/wiki/GNU_Lesser_General_Public_License`
[6] `http://it.wikipedia.org/wiki/Annotazione_(Java)`

3.3.3 Configurazione

Innanzitutto è necessario scaricare le librerie della versione desiderata di Hibernate e inserire i pacchetti *.jar* nelle directory di lavoro mantenendo la seguente organizzazione:

```
WORKDIR
    +lib
        ***librerie .jar
    +src
        ***classi java + file xml
```

All'interno della cartella *lib* vanno inserite le librerie richieste contenute nella cartella *required* all'interno del pacchetto Hibernate scaricato. Poi si devono aggiungere le librerie seguenti:

- **c3p0.jar**[7]. Sono localizzate all'interno della cartella *option* nel materiale complessivo scaricato. Forniscono i mezzi per gestire in maniera efficace le connessioni, minimizzando i costi di apertura e chiusura delle medesime.
- **hsqldb.jar**. Hsqldb è un DBMS sviluppato in Java.
- **log4j.jar**. E' un tool per la gestione dei log su Java.

Per poter utilizzare Hibernate è necessario definire una *Session Factory* che rappresenta lo strumento che si verrà ad utilizzare per permettere l'interazione tra gli oggetti Java persistenti e il database Nel listato qui mostrato è presente quello utilizzato in questo progetto.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
   Configuration DTD//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>

  <property name="hibernate.connection.url">jdbc:hsqldb:hsql://
     localhost</property>
  <property name="hibernate.connection.username">sa</property>
  <property name="hibernate.dialect">org.hibernate.dialect.
     HSQLDialect</property> -->

  <property name="hibernate.connection.driver_class">org.postgresql
     .Driver</property>
  <property name="hibernate.connection.url">jdbc:postgresql://
     localhost/progetto</property>
```

[7] Il file è scaricabile da `http://sourceforge.net/projects/c3p0/files/latest/download`

```
13  <property name="hibernate.connection.username">postgres</property
       >
    <property name="connection.password"></property>
15  <property name="hibernate.dialect">org.hibernate.dialect.
       PostgreSQLDialect</property>

17  <property name="hibernate.c3p0.min_size">5</property>
    <property name="hibernate.c3p0.max_size">20</property>
19  <property name="hibernate.c3p0.timeout">300</property>
    <property name="hibernate.c3p0.max_statements">50</property>
21  <property name="hibernate.c3p0.idle_test_period">3000</property>

23  <property name="hibernate.show_sql">false</property>
    <property name="hibernate.use_sql_comments">false</property>
25  <property name="hibernate.format_sql">true</property>

27  <property name="current_session_contex_class">thread</property>
    <property name="hibernate.order_updates">true</property>
29
    <mapping resource="DB/Articolo.hbm.xml"/>
31  <mapping resource="DB/Modello.hbm.xml"/>
    <mapping resource="DB/Centro_di_lavoro.hbm.xml"/>
33  <mapping resource="DB/Lavorazione.hbm.xml"/>
    <mapping resource="DB/Materiale.hbm.xml" />
35  <mapping resource="DB/Adopera.hbm.xml" />
    <mapping resource="DB/Prodotto.hbm.xml" />
37  <mapping resource="DB/Segue.hbm.xml"/>
    <mapping resource="DB/Macchina.hbm.xml"/>
39  <mapping resource="DB/Configurazione.hbm.xml"/>
    <mapping resource="DB/Presettaggio.hbm.xml"/>
41  <mapping resource="DB/Trattamento_galvanico.hbm.xml"/>
    <mapping resource="DB/Verniciatura.hbm.xml"/>
43  <mapping resource="DB/Stampaggio.hbm.xml"/>
    <mapping resource="DB/Parametro.hbm.xml"/>
45  <mapping resource="DB/Utilizza.hbm.xml"/>
    <mapping resource="DB/Componente.hbm.xml"/>
47  <mapping resource="DB/Usa.hbm.xml"/>
    <mapping resource="DB/Possiede.hbm.xml"/>
49  <mapping resource="DB/Descritta.hbm.xml"/>

