FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Università degli Studi di Salerno

CORSO DI INGEGNERIA DEL SOFTWARE
PROF. GIANCARLO NOTA
A.A. 2006/2007

PROGETTO: *G.P.C. (Gestione Polo Conciario)*

ESEGUITO DA:

Davino Cristiano: 0510200572
Caputo Luca Floriano: 0510200818
Ferro Amedeo: 0510200578
Mercogliano Umberto: 0510200638
Ferri Vincenzo: 0510200659
Fierro Annunziato: 0510200815

DOCUMENTO: **System Design Document(SDD)**

PROGETTO : GPC

DOCUMENTO : DOCUMENTO DI SYSTEM DESIGN DOCUMENT

SCRITTO DA : Davino Cristiano
DATA: 22/05/2007
Approvato dal team.

STRUMENTI C.A.S.E. DI SUPPORTO

STRUMENTO C.A.S.E.	PRODUTTORE	UTILIZZATO PER
WORD 2003	MICROSOFT	STESURA DOCUMENTO
WINPROJECT PRO 2003	MICROSOFT	SCHEDULING PROGETTO
POISEDON FOR UML 5.0	GENTLEWARE	GRAFICI UML
IBM RATIONAL MODELER	IBM	GRAFICI UML

1. Scopo del Sistema

Il Sistema che si vuole realizzare integra,sostanzialmente,vari aspetti di un polo conciario.
Possiamo individuare degli aspetti cardine all'interno del nostro sistema sintetizzati in questi punti.:

-) Gestione delle informazioni inerenti ai Fornitori
-) Gestione delle informazioni inerenti ai nostri Clienti
-) Informazioni inerenti le merci contenute all'interno del magazzino
-) Gestione dei documenti relativi alla compravendita dei prodotti

Ai fini di perseguire l'obiettivo proposto il Sistema dovrà

- Presentare un interfaccia che guidi il Gestore attraverso le varie funzionalità della sua modalità che permettano una buona gestione dei dati del Cliente,dei Fornitori,dei Documenti.
- Presentare un interfaccia che guidi il Magazziniere e che gli permetta di gestire al meglio le informazioni relative alle merci contenuti all'interno del Magazzino(con le relative quantità stoccate);degli ordini di vendita e dell'approvvigionamento.
- Fornire dei messaggi di errore chiari e precisi che informino gli utenti interessati.

Il software deve saper interagire con una Base di Dati contenete tutte le informazioni necessarie per il corretto svolgimento delle attività della sede.
Per ulteriori informazioni sulle funzionalità che il sistema deve offrire si rimanda al documento di analisi dei requisiti (RAD).

2. Obiettivi di design

Gli scopi del System Design sono quelli di definire gli obiettivi di progettazione del sistema, decomporre il sistema in sottosistemi più piccoli in modo da poterli assegnare a team individuali e selezionare alcune strategie quali:

- Scelte hardware e software
- Gestione dei dati persistenti
- Il flusso di controllo globale
- Le politiche di controllo degli accessi
- La gestione delle condizioni boundary (startup, shutdown, eccezioni)

Il System Design si focalizza sul dominio di implementazione, prende in input il modello di analisi e dopo averlo trasformato da in output un modello del sistema.
Principale obiettivo di tale fase è la definizione dell'architettura software del sistema ad un livello di dettaglio tale da consentire la successiva implementazione delle componenti strutturali del sistema software.

3. Prestazioni

1. **Tempo di risposta:**
Qualsiasi operazione deve essere soddisfatta entro un tempo di attesa accettabile, nell'ordine dei secondi.

2. **Memoria:**
I programmi sui terminali funzionano su una Virtual Machine Java, quindi tali terminali devono avere memoria sufficiente a garantire il corretto funzionamento di essa. Gli oggetti memorizzati nel sistema sono molto numerosi e quindi è necessaria una cospicua quantità di memoria ai fini di rispettare i requisiti del Sistema.

4. Affidabilità

1.Robustezza:
Gli input devono essere quando possibile guidati, per minimizzare le possibilità di inserimenti errati. Comunque, tutti i dati immessi devono essere controllati (login e password)per gli utenti aventi accesso al Sistema.

2. **Attendibilità** e **Tolleranza ai faul**t:
In caso di errore vengono mostrati messaggi informativi all'utente e l'operazione che si stava compiendo viene interrotta.

5. Costo

Non sono previsti costi di sviluppo o di manutenzione in termini di denaro perche utilizzeremo come linguaggio di programmazione "Java 6" che attualmente è stato rilasciato su licenza Open Source.
Discorso identico per il DBMS MySql che verrà utilizzato per lo storage dei dati.

6. Manutenzione

Per quanto possibile, il codice del sistema deve avvalersi dei principi della programmazione ad oggetti (astrazione, ereditarietà, incapsulamento), perché queste misure aumentano l'**estensibilit**à, la **modificabilit**à, la **leggibilità** del codice.

Per incrementare la **portabilità** del sistema si è scelto di usare la piattaforma Java, compatibile con numerose architetture hardware/software,un DBMS di tipo MySql compatibile con la maggior parte dei Sistemi Operativi.

7. Utenti finali

Utilità e Usabilità
Il sistema deve gestire tutto il processo di gestione di un Polo Conciario in modo semplice ed intuitivo attraverso delle interfacce User-Friendly che guideranno l'utente durante lo svolgimento della propria attività lavorativa.

Le interfacce grafiche, l'inserimento guidato e il controllo degli errori devono rendere il sistema facile da usare per l'utente.

8. Glossario Dei Termini

Criteri di performance: In questi criteri vengono inclusi: tempo di risposta, throughput (quantità di task eseguibili in un determinato periodo di tempo) e memoria.

Criteri di affidabilità: L'affidabilità include criteri di robustezza (capacità di gestire condizioni non previste), attendibilità (non ci deve essere differenza tra il comportamento atteso e quello osservato), disponibilità (tempo in cui il sistema è disponibile per l'utilizzo), tolleranza ai fault (capacità di operare in condizioni di errore), sicurezza, fidatezza (capacità di non danneggiare vite umane).

Criteri di costi: Vanno valutati i costi di sviluppo del sistema, alla sua installazione e al training degli utenti, eventuali costi per convertire i dati del sistema precedente, costi di manutenzione e costi di amministrazione.

Criteri di mantenimento: Tra i criteri di mantenimento troviamo: estendibilità, modificabilità, adattabilità, portabilità, leggibilità e tracciabilità dei requisiti.

Criteri di utente finale: In criteri dell'utente finale da tenere in considerazione sono quelli di utilità (quando bene il sistema dovrà facilitare e supportare il lavoro dell'utente) e quelli di usabilità.

Accoppiamento (coupling): Misura quanto un sistema è dipendente da un altro. Due sistemi si dicono loosely coupled (leggermente accoppiati) se una modifica in un sottosistema avrà poco impatto nell'altro sistema, mentre si dicono strongly coupled (fortemente accoppiati) se una modifica su uno dei sottosistemi avrà un forte impatto sull'altro. La condizione ideale di accoppiamento è quella di tipo loosely in quanto richiede meno sforzo quando devono essere modificate delle componenti.

