Progetto di un sistema gestionale per la “Spedizione Pacchi”

Esercitazione UML 2

Corso di Ingegneria del Software, prof. Stefano Russo

SOMMARIO

1 REQUISITI D'UTENTE

1.1 SERVIZI DEL SISTEMA DI GESTIONE SPEDIZIONI

Una compagnia di spedizioni richiede la realizzazione di un sistema gestionale a supporto del processo aziendale delle spedizioni. L'applicazione deve offrire un'interfaccia semplice e intuitiva per l'utilizzo da parte dei dipendenti della compagnia: essa prevede una suddivisione in categorie per i diversi servizi offerti:

- Spedizione
- Ricerca
- Gestione Clienti
- Gestione Pagamento e Fatturazione

È prevista inoltre una "sezione offerte" dove il direttore commerciale può aggiungere delle offerte speciali limitate nel tempo che sono automaticamente applicate durante il processo di spedizione dei pacchi.

È necessario che i clienti si registrino alla compagnia prima di usufruirne dei servizi. Durante la procedura di registrazione il cliente deve fornire:

- Cognome e nome
- Data di nascita
- Indirizzo
- Indirizzo e-mail
- Codice fiscale

Nel caso in cui il cliente sia un'azienda, la registrazione richiede:

- Denominazione
- Indirizzo
- Indirizzo e-mail
- Partita Iva

Per accedere alle funzionalità del sistema, ogni dipendente, compreso il direttore commerciale, deve identificarsi fornendo username e password.

1.2 SPEDIZIONE

Un dipendente prende in consegna il pacco del cliente ed avvia il processo di spedizione. Il processo richiede gli indirizzi del mittente e del destinatario: i dati del mittente sono ottenuti dai dati forniti dal cliente all'atto della registrazione (la registrazione deve essere stata precedentemente effettuata dal cliente; nel caso in cui il cliente non si sia registrato prima di una spedizione, l'impegato può effettuare egli stesso la registrazione del cliente quando questi si presenta per la prima volta per una spedizione); quelli del destinatario devono essere forniti dal cliente contestualmente alla spedizione.

Il costo di una spedizione prevede una componente base a cui si possono aggiungere extra: il costo base varia in funzione del peso e della dimensione del pacco (per scopi didattici trascuriamo la

distanza). Il peso è espresso in Kg, mente la dimensione del pacco può essere “piccola” o “grande”. Il costo base si calcola sommando 10 Euro per ogni Kg di peso, più 5 Euro se la dimensione del pacco è “piccola”, altrimenti 10 Euro se la dimensione è “grande”. Il cliente può scegliere tra la spedizione “standard” o “premium”. La spedizione “standard” ha tempi di consegna tra sette e dieci giorni lavorativi; la spedizione “premium” ha tempi di consegna tra uno e tre giorni lavorativi. La spedizione “premium” comporta un aumento del 10% del costo della spedizione. Infine, il cliente sceglie la modalità di pagamento: tale scelta ed il pagamento possono essere effettuati al momento o differiti. Una spedizione si considera come *non valida*, finché il pagamento non è completato: le spedizioni non convalidate rimangono in giacenza. Al termine della procedura, il sistema deve assegnare un codice identificativo alla spedizione.

1.3 RICERCA

La funzione di ricerca restituisce informazioni sulla posizione del pacco. Il pacco può essere (1) in movimento lungo una tratta, (2) in giacenza presso un magazzino, oppure (3) consegnato al destinatario. I magazzinieri aggiornano la posizione dei pacchi all’ingresso e all’uscita dei colli dal proprio magazzino. È infine compito del corriere aggiornare lo stato della spedizione una volta consegnato il pacco.

1.4 GESTIONE PAGAMENTO E FATTURAZIONE

Una spedizione può essere pagata tramite:

- Bollettino postale
- Bonifico bancario
- Carta di credito
- Carta di debito
- Carta prepagata
- Contanti

Al pagamento il cliente può richiedere una fattura. La fattura può essere consegnata in formato cartaceo, elettronico – inviandola alla casella e-mail del cliente – oppure in entrambe le forme.

1.5 SEZIONE OFFERTE

Il direttore commerciale può impostare delle offerte speciali per alcuni tipi di spedizione, che comportano uno sconto definito tra 5% e il 20% sul costo finale della spedizione. Ciascuna offerta si applica esclusivamente ad un tipo di spedizione.