51 </session-factory>
   </hibernate-configuration>
```

Ecco i dettagli relativi ai campi principali:

- Le righe dalla 11 alla 15 forniscono le informazioni inerenti al DBMS uti-

lizzato; in caso di cambiamento del medesimo queste sono le uniche informazioni da modificare.

- La riga 17 fornisce la quantità minima di connessioni tenute pronte dal connection pool, mentre la riga 18 il massimo.
- La riga 20 fornisce il numero massimo di prepared statements che possono essere in chache. Questo parametro è molto importante in termini prestazionali.
- Le righe dalla 42 alla 62 forniscono un'indicazione di dove si trovano i file di mappatura caratteristici del progetto.

Interfacce Hibernate fornisce tre principali interfacce per l'accesso al database, tutte appartenenti al package org.hibernate:

- *Session*: ogni istanza rappresenta una sessione di comunicazione tra applicazione e base di dati, inoltre comprende i metodi per salvataggio/caricamento degli oggetti.
- *Transaction*: ogni istanza rappresenta una transazione. In casi più complicati, gioca il ruolo di gestore di transazioni in un sistema che accede a più database all'interno di un'unica unità di lavoro di sistema.
- *Query*: interfaccia che permette di creare ed eseguire delle interrogazioni (sia nel linguaggio di query di Hibernate (HQL) che in SQL) che usino al loro interno delle variabili speciali.

3.4 Mapping

Come precedentemente scritto si è scelto di realizzare il collegamento che Hibernate permette tra gli oggetti Java persistenti e il database relazionale attraverso dei file di mappatura scritti in XML, uno per ogni classe che si vuole rendere persistente. In questi file sono presenti le informazioni sugli attributi e sui vincoli che definiscono il modello in esame in modo tale che la traduzione sia corretta.

Convenzione JavaBean Un singolo JavaBean essenzialmente è una classe Java che presenta una serie di campi, rappresentanti le proprietà dell'oggetto ed i relativi metodi *get* e *set*. Lo scopo del loro utilizzo è quello di incapsulare al loro interno i dati da rendere persistenti nelle tabelle della base di dati. Di fatto sono gli interpreti del *mapping object-relational*, cioè la trasposizione di dati da un database relazionale ad un oggetto Java. Al fine di funzionare come un JavaBean, una classe di un oggetto deve sottostare a certe convenzioni in merito ai nomi, alla costruzione e al comportamento dei metodi. Queste convenzioni rendono possibile l'utilizzo di *tool* che possono usare, riusare, sostituire e connettere i JavaBean. Le convenzioni richieste sono:

- La classe deve implementare un costruttore senza argomenti
- Le sue proprietà devono essere accessibili usando get, set, is (usato per i booleani al posto di get) e altri metodi (detti metodi accessori) seguendo una convenzione standard per i nomi
- La classe dovrebbe essere *Serializable* (capace di salvare e ripristinare il suo stato in modo persistente)
- Non dovrebbe contenere alcun metodo richiesto per la gestione degli eventi

Di norma quindi si rappresentano le classi definendo le rispettive variabili private, ma rendendo pubblici i metodi di get/set/is e i metodi accessori. Hibernate richiede che gli oggetti da rendere persistenti siano scritti secondo la convenzione JavaBean.

3.4.1 Entità

La mappatura di una entità rappresenta il caso di analisi più semplice ma da già un'idea sostanziale della logica da seguire. Il codice riportato è completo proprio per facilitarne la comprensione, mentre nei casi analizzati successivamente saranno presenti solamente le parti chiave, ovvero le istruzioni che effettivamente caratterizzano la mappatura.

Caso di analisi: l'entità *Macchina* L'entità è stata descritta nel capitolo 3 quindi si riporta a tale punto la descrizione (si veda 2.1). In figura la rappresentazione completa degli attributi.

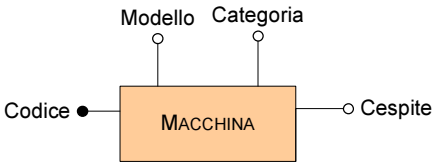

Figura 3.1: L'entità macchina

Le operazioni da svolgere sono state innanzitutto la definizione delle variabili d'ambiente che rappresentassero l'analogia con gli attributi dell'entità e, nel caso in esame (ma vale per tutti gli altri con la stessa forma) si è scelto di convertire un *VARCHAR(255)* con un oggetto di tipo *String*. Poi per tutte le variabiili di persistenza si sono implementati i metodi *get* e *set* come da convenzione JavaBean ed infine i due metodi necessari per permettere che il confronto tra i record di una tabella potesse essere rappresentato sul piano degli oggetti, questo è stato realizzato mediante i metodi *hashCode()* e *equals(Object obj)* (si veda 5.2).

Il file di mappatura (5.3) è caratterizzato dalle dualità classe-tabella e attributo-colonna, in particolare si è definito:

- il nome della tabella associata;
- la chiave primaria tra il target *<id> </id>*;
- le caratteristiche degli attributi tra il target *<property> </property>*.

Il file SQL generato automaticamente (5.4) è risultato essere perfettamente compatibile con quello definito nella progettazione fisica. In appendice il codice completo relativo alla mappatura esaminata.

3.4.2 Associazione

La mappatura di una associazione riprende il lavoro che è stato fatto per il caso dell'entità con l'aggiunta delle specifiche per garantire l'integrità referenziale.

Realizzato

Figura 3.2: L'associazione binaria Realizzato

L'entità Realizzato è del tipo Uno a Molti. Per una corretta mappatura si deve inserire all'interno della classe Modello un riferimento all'oggetto Articolo

```
...
private Articolo articolo;
...
```

mentre all'interno della classe Articolo vi si dovrà aggiungere una struttura dati che racchiuda tutti i modelli ad esso collegati, tipicamente si usa il contenitore di dati *set*:

```
...
private Set<Modello> modelli = new HashSet<Modello>();
...
```

Per quanto concerne i file di mappatura all'interno del file inerente al Modello dovrà essere specificato il vincolo referenziale

```
...
<many-to-one name="articolo" column="ARTICOLO"/>
...
```

mentre nella mappatura di Articolo si dovrà specificare il contenuto del struttura dati definita nel file java.