Coesione: Misura la dipendenza tra le classi contenute in un sottosistema. La coesione è alta se due componenti di un sottosistema realizzano compiti simili o sono collegate l'una con l'altra attraverso associazioni (es. ereditarietà).

Architetture software: Un architettura software include scelte relative alla decomposizione in sottosistemi, flusso di controllo globale, gestione delle condizione limite e i protocolli di comunicazione tra i sottosistemi. E' da notare che la decomposizione dei sottosistemi è una fase molto critica in quando una volta iniziato lo sviluppo con una determinata decomposizione è complesso ed oneroso dover tornare indietro in quanto molte interfacce dei sottosistemi dovrebbero essere modificate.

File: La prima tipologia da una parte richiede una logica più complessa per la scrittura e lettura, dall'altra permette un accesso ai dati più efficiente.

DBMS relazionale: Un DMBS relazione fornisce un' interfaccia di più alto livello rispetto al file. I dati vengono memorizzati in tabelle ed è possibile utilizzare un linguaggio standard per le operazioni (SQL). Gli oggetti devono essere mappati sulle tabelle per poter essere memorizzati.

DBMS ad oggetti: Un database orientato ad oggetti è simile ad un DBMS relazionale con la differenza che non è necessario mappare gli oggetti in tabelle in quanto questi vengono memorizzati così come sono. Questo tipo di database riduce il tempo di setup iniziale (si risparmia sulle decisioni di mapping) ma sono più lenti e le query sono di più difficile comprensione.

Tabella di accesso globale: Ogni riga della matrice contiene una tripla (attore, classe, operazione). Se la tupla è presente per una determinata classe e operazioni l'accesso è consentito altrimenti no.

Access control list (ACL): Ogni classe ha una lista che contiene una tupla (attore, operazione) che specifica se l'attore può accedere a quella determinata operazione della classe a cui la ACL appartiene.

Capability: Una capability è associata ad un attore ed ogni riga della matrice contiene una tupla (classe, operazione che l'attore a cui è associata può eseguire.

Flusso di controllo: è una sequenza di azioni di un sistema. In un sistema Object Oriented una sequenza di azioni include prendere decisioni su quali operazioni eseguire ed in che ordine. Queste decisioni sono basate su eventi esterni generati da attori o causati dal trascorrere del tempo. Esistono tre tipi di controlli di flusso: (Procedure-driven control , Event-driven control ,Threads).

Procedure-driven control: Le operazioni rimangono in attesa di un input dell'utente ogni volta che hanno bisogno di elaborare dati. Questo tipo di controllo di flusso è particolarmente usato in sistemi legacy di tipo procedurale.

Event-driven control: In questo controllo di flusso un ciclo principale aspetta il verificarsi di un evento esterno. Non appena l'evento diventa disponibile la richiesta viene direzionata all'opportuno oggetto. Questo tipo di controllo ha il vantaggio di centralizzare tutti gli input in un ciclo principale ma ha lo svantaggio di rendere complessa l'implementazione di sequenze di operazioni composte di più passi.

Threads: Questo controllo di flusso è una modifica del procedure-driven control che aggiunge la gestione della concorrenza. Il sistema può creare un arbitrario numero di threads (processi leggeri), ognuno assegnato ad un determinato evento.

Diagramma delle Classi: Notazione UML rappresentante la struttura del sistema in termini di oggetti, classi, attributi, operazioni e associazioni. I diagrammi delle classi sono utilizzati per rappresentare i modelli a oggetti durante lo sviluppo.

Classe: E' un'astrazione di un insieme di oggetti con gli stessi attributi, operazioni, relazione e semantica.

UML: E' l'acronimo di "Unified Modeling Language", è un insieme standard di notazioni grafiche per rappresentare modelli di software; esempi: Diagrammi UML per la rappresentazione delle classi ecc.

Oggetto: E' un'istanza di una classe, rappresenta lo "stato" temporale di una classe, l'oggetto ha una propria identità e memorizza i valori degli attributi.

Dominio di un Applicazione: Rappresenta tutti gli aspetti del problema dell'utente; include l'ambiente fisico, gli utenti e gli altri attori coinvolti con la descrizione del processo lavorativo che essi effettuano.

Attributi: E' una specificazione di una classe che comprende un certo insieme di possibili valori.

Operazione: E' un comportamento elementare di una classe; ad esempio: modifica l'attributo residenza nella classe cliente (operazione: modifica residenza).

Packaging (Requisito): E' un requisito funzionale, rappresenta una particolare restrizione sulla consegna attuale del sistema.

Requisiti Funzionali: Un'area di funzionalità che il sistema deve supportare; descrivono le interazioni tra un attore e il sistema a prescindere dalla realizzazione vera e propria del sistema stesso. Ad essi correlati vi è il Modello funzionale.

Modello Funzionale: Descrive la funzionalità del sistema dal punto di vista dell'utente.

Requisiti non Funzionali: Una costrizione visibile all'utente sul sistema. Descrivono gli aspetti del sistema direttamente visibili dall'utente che non sono associati con la funzionalità del sistema stesso.

Attore: Un'entità che interagisce con il sistema eseguendo specifiche operazioni.

Caso d'uso: Una sequenza generale di interazioni tra uno o più attori e il sistema.

Scenario: Illustra una concreta sequenza di interazioni tra uno o più attori e il sistema; fornisce una descrizione particolareggiata di un determinato Caso d'Uso; descrive l'istante temporale, lo "status" di un particolare Caso d'Uso.

Cliente: Fornisce l'interfaccia grafica tra il sistema e il cliente per iniziare tutti i casi d'uso.

Gestore : Fornisce l'interfaccia grafica tra il sistema e l'amministratore per iniziare tutti i casi d'uso.

Server Amministratore: Gestisce le richieste dei vari componenti del sistema inviando dati al client specifico.

GestioneRicerca: Permette di eseguire le operazioni che riguardano la ricerca all'interno del database dopo aver specificato dei criteri di ricerca.

GestioneOrdine: Permette di eseguire le operazioni che riguardano gli ordini : creazione e monitoraggio della prenotazione.

GestioneAccount: Permette di eseguire le operazioni sugli account utenti presenti all'interno del nostro Sistema.

GestioneFornitori: Permette di eseguire le operazioni che riguardano i fornitori presso i quali l'azienda può rifornirsi: inserimento,monitoraggio,cancellazione,etc..

GestioneClienti: Permette di eseguire le operazioni che riguardano i clienti dell'azienda:inserimento,cancellazione,monitoraggio.

GestioneProdotti: Permette di eseguire le operazioni che riguardano le merci trattate dall'azienda e presenti nel Magazzino: monitoraggio.

GestioneFatture e DDT: Permette di eseguire le operazioni che riguardano i documenti trattati dall'azienda durante la sua attività.

9. Overview del documento

La fase di design è suddivisa secondo i seguenti step:

Decomposizione del Sistema in sottosistemi

Utilizzando come base i casi d'uso e l'analisi e seguendo una particolare architettura software il sistema verrà decomposto in sottosistemi.
Lo scopo è quello di poter assegnare a un singolo sviluppatore o ad un team parti software semplici. In questa fase verrà descritto come i sottosistemi sono collegati alle classi. Un sottosistema è caratterizzato dai servizi (insieme di operazioni) che esso offre agli altri sottosistemi. L'insieme di servizi che un sistema espone viene chiamato interfaccia (API: application programming interface) che include, per ogni operazione: i parametri, il tipo e i valori di ritorno. Le operazioni che essi svolgono verranno descritte ad alto livello senza entrare troppo nello specifico.