Le offerte sono disponibili solo per un tempo limitato e sono applicate automaticamente alle nuove spedizioni che rientrano nelle promozioni.

1.6 PERSISTENZA DEI DATI

Il sistema deve memorizzare in un *database* tutte le informazioni riguardanti: i clienti registrati, le offerte, e le spedizioni, in particolare tenendo traccia per queste ultime del loro ultimo stato e di quello storico.

2 DOCUMENTO DI SPECIFICA DEI REQUISITI

Il capitolo precedente elenca i *requisiti utente* (*User Requirements*), che costituiscono una descrizione poco strutturata di esigenze e funzionalità che un committente potrebbe fornire per descrivere dal suo punto di vista il sistema che richiede sia realizzato.

La prima fasi di lavoro consiste nell'analizzare i requisiti utente e quindi specificare i requisiti software (*Software Requirements*), in maniera il più possibile completa, non ambigua, e senza inconsistenze o contraddizioni. Da questo flusso di lavoro otteniamo due output: il *documento di specifica dei requisiti* e il *modello dei casi d'uso*.

Per ottenere una descrizione dettagliata delle caratteristiche delle funzionalità e delle proprietà che il sistema da realizzare deve possedere, conduciamo una analisi dei requisiti del sistema. Il *documento di specifica dei requisiti* (chiamato SRS – Software Requirement Specification o DSR – Documento di Specifica dei Requisiti) conclude la fase di analisi e specifica dei requisiti e spesso costituisce la base contrattuale con il committente.

L'analista si concentra sul *problema*, individuando gli elementi del dominio del problema, le relazioni tra di essi, e i requisiti. Se opportuno, redige un *glossario di progetto*, utile per determinare i vocaboli che sono usati nella documentazione. Lo scopo della specifica dei requisiti è definire i requisiti *funzionali* e *non funzionali* del sistema da realizzare, identificando e risolvendo gli elementi che possono generare ambiguità.

I requisiti possono essere specificati in maniera *formale*, *semi-formale* o *informale*. Le specifiche sono dette *operazionali* se descrivono i requisiti del sistema in termini del loro comportamento desiderato; sono dette specifiche *descrittive*, se sono poste in forma dichiarativa e asseriscono le proprietà che il sistema deve avere. Lo stile da preferire per le specifiche deve essere scelto in base alla grandezza e complessità del sistema, in base al grado di ambiguità che i requisiti posseggono, ed anche in base al processo di sviluppo o a vincoli normativi. Tipicamente, un documento di specifica dei requisiti adotta nelle sue parti differenti livelli di rigore, in maniera commisurata al livello di ambiguità che una specifica può generare.

Sebbene non vi sia uno standard, generalmente un requisito si presenta in una forma del tipo:

<ID> <SISTEMA> <TIPO> <PRIORITÀ>

ove:

- <ID> è un identificativo univoco del requisito;
- <SISTEMA> è il sistema o sua parte a cui si attribuisce;
- <TIPO> indica la tipologia del requisito; esempi sono:
    - FUN: indica un requisito su una funzionalità del sistema;
    - PRE: indica un requisito prestazionale;
    - PER: indica un requisito di persistenza dei dati;
    - AFF: indica un requisito di affidabilità;
    - DIS: indica un requisito di disponibilità del sistema software;
    - SIC: indica un requisito di sicurezza.
- <PRIORITÀ> può assumere valori quali "obbligatorio", "opzionale", …;

- Spesso si adotta il metodo MoSCoW per attribuire le priorità, che identifica quattro categorie di priorità: Must, Must not, Should, Could, Won't/Would.

Per esempio, detto SMS l'intero sistema (*Shipment Management System*), i requisiti vanno raccolti, catalogati e specificati ciascuno con frasi sempliciche esprimano in maniera chiara la caratteristica (funzionalità, fattore di qualità, o altro) che il sistema da realizzare deve possedere, per es. nel modo seguente:

DATI.1 SMS PER <must> - Il sistema deve memorizzare tutte le informazioni riguardanti i clienti registrati in un database relazionale esterno per garantirne la persistenza.

SEC.1 SMS SIC <must> - Ciascun utilizzatore del sistema deve autenticarsi tramite username e password per accedere ai suoi servizi.

SEC.2 SMS SIC <must not> - Il sistema NON DEVE memorizzare in maniera persistente alcun dato sensibile del cliente relativo alla modalità di pagamento (es.: nr. della carta di credito).