```
...
<set
    name = "modelli"
    inverse = "true">
    <key column = "codice" />
    <one-to-many class = "DB.Articolo" />
</set>
...
```

Realizza

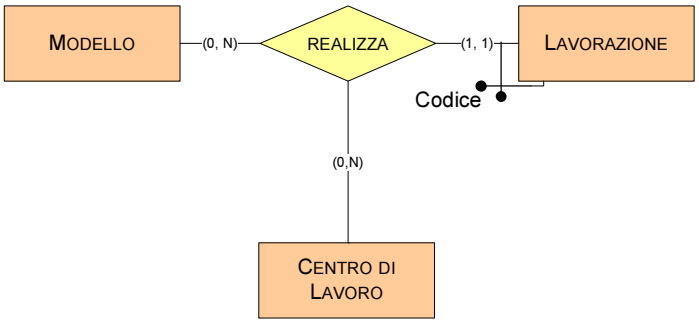

Figura 3.3: L'associazione ternaria Realizza

Questa associazione presenta un caso particolare in quanto rappresenta una entità debole rispetto a due altre entità in maniera congiunta con in più un attributo chiave proprio. Dall'analisi e dalle prove effettuate si evince che per mappare correttamente questa associazione, all'interno della classe Lavorazione si deve definire una sottoclasse statica che conterrà solamente gli attributi chiave (si veda in appendice 5.5), mentre per quanto riguarda le due entità padre basterà specificare un contenitore similmente per quanto fatto nell'entità Articolo nell'esempio precedente.

```
...
private Set<Lavorazione> lavorazioni = new HashSet<Lavorazione>();
...
```

Per mappare questa situazione si deve specificare all'interno del target <composite-id> la sotto-mappatura inerente alla classe interna.

```
<composite-id name="chiave" class="DB.Lavorazione$Dipendenza">
      <key-property name="modello" column="MODELLO" />
      <key-property name="centro" column="CENTRO_DI_LAVORO" />
      <key-property name="codice" column="CODICE" />
</composite-id>
```

```
<!-- definizione attributi -->

<many-to-one name="modello"
             column="MODELLO"
             not-null="true"
             insert="false"
             update="false" />
<many-to-one name="centro"
             column="CENTRO_DI_LAVORO"
             not-null="true"
             insert="false"
             update="false"/>
```

Associazione Utilizza

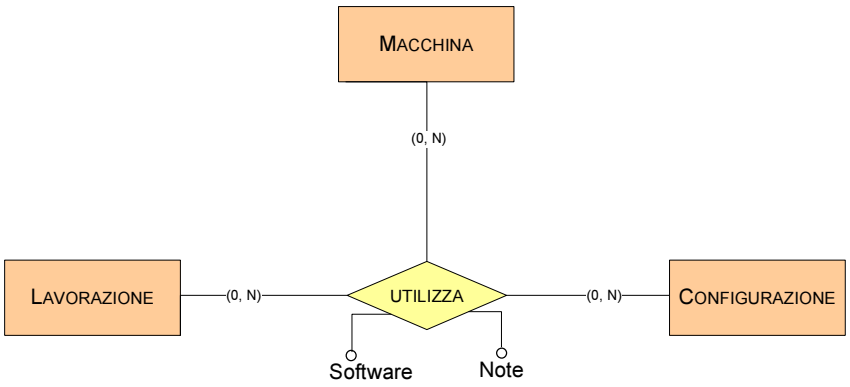

Figura 3.4: L'associazione ternaria Utilizza

Come ultimo esempio si è deciso di riportare un'associazione ternaria Molti-a-Molti che sia affine all'entità Lavorazione, di cui si è trattato sopra. In linea di massima la struttura è la medesima di quanto visto finora (si utilizza anche in questo caso una classe interna per specificare gli attributi identificatori), l'unica particolarità risiede nell'avere un attributo della chiave composto e per questo va specificato utilizzando il target <key-many-to-one> come da figura

```
...
<composite-id name="codice" class="DB.Utilizza$Id">
       <key-many-to-one name="chiave_lavorazione" class="DB.
          Lavorazione">
       <column name="MODELLO"/>
       <column name="FASE"/>
       <column name="LAVORAZIONE"/>
```