Mapping hardware/software

Molti sistemi complessi necessitano di lavorare su più di un computer interconnessi da rete. L'uso di più computer può ottimizzare le performance e permettere l'utilizzo del sistema a più utenti distribuiti sulla rete.
In questa fase verranno prese alcune decisioni per quanto riguarda le piattaforme hardware e software su cui il sistema dovrà girare (es Unix vs Windows, Intel vs Sparc etc).
Una volta decise le piattaforme è necessario mappare le componenti su di esse. Questa operazione potrebbe portare all'introduzione di nuove componenti per interfacciare i sottosistemi su diverse piattaforme (es. una libreria per il collegamento ad un DBMS).
Sfortunatamente, da una parte, l'introduzione di nuovi nodi hardware distribuisce la computazione, dall'altro introduce alcune problematiche tra cui la sincronizzazione, la memorizzazione, il trasferimento e la replicazione di informazioni tra sottosistemi.

Gestione dei dati persistenti

Il modo in cui i dati vengono memorizzati può influenzare l'architettura del sistema e la scelta di uno specifico database . In questa fase vanno identificati gli oggetti persistenti e scelto il tipo di infrastruttura da usare per memorizzarli (dbms, file o altro).
Una volta decisi gli oggetti dobbiamo decidere come questi oggetti devono essere memorizzati. Principalmente potremmo avere a disposizione tre mezzi: file, dbms relazionale e dbms ad oggetti.

Controllo di accesso

In un sistema multi utenza è necessario fornire delle politiche di accesso alle informazioni. Nell'analisi sono stati associati casi d'uso ad attori, in questa fase vanno definite in modo più preciso le operazioni e le informazioni effettuabili da ogni singolo attore e come questi si autenticano al sistema.
Scegliere una soluzione impatta sulle performance del sistema. Ad esempio scegliere una tabella di accesso globale potrebbe far consumare molta memoria. Le altre vanno usate in base al tipo di controllo che vogliamo effettuare: se vogliamo rispondere più velocemente alla domanda "chi può accedere a questa classe?" useremo una ACL (Access control list), se invece vogliamo rispondere più velocemente alla domanda "a quale operazione può eccedere questo attore?" useremo una capability.

Flusso di controllo globale

In un sistema Object Oriented una sequenza di azioni include prendere decisioni su quali operazioni eseguire ed in che ordine. Queste decisioni sono basate su eventi esterni generati da attori o causati dal trascorrere del tempo. In questa fase verrà scelto il che modo gestire il flusso di controllo tra le opzioni possibili (Procedure-driver control, Event-driven control, Threads).

Condizioni limite

Le condizioni limite del sistema includono lo startup, lo shutdown, l'inizializzazione e le gestione di fallimenti come corruzione di dati, caduta di connessione e caduta di componenti.
A tale scopo varranno elaborati dei casi d'uso che specificano la sequenza di operazioni in ciascuno dei casi sopra elencati. In generale per ogni oggetto persistente, si esamina in quale caso d'uso viene creato e distrutto. Per ogni componente vanno aggiunti tre casi d'uso per l'avvio, lo shutdown e per la configurazione.

Per ogni tipologia di fallimento delle componenti bisognerà specificare come il sistema si accorge di tale situazione, come reagisce e quali sono le conseguenze.

10. Architettura Software proposta

Overview:

Il Sistema è stato decomposto utilizzando l'architettura "Plan-Do-Check-Act"

Il livello "Plan" consente di :
-) Definire il problema / impostare il progetto
-) Documentare la situazione di partenza
-) Analizzare il progetto
-) Pianificare le operazioni da realizzare

Il livello "Do" consente di :
-) Identificare soluzioni e miglioramenti
-) Gestire il miglioramento come un vero e proprio progetto
-) Adottare strumenti di analisi quantitativa delle grandezze coinvolte nel miglioramento

Il livello "Check" consente di :
-) Verificare i risultati ottenuti rispetto agli obbiettivi prefissati
-) Se si è raggiunto l'obbiettivo : passare a fase 1 ACT
-) Se non si è raggiunto l'obbiettivo : passare a fase 2 ACT

Il livello "Act" consente di :

1) Obbiettivo raggiunto :
   - Standardizzare, consolidare e addestrare gli operatori

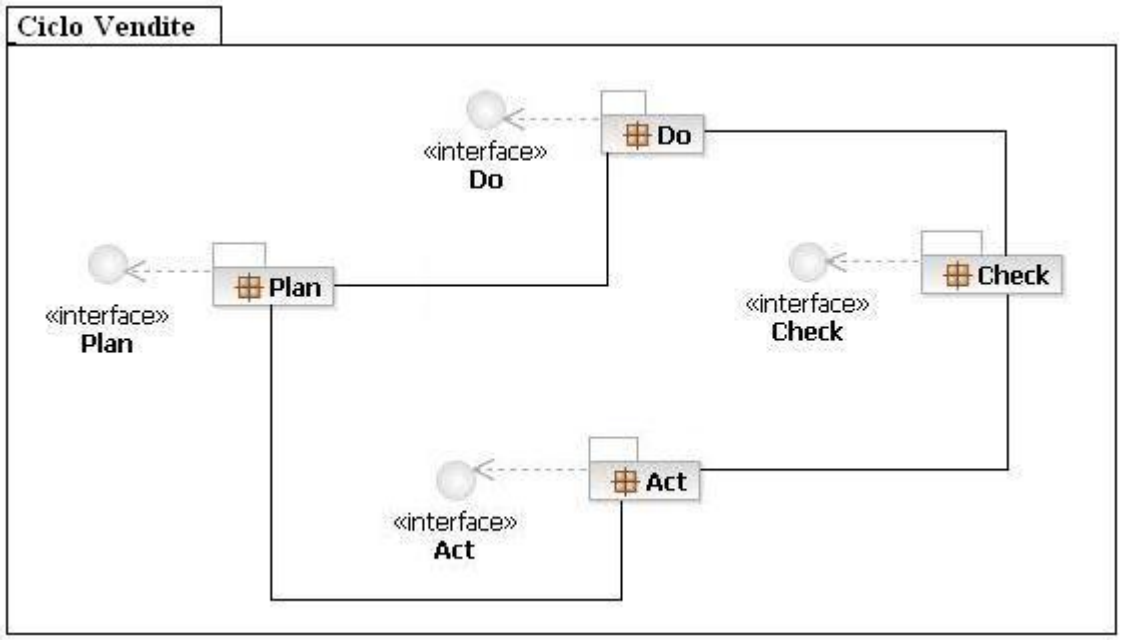
2) Obbiettivo non raggiunto :
   - Ripetere il ciclo PDCA sullo stesso problema analizzando criticamente le varie fasi del ciclo precedente ai fini di individuarne le cause del non raggiungimento degli obbiettivi