2.1 Diagramma dei Casi d'Uso

Per specificare i requisiti funzionali del sistema è possibile impiegare i diagrammi dei casi d'uso UML, e completare la specifica con il *documento di specifica dei casi d'uso*.

Un diagramma dei casi d'uso mostra i confini del sistema, le funzionalità che offre, e gli attori che interagiscono con le sue funzionalità. I casi d'uso costituiscono la specifica di una sequenza di azioni, incluse eventuali sequenze alternative e sequenze di errore, che il sistema o una sua parte può eseguire interagendo opportunamente con gli attori (primari e secondari).

Per sviluppare un diagramma dei casi d'uso si può iniziare individuando i *ruoli* delle entità che interagiscono col sistema (ovvero, gli *attori*). Quindi ragioniamo sul *che cosa* gli attori fanno quando operano col sistema: un caso d'uso descrive una interazione tra l'utente e il sistema per svolgere *un'unità di lavoro utile*.

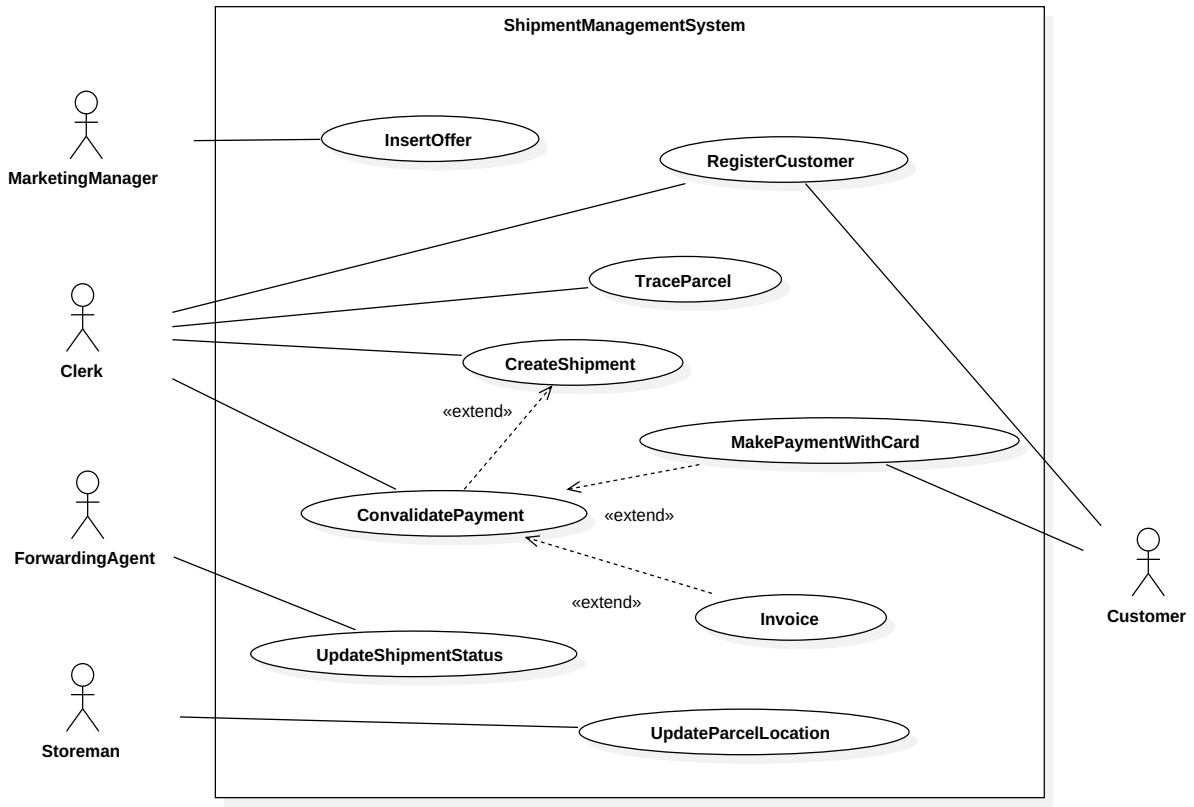

Figura 1 Use Case Diagram del Sistema di Gestione Spedizioni

2.2 SCENARI DEI CASI D'USO

I casi d'uso devono essere completati con una descrizione strutturata e dettagliata dell'interazione tra attori e sistema, la **specifica dei casi d'uso**. Una sequenza di eventi, definita come ***scenario** del caso d'uso,* elenca i passi che compongono un caso d'uso: un caso d'uso ha uno *scenario principale* e può avere uno o più *scenari secondari*, che seguono una sequenza di eventi alternativa. La ***specifica dei casi d'uso*** è costituita dall'insieme dei casi d'uso e dei loro scenari, e costituisce una specifica dei requisiti funzionali del sistema da realizzare. È opportuno documentare ciascun caso d'uso con una descrizione schematica, ad es. con una scheda strutturata. UML non definisce uno standard, e ogni azienda/organizzazione ne definisce uno proprio. Lo scenario principale di un caso d'uso può essere documentato con il generico schema mostrato in Tabella 1.