```
</key-many-to-one>

    <key-property name="IdConfigurazione" column="CONFIGURAZIONE"
        />
    <key-property name="IdMacchina" column="MACCHINA" />

</composite-id>
...
```

3.5 Data Access Object

L'utilizzo diretto degli oggetti implica la conoscenza delle relative modalità di accesso che essi forniscono. Questo, in applicazioni distribuite causa una dipendenza tra la logica di business[8] e la logica di accesso ai dispositivi di persistenza, quali in questo caso database relazionali. L'obiettivo del pattern *DAO* (Data Access Object) è quello di eliminare questa dipendenza. Ciò si ottiene implementando delle classi apposite che concentrino il codice per l'accesso al sistema di persistenza, mascherando così al programmatore che andrà ad interfacciarsi con gli oggetti persistenti tutta la parte sottostante relativa al database (utilizzando cioè il paradigma dell'incapsulamento).
Nel caso dell'applicazione in esame tutte le operazioni di persistenza di base sono state raggruppate nella super interfaccia generica *GenericDAO* (si veda 5.6). Una classe poi la implementa adottando la soluzione di persistenza caratteristica di Hibernate (5.7). Si deve quindi creare una interfaccia DAO (e in seguito implementarla) per ogni classe che rappresenti una entità del dominio di applicazione. Ognuna di queste sarà definita dai metodi di interrogazione e manipolazione della corrispondente classe di dominio. In particolare conterrà le funzionalità di base CRUD.
In figura l'interfaccia relativa all'oggetto Articolo:

```
package DAO;
import java.util.Set;
import DB.*;

public interface ArticoloDAO extends GenericDAO<Articolo, String>
{

    Set<Modello> readModelli (Articolo articolo);
}
```

e la sua implementazione:

[8] http://it.wikipedia.org/wiki/Business_logic

```
1 public class ArticoloDAOHibernate extends GenericHibernateDAO <
    Articolo, String> implements ArticoloDAO
  {
3
      @Override
5     public Set<Modello> readModelli(Articolo articolo){
          return articolo.getModelli();
7     }//readLavorazioni

9 }//class
```

Capitolo 4

Conclusioni

Questo elaborato nasce con l'intento di progettare uno strumento che permetta di gestire in maniera efficace il processo produttivo di un'azienda. Le motivazioni che hanno portato a questa scelta sono da ricercasi nell'ineludibile compito che hanno tali aziende di informatizzare il bagaglio di informazioni che la funzione sistemi informativi si occupa di fornire, in particolare per quanto riguarda l'industrializzazione. Spesso tutto questo non viene messo in atto a causa degli elevati costi delle soluzioni presenti sul mercato e perché frequentemente queste sono solo degli *optional* aggiunti a forniture fisiche di prodotti, quali ad esempio macchinari o utensili. Si è voluto quindi implementare un sistema fatto su misura per la catalogazione del processo produttivo, atto a mostrare tutto ciò che si necessita per realizzare un articolo commerciale.
Per adempiere a tale scopo si è innanzitutto proceduto a raccogliere le informazioni necessarie che caratterizzano oggigiorno i meccanismi che portano alla produzione di un bene, focalizzando l'attenzione sul concetto di lavorazione e su come essa possa essere rappresentata e gestita. L'analisi si è via via affinata cercando di mantenere contemporaneamente un elevato grado di precisione e flessibilità: difatti il progetto non è stato studiato per un'unica realtà industriale ma guarda ad uno spettro ampio di tipologie aziendali. E' possibile considerare sia delle lavorazioni appartenenti alle quattro delle principali categorie di azione rappresentate (asportazione, trattamento galvanico, verniciatura, stampaggio) che altre non direttamente catalogabili con queste, dando comunque la possibilità di visualizzarne le informazioni tecniche necessarie per il corretto svolgimento. Un aspetto chiave è l'impossibilità di definire una lavorazione come un elemento a se stante ma va sempre considerata all'interno del contesto del modello in esame e del centro di lavoro specifico. Esaminate tutte le informazioni si è proceduto con la realizzazione del modello concettuale, ovvero con la stesura di uno schema sì formale ma ad alto livello di astrazione, che permetta comunque di dare una prima impostazione al sistema da realizzare. Successivamente si è provveduto alla progettazione logica, dove si ottiene una rappresentazione fedele ed efficiente delle informazioni contenute nel modello concettuale, e infine con la progettazione fisica che porta ad avere l'implementazione dello schema logico in un linguaggio SQL.

Per la parte inerente all'applicazione si è scelto di utilizzare il linguaggio Java, ma diversamente da quanto comunemente si usa fare, non è stato utilizzando il sistema di collegamento applicazione - database tramite JDBC ma si è preferito utilizzare la tecnologia ORM (Object Relation Mapping) che permette di realizzare la dualità oggetto - tabella. Nello specifico si è utilizzata la più nota per Java ovvero Hibernate.
Questa scelta non è stata casuale ma frutto dei vantaggi che ne derivano. Difatti Hibernate permette di ridurre drasticamente il numero delle righe di codice necessarie per colloquiare con il database grazie agli automatismi che introduce per le operazioni tipiche CRUD (Create Read Update Delete), presenta una gestione efficiente delle connessioni, rende il codice perfettamente scalabile e adattabile alle modifiche e/o revisioni che spesso possono capitare nel ciclo di vita di una applicazione, porta una elevata portabilità rispetto alla tecnologia DMBS utilizzata.
Il lavoro svolto è stato quello di mappare gli oggetti persistenti nelle relative tabelle mantenendo la struttura ricavata dalla parte di progettazione. Questo ha richiesto una parte notevole del tempo, in quanto si è dovuto provare diverse tipologie di mappature fino a trovare quelle che più si adattavano alla situazione in esame. L'ultimo passaggio è stato fornire degli appositi metodi per la navigazione tra gli oggetti che sia indipendente dalla logica di accesso al dispositivo di persistenza e per far ciò si è utilizzato il patter DAO (Data Access Object).
L'applicazione fin qui descritta presenta numerose possibilità di sviluppo. Innanzitutto si può implementare la parte grafica, in modo tale da renderla facilmente utilizzabile. Poi si potrà interfacciare con altre applicazioni simili che però prendano in considerazione gli aspetti che qui sono stati volutamente trattati marginalmente quali la gestione del reperimento dei materiali, le forniture di ordini inerenti a gruppi di modelli, la catalogazione degli componenti macchina (ad esempio gli utensili) con tutti i dettagli che li caratterizzano, e molti altri.

Capitolo 5

Appendice

5.1 Codice SQL