11. Decomposizione in sottosistemi

Il sistema sarà diviso nei seguenti sottosistemi:

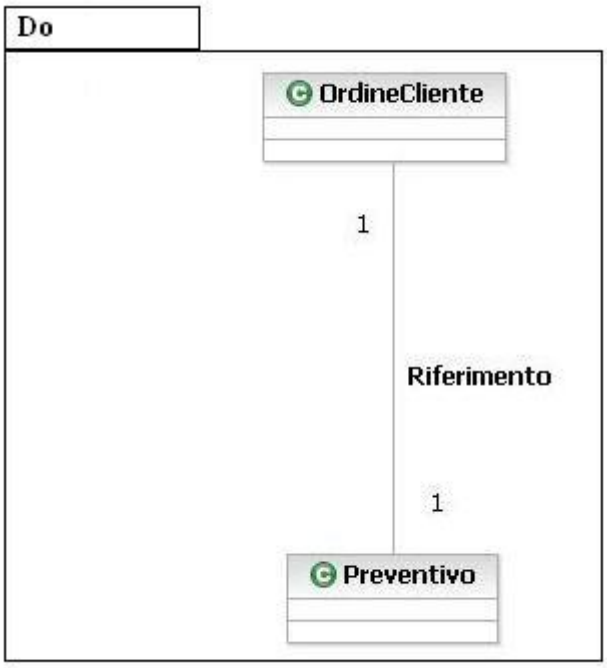
-)Sottosistema relativo al “Ciclo Delle Vendite”
-)Sottosistema relativo al “Ciclo Degli Acquisti”
-)Sottosistema per il Login Utente
-)Sottosistema del Magazzino(Gruppo 4)