```
CREATE TABLE ARTICOLO (
    codice VARCHAR (255),
    descrizione TEXT,
    PRIMARY KEY (codice)
);

CREATE TABLE MODELLO(
    codice VARCHAR(255),
    idarticolo VARCHAR (255) NOT NULL,
    nome VARCHAR(255) NOT NULL UNIQUE,
    status VARCHAR(255),
    descrizione TEXT,
    immagine_2D VARCHAR(255),
    immagine_3D VARCHAR(255),
    PRIMARY KEY (codice),
    FOREIGN KEY(idarticolo) REFERENCES Articolo(codice) ON DELETE
       CASCADE ON UPDATE CASCADE
);

CREATE TABLE CENTRO_DI_LAVORO(
    codice VARCHAR(255),
    livello INT,
    manodopera_impiegata INT,
    PRIMARY KEY (codice)
);

CREATE TABLE LAVORAZIONE (
    codice VARCHAR(255),
    idmodello VARCHAR(255),
    idcentro VARCHAR(255),
```

```
    tempo TIME,
    note TEXT,
    immagine_2D VARCHAR(255),
    immagine_3D VARCHAR(255),
    idlavorazione VARCHAR(255),
    PRIMARY KEY (codice, idmodello, idcentro),
    FOREIGN KEY(idmodello) REFERENCES Modello(codice) ON DELETE
       CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(idcentro) REFERENCES Centro_di_Lavoro(codice) ON
       DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE MATERIALE(
    codice VARCHAR(255),
    categoria VARCHAR(255),
    PRIMARY KEY(codice)
);

CREATE TABLE ADOPERA(
    idlavorazione VARCHAR(255),
    idmodello VARCHAR(255),
    idcentro VARCHAR(255),
    idmateriale VARCHAR(255),
    quantita INT NOT NULL,
    note TEXT,
    PRIMARY KEY(idlavorazione, idmodello, idcentro, idmateriale),
    FOREIGN KEY(idlavorazione,idmodello,idcentro) REFERENCES
       Lavorazione(codice,idmodello,idcentro) ON DELETE CASCADE ON
        UPDATE CASCADE,
    FOREIGN KEY (idmateriale) REFERENCES Materiale(codice) ON
       DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE PRODOTTO(
    idmateriale VARCHAR(255),
    idlavorazione VARCHAR(255),
    idmodello VARCHAR(255),
    idcentro VARCHAR(255),
    note TEXT,
    PRIMARY KEY(idlavorazione, idmodello, idcentro, idmateriale),
    FOREIGN KEY(idlavorazione,idmodello,idcentro) REFERENCES
       Lavorazione(codice,idmodello,idcentro) ON DELETE CASCADE ON
        UPDATE CASCADE,
    FOREIGN KEY (idmateriale) REFERENCES Materiale(codice) ON
       DELETE CASCADE ON UPDATE CASCADE
```

```
67 );

69 CREATE TABLE SEGUE(
       idlavorazione1 VARCHAR(255),
71     idmodello1 VARCHAR(255),
       idcentro1 VARCHAR(255),
73     idlavorazione2 VARCHAR(255),
       idmodello2 VARCHAR(255),
75     idcentro2 VARCHAR(255),
       note TEXT,
77     PRIMARY KEY(idlavorazione1, idmodello1, idcentro1,
           idlavorazione2, idmodello2, idcentro2),
       FOREIGN KEY(idlavorazione1,idmodello1,idcentro1) REFERENCES
           Lavorazione(codice,idmodello,idcentro) ON DELETE CASCADE ON
            UPDATE CASCADE,
79     FOREIGN KEY(idlavorazione2,idmodello2,idcentro2) REFERENCES
           Lavorazione(codice,idmodello,idcentro) ON DELETE CASCADE ON
            UPDATE CASCADE
   );
81

   CREATE TABLE MACCHINA(
83     codice VARCHAR(255),
       modello VARCHAR(255) NOT NULL,
85     categoria VARCHAR(255) NOT NULL,
       cespite VARCHAR (255) UNIQUE,
87     PRIMARY KEY(codice)
   );
89

   CREATE TABLE CONFIGURAZIONE(
91     codice VARCHAR(255),
       categoria VARCHAR(255),
93     note TEXT,
       PRIMARY KEY (codice)
95 );

97 CREATE TABLE PRESETTAGGIO(
       idconfigurazione VARCHAR(255),
99     numero_taglienti INT,
       lunghezza_tagliente FLOAT,
101    diametro_tagliente FLOAT,
       configurazione_DxSx BOOLEAN,
103    materiale VARCHAR(255),
       immagine_2d VARCHAR(255),
105    PRIMARY KEY(idconfigurazione),
       FOREIGN KEY(idconfigurazione) REFERENCES Configurazione(codice)
```

```
107 );

109 CREATE TABLE TRATTAMENTO_GALVANICO(
        idconfigurazione VARCHAR(255),
111     tensione FLOAT,
        intesita_corrente FLOAT,
113     tempo TIME,
        superficie FLOAT,
115     temperatura FLOAT,
        PRIMARY KEY(idconfigurazione),
117     FOREIGN KEY(idconfigurazione) REFERENCES Configurazione(codice)
    );
119
    CREATE TABLE STAMPAGGIO(
121     idconfigurazione VARCHAR(255),
        tempo_iniezione TIME,
123     temperatura_iniezione FLOAT,
        pressione FLOAT,
125     PRIMARY KEY(idconfigurazione),
        FOREIGN KEY(idconfigurazione) REFERENCES Configurazione(codice)
127 );

129 CREATE TABLE VERNICIATURA(
        idconfigurazione VARCHAR(255),
131     tempo_avanzamento TIME,
        tempo_essicazione TIME,
133     tempo_passivazione TIME,
        PRIMARY KEY(idconfigurazione),
135     FOREIGN KEY(idconfigurazione) REFERENCES Configurazione(codice)
    );
137
    CREATE TABLE PARAMETRO(
139     codice VARCHAR(255),
        unita_misura VARCHAR(255),
141     descrizione TEXT,
        PRIMARY KEY (codice)
143 );

145 CREATE TABLE UTILIZZA(
        idlavorazione VARCHAR (255),
147     idmodello VARCHAR (255),
        idcentro VARCHAR(255),
149     idmacchina VARCHAR (255),
        idconfigurazione VARCHAR (255),
151     software VARCHAR(255),
```

```
    note TEXT,
153 PRIMARY KEY (idlavorazione, idmodello, idcentro, idmacchina,
       idconfigurazione),
    FOREIGN KEY(idlavorazione,idmodello,idcentro) REFERENCES
       Lavorazione(codice,idmodello,idcentro) ON DELETE CASCADE ON
        UPDATE CASCADE,
155 FOREIGN KEY(idmacchina) REFERENCES Macchina(codice) ON DELETE
       CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(idconfigurazione) REFERENCES Configurazione(codice)
        ON DELETE CASCADE ON UPDATE CASCADE
157 );

159 CREATE TABLE COMPONENTE(
    codice VARCHAR(255),
161 categoria VARCHAR(255) NOT NULL,
    descrizione TEXT,
163 cespite VARCHAR(255) UNIQUE,
    PRIMARY KEY (codice)
165 );

167 CREATE TABLE USA(
    idpresettaggio VARCHAR (100),
169 idcomponente VARCHAR (100),
    descrizione TEXT,
171 numero INT NOT NULL DEFAULT '1',
    PRIMARY KEY (idpresettaggio, idcomponente),
173 FOREIGN KEY(idpresettaggio) REFERENCES Presettaggio(
       idconfigurazione) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(idcomponente) REFERENCES Componente(codice) ON
       DELETE CASCADE ON UPDATE CASCADE
175 );