Sistema Architettura PDCA – Ciclo Vendite



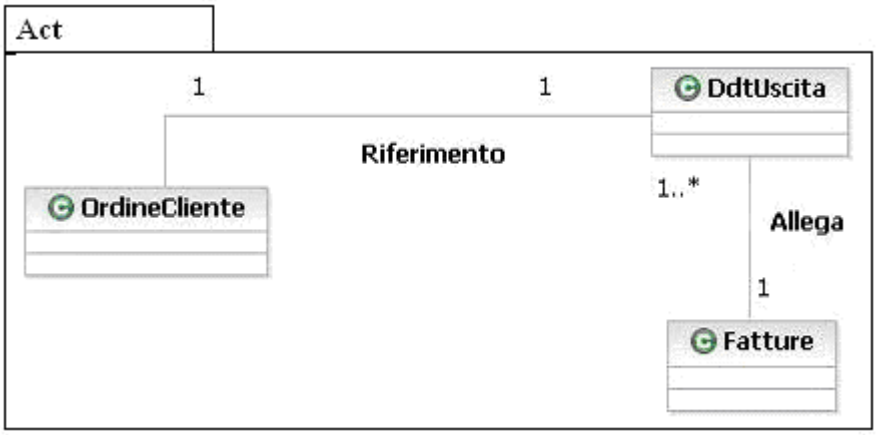
Il Package Plan – Ciclo Vendite

Il Package Do – Ciclo Vendite

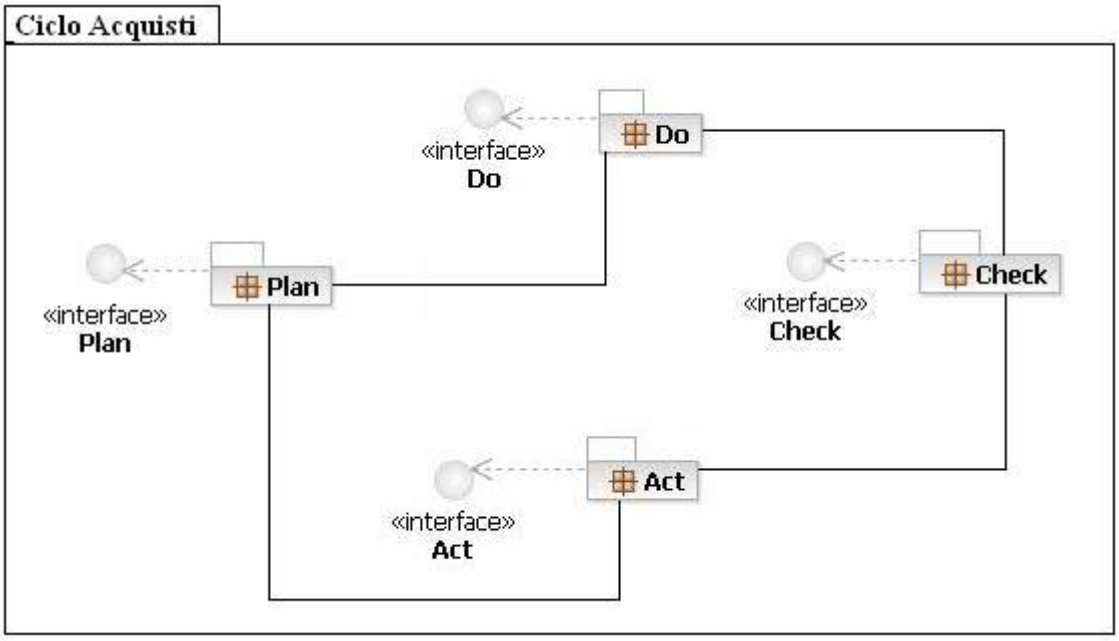


Il Package Check – Ciclo Vendite

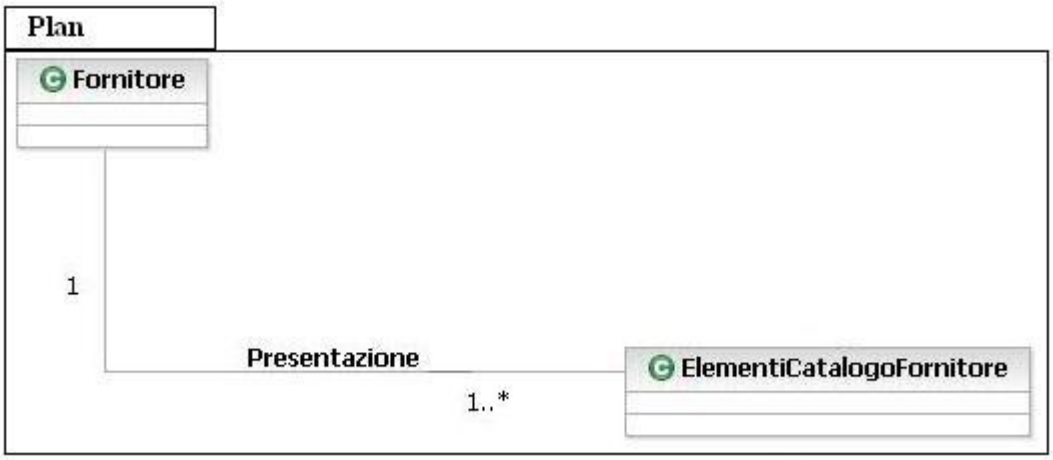
Il Package Act – Ciclo Vendite



Sistema Architettura PDCA – Ciclo Acquisti

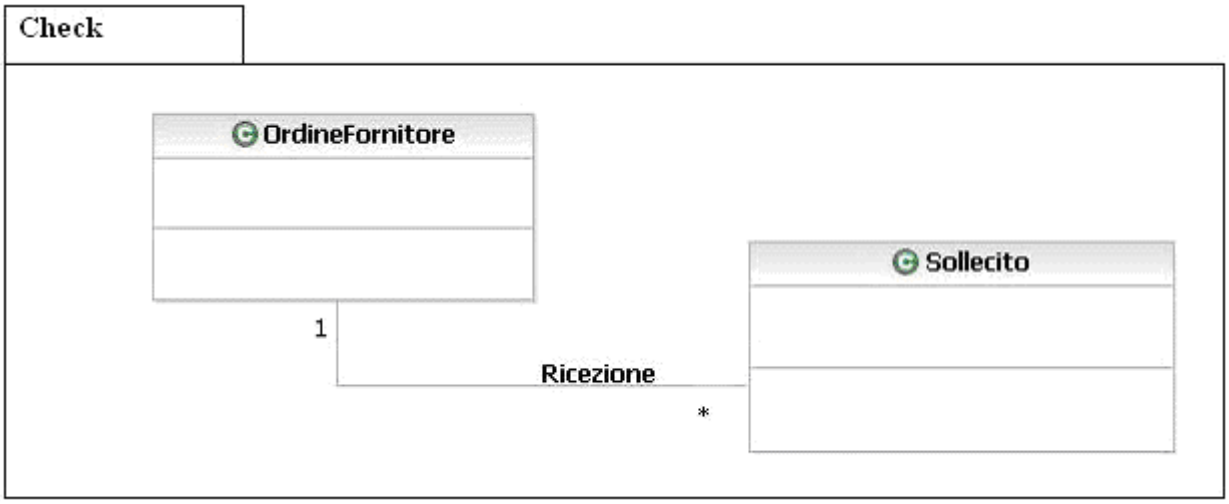


Package Plan – Ciclo Acquisti

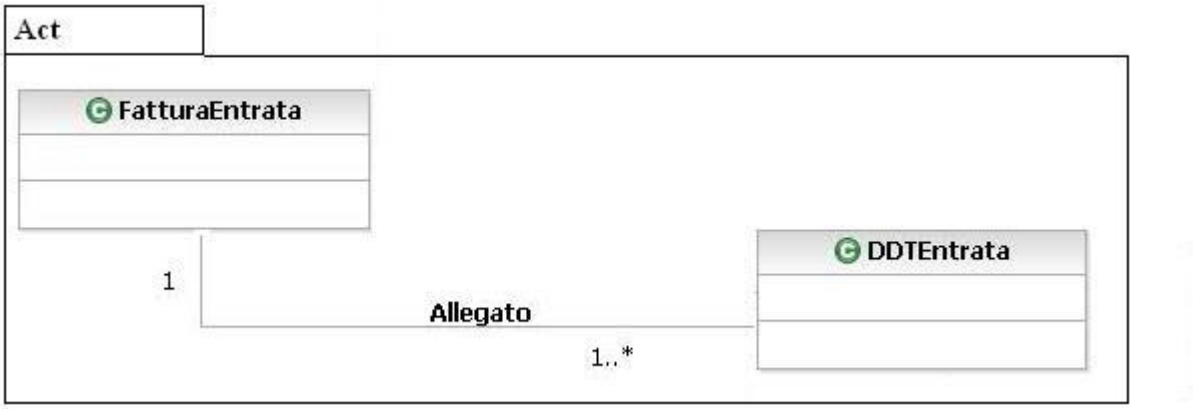


Package Do – Ciclo Acquisti

Package Check – Ciclo Acquisti



Package Act – Ciclo Acquisti



12. Scomposizione in Sottosistemi

Mapping Hardware / Software

Il sistema progettato presenta i seguenti nodi: Terminale e Base di Dati. Il nodo Terminale può essere utilizzato dalle due figure principali del nostro Sistema: Il Gestore del Polo Conciario e il Magazziniere. Sarà, dunque, composto da una rete interna che ci permette di gestirlo nella sua interezza. In questa rete interna ogni elemento dell'architettura è interconnesso tramite una rete dati digitale al server, che contiene il database del sistema. Il Sistema necessita di un server Linux / Windows con collegamento TCP/IP. Per la comunicazione utilizza il protocollo http e necessità di una Java Virtual Machine (maggiore J2se 1.5). Sul server deve risiedere un database relazionale (MySql). La capacità è una delle caratteristiche più importanti e non deve essere sottovalutata infatti si deve permettere di memorizzare una grande quantità di dati presenti nella base di dati. Il server deve garantire il backup dei dati presenti all'interno della base di dati e in più deve essere strutturato in modo da non rallentarsi notevolmente con l'aumentare degli accessi al Server MySql e con l'aumentare delle dimensioni di quest'ultimo.

13. Gestione dei dati persistenti

I dati necessari al sistema per un corretto funzionamento verranno memorizzati, tramite l'utilizzo del DBMS MySql, in modo persistente all'interno di un Database. Sarà il DBMS a gestire completamente la memorizzazione di informazioni in modo persistente all'interno delle varie tabelle. Tramite questi sarà possibile interrogare il DB ed effettuare tutte le relative operazioni. Quindi non vi sarà un utilizzo di file supplementari per memorizzare le informazioni presenti nel DB, dato che tali informazioni sono memorizzate in modo persistente dal DBMS.

14. Controllo Accessi

Gli accessi al Sistema sono controllati appena si tenta di utilizzare una specifica modalità. Infatti comparirà a video un finestra dove, per accedere alla modalità dovuta, dovrà essere inserita una login e una password personale. La login è uguale per tutti i Clienti , mentre la password è unica per ognuno. Questo evita che possono avvenire accessi alle funzionalità da parte di persone che non sono autorizzate, mantenendo in tal modo l'integrità del Sistema. Questo è importante poiché vengono gestiti dati relativi ai Clienti ed eventuali fughe di informazioni potrebbe causare seri problemi alla società.