177 CREATE TABLE POSSIEDE(
    idconfigurazione VARCHAR (255),
179 idparametro VARCHAR (255),
    valore VARCHAR(255),
181 tolleranze TEXT,
    PRIMARY KEY (idconfigurazione, idparametro),
183 FOREIGN KEY(idconfigurazione) REFERENCES Configurazione(codice)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(idparametro) REFERENCES Parametro(codice) ON DELETE
        CASCADE ON UPDATE CASCADE
185 );

187 CREATE TABLE DESCRITTA(
```

```
    idlavorazione VARCHAR (255),
    idmodello VARCHAR (255),
    idcentro VARCHAR (255),
    idparametro VARCHAR (255),
    valore VARCHAR(255),
    PRIMARY KEY (idlavorazione, idmodello, idcentro, idparametro),
    FOREIGN KEY(idlavorazione,idmodello,idcentro) REFERENCES
       Lavorazione(codice,idmodello,idcentro) ON DELETE CASCADE ON
        UPDATE CASCADE,
    FOREIGN KEY(idparametro) REFERENCES Parametro(codice) ON DELETE
        CASCADE ON UPDATE CASCADE
);
```

5.2 Macchina.java

```
package DB;

public class Macchina {

    private String codice;
    private String modello;
    private String categoria;
    private String cespite;

    public Macchina(){} //costruttore vuoto richiesto

    public Macchina(String codice, String modello, String categoria
        , String cespite){
        this.codice = codice;
        this.modello = modello;
        this.categoria = categoria;
        this.cespite = cespite;
    }//costruttore

    public String getCodice() {
        return codice;
    }//getCodice

    public void setCodice(String codice) {
        this.codice = codice;
    }//setCodice

    public String getModello() {
        return modello;
```

```
    }//getModello

    public void setModello(String modello) {
        this.modello = modello;
    }//setModello

    public String getCategoria() {
        return categoria;
    }//getCategoria

    public void setCategoria(String categoria) {
        this.categoria = categoria;
    }//setCategoria

    public String getCespite() {
        return cespite;
    }//getCespite

    public void setCespite(String cespite) {
        this.cespite = cespite;
    }//setCespite

    public int hashCode() {
        return getCodice().hashCode();
    }//hashCode

    public boolean equals(Object obj) {
        if (this == obj)return true;
        if (obj == null)return false;
        if (!(obj instanceof Macchina)) return false;
        final Macchina that = (Macchina) obj;
        return this.getCodice().equals(that.getCodice());
    }//equals

}//class
```

5.3 Macchina.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
```

```
7
    <class
9       name="DB.Macchina"
        table="MACCHINA">
11
        <id
13          name="codice"
            column="CODICE">
15      </id>

17      <property
            name="modello"
19          column="MODELLO"/>

21      <property
            name="categoria"
23          column="CATEGORIA"/>

25      <property
            name="cespite"
27          column="CESPITE"/>

29  </class>

31 </hibernate-mapping>
```

5.4 Codice SQL autogenerato di Macchina

```
1  CREATE TABLE macchina
   (
3    codice character varying(255) NOT NULL,
     modello character varying(255),
5    categoria character varying(255),
     cespite character varying(255),
7    CONSTRAINT macchina_pkey PRIMARY KEY (codice )
   )
9  WITH (
     OIDS=FALSE
11 );
   ALTER TABLE macchina
13   OWNER TO postgres;
```

5.5 Sottoclasse di Lavorazione

```
1  public static class Dipendenza implements Serializable{
2
3      public String modello;
4      public String centro;
5      public String codice;
6
7      public Dipendenza(){}
8
9      public String getCodice() {
10         return codice;
11     }//getCodice
12
13     public void setCodice(String codice) {
14         this.codice = codice;
15     }//setCodice
16
17     public int hashCode(){
18         return modello.hashCode() + centro.hashCode() + codice.
               hashCode();
19     }//hasCode
20
21     public boolean equals (Object obj){
22         if (this == obj) return true;
23         if (this ==null) return false;
24         if (!(obj instanceof Dipendenza)) return false;
25         final Dipendenza that = (Dipendenza) obj;
26
27         return this.modello.equals(that.modello) && this.centro.
               equals(that.centro) && this.codice.equals(that.codice);
28     }//equals
29 }
```

5.6 GenericDAO

```
1  package DAO;
2
3  import persistence.DbConnection;
4
5      @SuppressWarnings("rawtypes")
6      public static final Class HIBERNATE = DAO.hibernate.
           HibernateDAOFactory.class;
```

```
7
    // Metodo per istanziare un DAOFactory concreto.
9   @SuppressWarnings("rawtypes")
    public static DAOFactory instance(Class factory) {
11      try {
            return (DAOFactory)factory.newInstance();}
13      catch (Exception ex) {
            throw new RuntimeException("Impossibile creare il
               DAOFactory: " + factory);
15      }
    }//DAOFactory
17
    /*Oggetto DbConnection che gestisce la connessione al DB e il
19  * contesto di persistenza*/
    public static DbConnection dbConnection;
21
    //Metodo che restituisce l'oggetto DbConnection
23  public abstract DbConnection getDbConnection();

25  // Elenco di tutti metodi astratti che restituiscono una
       istanza del rispettivo DAO
    public abstract ArticoloDAO getArticoloDAO();
27  public abstract DescrittaDAO getDescrittaDAO();
    public abstract Centro_di_lavoroDAO getCentro_di_lavoroDAO();
29  public abstract LavorazioneDAO getLavorazioneDAO();
    public abstract MacchinaDAO getMacchinaDAO();
31  public abstract ModelloDAO getModelloDAO();
    public abstract ParametroDAO getParametroDAO();
33  public abstract PossiedeDAO getPossiedeDAO();
    public abstract PresentaDAO getPresentaDAO();
35  public abstract PresettaggioDAO getPresettaggioDAO();
    public abstract SegueDAO getSegueDAO();
37  public abstract UsaDAO getUsaDAO();
    public abstract UtensileDAO getUtensileDAO();
39  public abstract UtilizzaDAO getUtilizzaDAO();
    public abstract MaterialeDAO getMaterialeDAO();
41 }//class
```

5.7 GenericDAOHibernate

```
1  package DAO.hibernate;

3  import persistence.HibernateUtil;
   import org.hibernate.*;
```

```
5  import org.hibernate.criterion.*;
7  import DAO.GenericDAO;
9  import java.util.*;
   import java.io.Serializable;
11 import java.lang.reflect.ParameterizedType;
13 public abstract class GenericHibernateDAO<T, ID extends
       Serializable>
           implements GenericDAO<T, ID> {
15
       private Class<T> persistentClass;
17     private Session session;
19     @SuppressWarnings("unchecked")
       public GenericHibernateDAO() {
21         this.persistentClass = (Class<T>) ((ParameterizedType)
               getClass()
                                   .getGenericSuperclass()).
                                      getActualTypeArguments()[0];
23      }
25     public void setSession(Session s) {
           this.session = s;
27     }
29     protected Session openSession(){
           if (session == null)
31             session = HibernateUtil.getSessionFactory().openSession
                   ();
           return session;
33         }
35     protected Session getSession() {
           if (session == null)
37             session = HibernateUtil.getSessionFactory().
                   getCurrentSession();
           return session;
39     }
41     public Class<T> getPersistentClass() {
           return persistentClass;
43     }
```

```
45      @SuppressWarnings("unchecked")
        public T findById(ID id, boolean lock) {
47          T entity = (T) getSession().load(getPersistentClass(), id);
            return entity;
49      }

51      public List<T> findAll() {
            return findByCriteria();
53      }

55      @SuppressWarnings("unchecked")
        public List<T> findByExample(T exampleInstance, String...
            excludeProperty) {
57          Criteria crit = getSession().createCriteria(
                getPersistentClass());
            Example example = Example.create(exampleInstance);
59          for (String exclude : excludeProperty) {
                example.excludeProperty(exclude);
61          }
            crit.add(example);
63          return crit.list();
        }
65
        public T makePersistent(T entity) {
67          getSession().saveOrUpdate(entity);
            return entity;
69      }

71      public void makeTransient(T entity) {
            getSession().delete(entity);
73      }

75      public void flush() {
            getSession().flush();
77      }

79      public void clear() {
            getSession().clear();
81      }

83      @SuppressWarnings("unchecked")
        protected List<T> findByCriteria(Criterion... criterion) {
85          Criteria crit = getSession().createCriteria(
                getPersistentClass());
            for (Criterion c : criterion) {
```

```
87              crit.add(c);
            }
89          return crit.list();
        }
91  }
```

Elenco delle figure

Bibliografia

[1] **C. Bauer, G. King**, *Java persistence with Hibernate*. Manning, 2007.

[2] **P.Atzeni, S. Ceri, S. Paraboschi, R. Torlone,** *Basi di dati, modelli e linguaggi di interrogazione*. McGraw-Hill, 3â edizione, 2009.

[3] **A. Elmastri, B. Navathe,** *Sistemi di basi di dati, fondamenti*. Pearson, 5â edizione, 2007.

[4] **M. Carraro,** *Studio di Hibernate attraverso i costrutti del modello relazionale*. Tesi di Laurea, Università degli studi di Padova, 2012.

[5] **M. Rampazzo,** *Progettazione e realizzazione di un sistema di gestione di esperimenti scientifici basato su architetture Spring-Hibernate*. Tesi di Laurea, Università degli studi di Padova, 2012.

[6] **A. Baudoin,** *Impara* "LaTeX!", Documentazione on-line di LaTeX, 1998.

[7] **Hibernate** `http://www.hibernate.org`

[8] **Postgresql** `http://www.postgresql.org`

[9] **Eclipse** `http://www.eclipse.org`