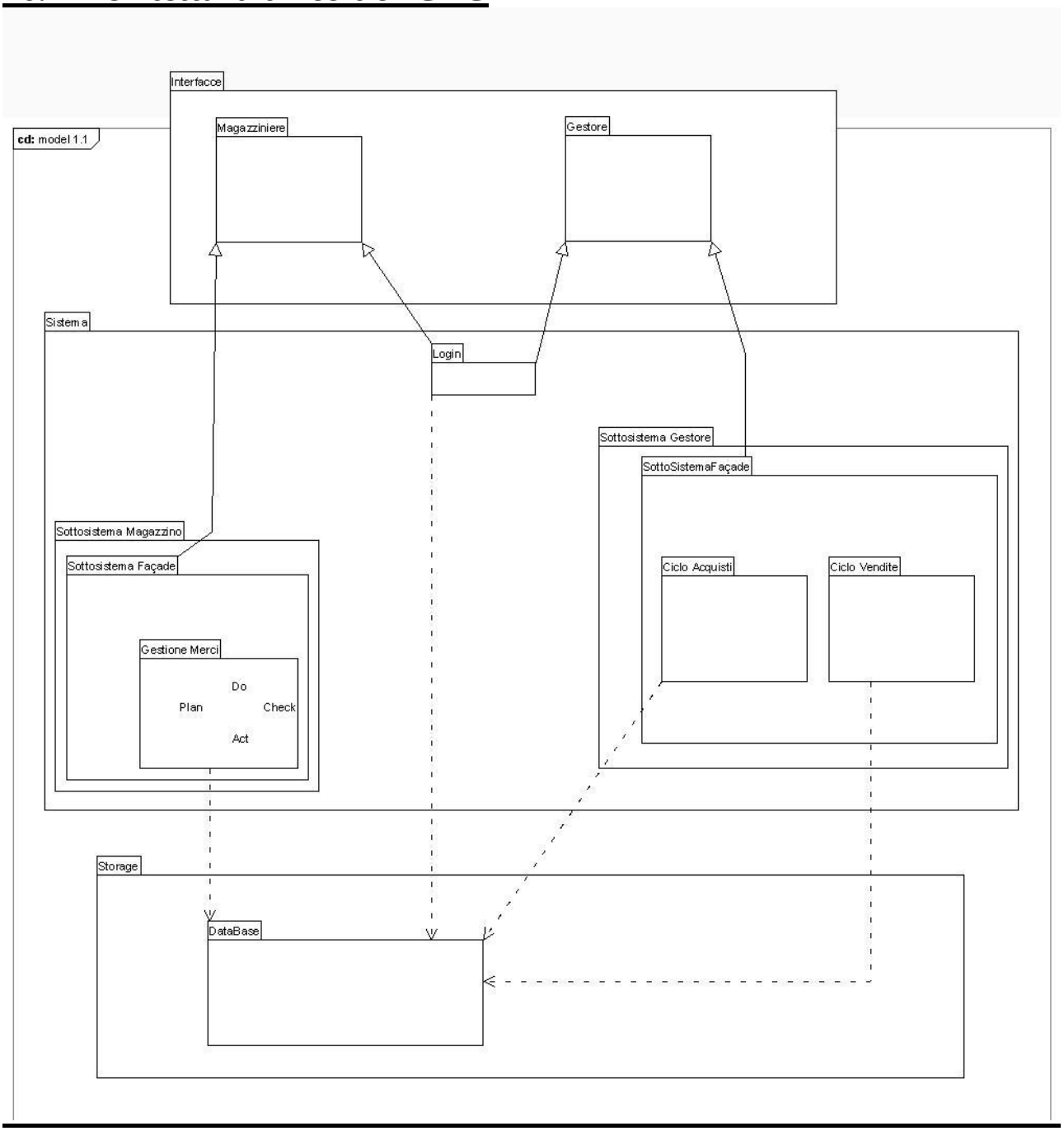
Presentiamo quindi una matrice degli accessi per esplicare meglio la politica di controllo degli accessi utilizzata:

Partecipanti/Oggetti	GestioneDati	Magazzino
Gestore	Gestione Account () Gestione Clienti () Gestione Fornitori () Gestione Ordini Fornitore () Gestione Ordini Cliente () Gestione Fatture () Gestione DDT () Gestione Elementi Catalogo () Gestione Elementi Catalogo Fornitore () Gestione Listini () Gestione Solleciti ()	Visualizza Magazzino ()
Magazziniere	Elenca Ordini Fornitore () Elenca Ordini Cliente ()	**Delegato al Gruppo 4**

15. Flusso di controllo globale

Il sistema è stato decomposto usando l'architettura su tre livelli "Three – tier" e poi integrato con l'approccio "Plan-Do-Check-Act".
Il Livello Superiore(**layer *interface***)è implementato dai sottosistemi "Gestore" e "Magazziniere" che comprendono tutti i boundary accessibili dagli utenti che utilizzano il nostro Sistema.
Il Livello intermedio (layer *application logic*) include tutti i sottosistemi che controllano le entità presenti nel sistema.
Infine il Livello Inferiore (layer *storage*) è implementato dal sottosistema "Storage" che permette la memorizzazione delle informazioni persistenti e l'interrogazione.
L'architettura Three-tier permette di individuare facilmente una gerarchia tra i sottosistemi, organizzati secondo i layer dell'architettura stessa e il modello PDCA.
I sottosistemi del layer application logic sono tutti accessibili dal layer interface e tutti accedono al layer storage per ottenere o conservare le informazioni collegate alle entità.
Inoltre, all'interno del layer application logic alcuni sottosistemi comunicano tra loro per ottenere informazioni.

16. Architettura three-tier GPC



17. Condizioni Limite

Saranno riportate alcune condizioni limite del Sistema. Saranno prese in considerazione le seguenti condizioni limite .

SOTTOSISTEMA: TerminaleUtente	
START-UP	L'Utente accede al Sistema mediante Login e Password.
SHUT DOWN	L'Utente esce dal Sistema.
POSSIBILI ERRORI	1. Accesso non consentito per Login e/o Password errate. 2. Il Sistema si disconnette momentaneamente per via di un fallimento hardware.

SOTTOSISTEMA: GestioneCliente/Fornitore	
START-UP	Il Gestore vuole registrare un nuovo cliente/fornitore
SHUT DOWN	Il Gestore ha completato l'operazione di registrazione del nuovo cliente/fornitore
POSSIBLI ERRORI	1. Momentanea disconnessione del Sistema per fallimento hardware. 2. Dati inseriti in modo scorretto. 3. Aggiornamento non effettuato.

SOTTOSISTEMA: GestioneDati (Generica)	
START-UP	Il Gestore vuole modificare dei dati all'interno del DataBase.
SHUT DOWN	Il Gestore ha completato la sua operazione di aggiornamento dei dati all'interno del DataBase.
POSSIBILI ERRORI	1. Dato da aggiornare riguarda un oggetto (entità) non preesistente all'interno del DataBase. 2. Inserimento di un codice oggetto errato 3. Momentanea disconnessione del Sistema dal DataBase. 4. Dati inseriti in maniera errata. 5. La scheda che compare non è quella desiderata. 6. Errore nell'aggiornamento della scheda

SOTTOSISTEMA: GestioneMessaggio	
START-UP	L'utente vuole gestire (Visualizzare/Inserire/Cancellare) un messaggio
SHUT DOWN	L'utente ha completato la sua operazione sui messaggi.
POSSIBILI ERRORI	1. Oggetto del messaggio non specificato 2. Testo del messaggio non specificato 3. Nessun messaggio presente nel DB. 4. Momentanea disconnessione del Sistema dal DataBase. 5. Errore nell'operazione sui messaggi.

18. SCRIPT CREAZIONE DATABASE :

```
drop database GPC;

create database GPC;

use GPC;

create table Cliente(
Codice int (10) not null auto_increment,
CodFiscale varchar (20) not null,
Categoria varchar (20) not null,
Nome varchar(20) not null,
Cognome varchar (20) not null,
Indirizzo varchar (50) not null,
Localita varchar (20) not null,
Cap int (5) not null,
SiglaProv char (2) not null,
Fiducia decimal (10,2) not null,
RagSociale varchar (50) not null,
Telefono varchar(20) not null,
Cellulare varchar (15) not null,
PagamentoPredefinito varchar (20) not null,
primary key (Codice)
);


create table Fornitore(
PIva varchar (11) not null,
Categoria varchar (20) not null,
RagSociale varchar (50) not null,
Indirizzo varchar (50) not null,
Localita varchar (20) not null,
Cap int (5) not null,
SiglaProv char (2) not null,
Fiducia varchar (20) not null,
Telefono varchar(20) not null,
Cellulare varchar (15) not null,
PagamentoPred varchar (20) not null,
ScontoPredef decimal  (4,2) not null,
Rappresentante varchar (50) not null,
primary key (PIva)
);

create table PrimaNotaCliente(
idPrimaNota int(10) not null,
CodCl int (10) not null,
```

```
descrizione varchar(50) not null,
dare decimal (10,2) default 0,
avere decimal (10,2) default 0,
primary key (idPrimaNota,CodCl),
foreign key (CodCl) references Cliente (Codice) on delete restrict
);


create table PrimaNotaFornitore(
idPrimaNota int(10) not null,
CodFor varchar (11) not null,
descrizione varchar(50) not null,
dare decimal (10,2) default 0,
avere decimal (10,2) default 0,
primary key (idPrimaNota,CodFor),
foreign key (CodFor) references Fornitore (PIva) on delete restrict
);

create table listino (
idlistino int (10) not null auto_increment,
descrizione varchar (20) not null,
percentuale decimal (4,2) not null,
primary key (idlistino)
);

create table Preventivo(
IdPrev int (10) not null,
CodCliente int (10) not null,
DataPreventivo date not null,
listino int (10) default null,
primary key ( IdPrev , CodCliente),
foreign key (CodCliente) references Cliente (Codice) on delete cascade,
foreign key (listino) references listino (idlistino) on delete restrict
);

create table ElementiCatalogoAzienda (
idElemento int (10) not null auto_increment,
descrizione varchar(50) not null,
prezzounitario decimal(10,2) not null,
inProduzione  set( "SI", "NO") default "SI",
primary key (idElemento)
);

create table Comprende (
idPrev int (10) not null,
idElemento int (10) not null,
primary key (idPrev, idElemento),
```

```
foreign key (idPrev) references Preventivo (IdPrev) on delete restrict,
foreign key (idElemento) references ElementiCatalogoAzienda (idElemento) on
delete restrict
);

create table ordineCliente (
idOrdine int (10) not null auto_increment,
idCliente int (10) not null,
listino int (10) default null,
dataOrdineRichiesta date not null,
dataOrdineArrivo date not null,
imponibile decimal(10,2) not null,
stato varchar (15) default 'in valutazione',
primary key (idOrdine),
foreign key (idCliente) references Cliente (Codice) on delete restrict,
foreign key (listino) references listino (idlistino) on delete restrict
);


create table ElementiCatalogoFornitore (
idElemento int (10) not null,
idFornitore varchar (11) not null,
descrizione varchar(50) not null,
prezzounitario decimal(10,2) not null,
inProduzione set( "SI", "NO") default "SI",
primary key (idElemento,idFornitore),
foreign key (idFornitore) references Fornitore (PIva) on delete restrict
);

create table OCCompostoDa (
idOrdineCliente int (10) not null,
idElemento int (10) not null,
primary key (idOrdineCliente, idElemento),
foreign key (idOrdineCliente) references ordineCliente (idOrdine) on delete restrict,
foreign key (idElemento) references ElementiCatalogoAzienda (idElemento) on
delete restrict
);

create table ordineFornitore (
idOrdine int (10) not null auto_increment,
idFornitore varchar (11) not null,
imponibile decimal (10,2) not null,
dataOrdineEffettuato date not null,
dataOrdineArrivo date not null,
primary key (idOrdine),
foreign key (idFornitore) references Fornitore (PIva) on delete restrict
);
```

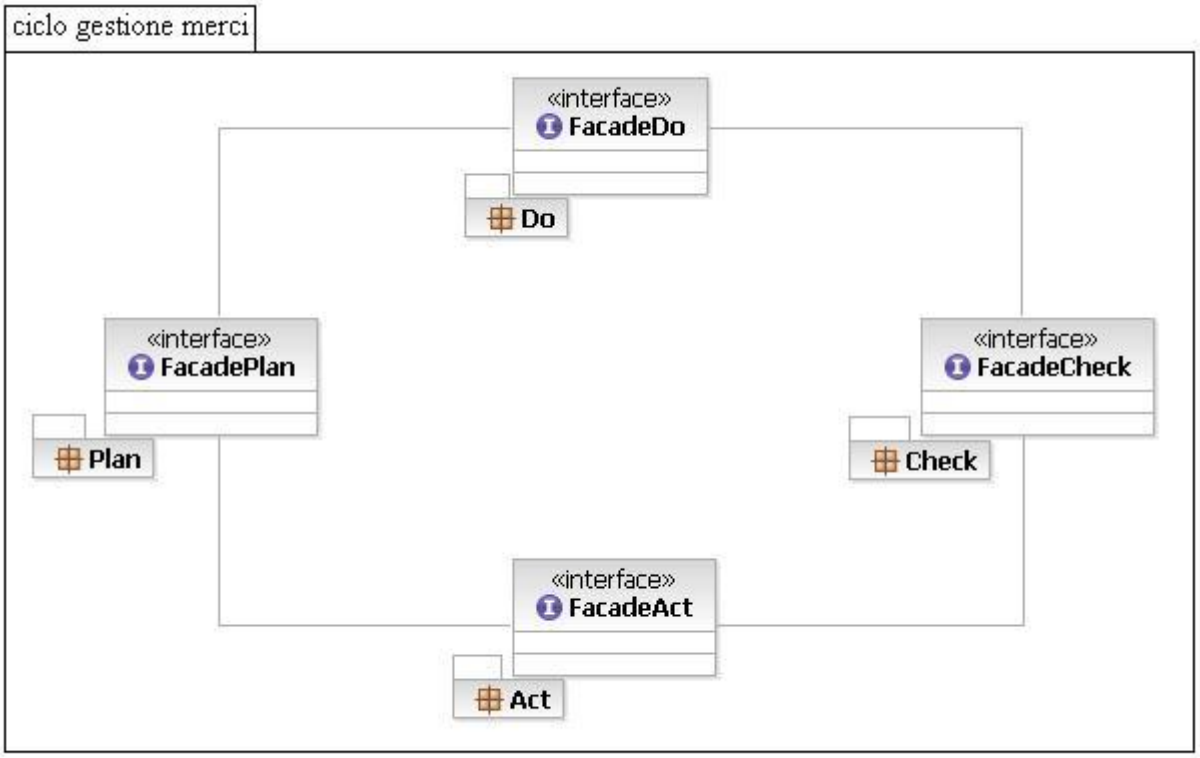
```
create table OFCompostoDa (
idOrdineFornitore int (10) not null,
idElemento int (10) not null,
primary key (idOrdineFornitore, idElemento),
foreign key (idOrdineFornitore) references ordineFornitore (idOrdine) on delete
restrict,
foreign key (idElemento) references ElementiCatalogoFornitore (idElemento) on
delete restrict
);

create table solleciti(
idSollecito int (5) not null,
idFornitore varchar (11) not null,
DataSoll date not null,
descrizione varchar (40) not null,
primary key (idSollecito,idFornitore),
foreign key (idFornitore) references Fornitore (PIva) on delete restrict
);

create table utente (
idUtente int (10) not null auto_increment,
nome varchar (20) not null,
cognome varchar (20) not null,
indirizzo varchar (50) not null,
telefono varchar (20) not null,
primary key (idUtente)
);

create table accountUtente (
idAccount int (10) not null auto_increment,
idUtente int (10) not null,
username varchar (10) not null,
passwd varchar (10) not null,
tipologia varchar (20) not null,
primary key (idAccount),
foreign key (idUtente) references utente (idUtente) on delete cascade
);

create table fatturaEntrata (
idFattura int (10) not null auto_increment,
numFattura int (10) not null,
dataFattura date not null,
iva decimal (10,2) not null,
totaleFattura decimal(10,2) not null,
primary key (idFattura)
);
```

```
create table fatturaUscita (
idFattura int (10) not null auto_increment,
numfattura int (10) not null,
dataFattura date not null,
listino int (10) default null,
iva decimal (10,2) not null,
totaleFattura decimal(10,2) not null,
primary key (idFattura),
foreign key (listino) references listino (idListino) on delete restrict
);

/* simile a quella che sarà fornita dal gruppo 4*/
create table ddtEntrata (
idDDT int (10) not null auto_increment,
dataDDT date not null,
idFatturaEntrata int (10) default null,
primary key (idDDT),
foreign key (idFatturaEntrata) references fatturaEntrata (idFattura) on delete restrict
);

/* simile a quella che sarà fornita dal gruppo 4*/
create table ddtUscita (
idDDT int (10) not null auto_increment,
dataDDT date not null,
idFatturaUscita int (10) default null,
primary key (idDDT),
foreign key (idFatturaUscita) references fatturaUscita (idFattura) on delete restrict
);
```

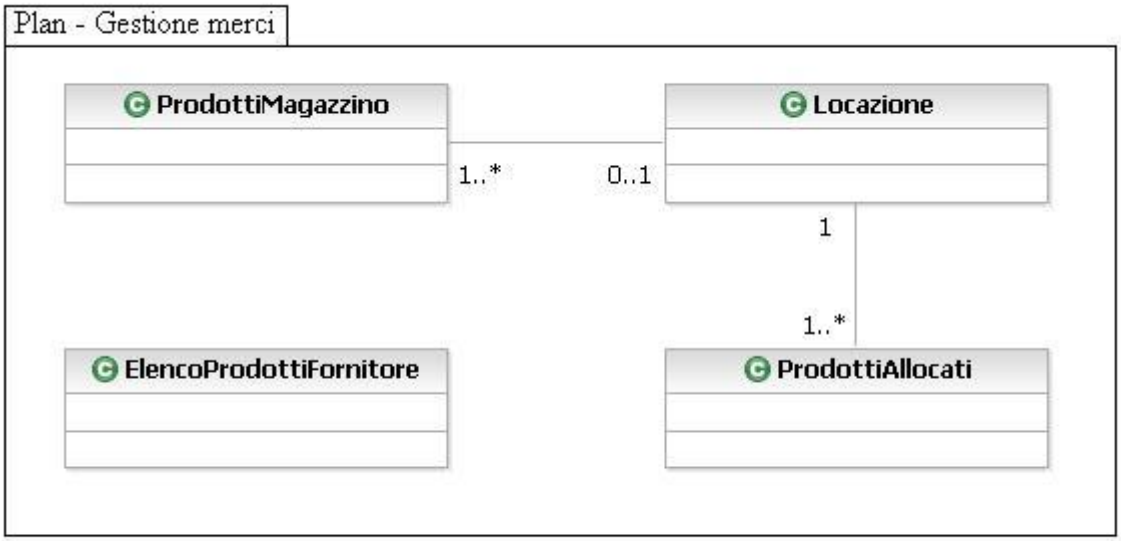
---

19. Materiale del Gruppo 4

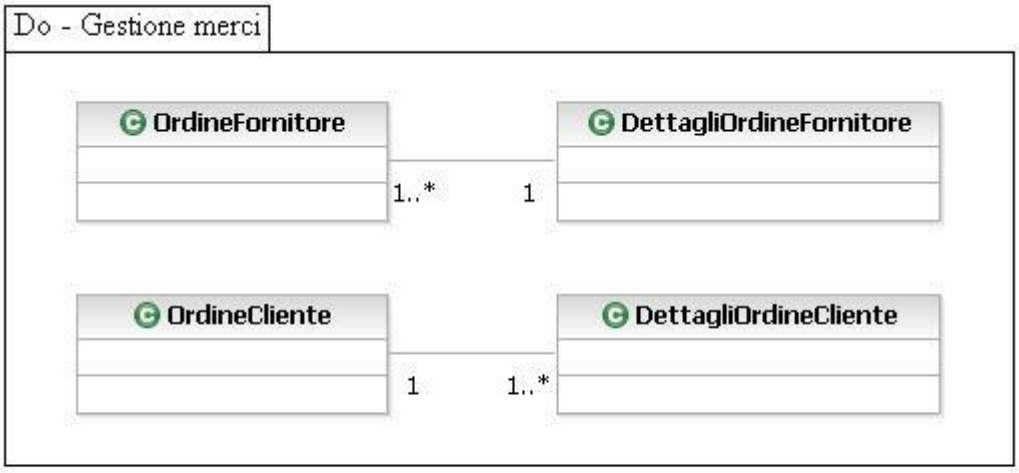
Plan – Do – Check – Act



Plan Merci

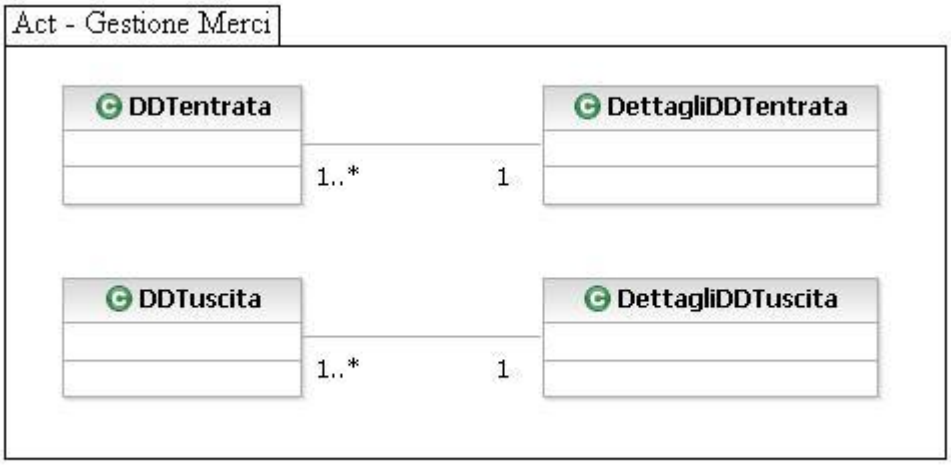


Do Merci



Check Merci

Act Merci



Class Diagram