Tabella 1 Tipica struttura dello scenario principale di un caso d'uso

Caso d'uso	**<nome>**
ID	<id>
Breve Descrizione	
Attori primari	<attori che avviano il caso d'uso>
Attori secondari	<attori che partecipano al caso d'uso avviato>
Precondizioni	<stato del sistema prima di avviare il caso d'uso>
Sequenza degli eventi principali	<lista dei passi effettivi del caso d'uso>
Postcondizioni	<lo stato del sistema dopo aver eseguito il caso d'uso>
Sequenza degli eventi alternativa	

La Tabella 2 mostra la specifica dello scenario principale del caso d'uso *CreaSpedizione*.

Tabella 2 Specifica del caso d'uso RegistraCliente

Caso d'uso	**CreaSpedizione**
ID	CREA_SPED
Breve Descrizione	Creazione di una nuova spedizione
Attori primari	Sportellista
Attori secondari	Nessuno
Precondizioni	Il mittente è un cliente registrato al sistema
Sequenza degli eventi principali	1. Il caso d'uso inizia quando lo sportellista seleziona "crea nuova spedizione" 2. Lo sportellista inserisce peso e dimensione del pacco 3. Lo sportellista inserisce il tipo di spedizione 4. Lo sportellista inserisce l'identificativo del cliente come mittente 5. Se il destinatario è un cliente, a. Lo sportellista inserisce l'identificativo del cliente come destinatario, altrimenti b. Lo sportellista inserisce l'indirizzo del destinatario 6. Lo sportellista richiede l'inserimento dei dati 7. Il sistema ricava l'indirizzo del mittente dall'identificativo del cliente 8. Se il destinatario è un cliente, a. Il sistema ricava l'indirizzo del destinatario dall'identificativo del cliente 9. Il sistema crea la spedizione 10. Per ogni offerta nel sistema, a. Se l'offerta è applicabile, i. Il sistema applica l'offerta alla spedizione 11. Il sistema aggiorna il database con la nuova spedizione 12. Il sistema restituisce un messaggio di conferma e l'identificativo della spedizione 13. Se lo sportellista seleziona "convalida pagamento immediatamente", a. *punto di estensione*: ConvalidaPagamento
Postcondizioni	Il database contiene la nuova spedizione
Sequenza degli eventi alternativa	Nessuna

3 MODELLO DI ANALISI

Dopo aver individuato i casi d'uso del sistema, si passa all'analizzare singolarmente e in maniera iterativa ogni singolo caso d'uso in maniera tale da produrre:

- Le *classi di analisi*, che modellano i concetti chiave del dominio del problema (entità e relazioni tra entità);
- Le *realizzazioni dei casi d'uso*, che mostrano come le istanze delle classi di analisi interagiscono per realizzare il comportamento del sistema specificato da un caso d'uso.

Questo insieme di documenti costituisce il modello di analisi del sistema. Il modello di analisi utilizza termini propri del dominio del problema e non offre soluzioni progettuali o implementative, limitandosi all'analisi del problema (*what* versus *how*). Più un modello di analisi è semplice e di facile comprensione, più la sua qualità è migliore.

Particolare attenzione va posta al concetto di *processo iterativo*: esso indica che i manufatti (***artifact***) del modello possono essere raffinati e completati con l'acquisizione di nuove informazioni durante l'analisi.

In analisi il **diagramma delle classi** (*class diagram* – CD) è usato per creare un *modello concettuale* del dominio, ovvero esso fornisce una rappresentazione delle entità del mondo reale presenti nel dominio di interesse (***domain objects***) e cattura le relazioni tra di esse. Non occorre completare il CD con quei dettagli (e *ornamenti UML*) che non sono rilevanti ai fini di comprensione del dominio, come il tipo delle proprietà delle classi o la cardinalità delle associazioni. La specifica di questi elementi può essere rimandata alle fasi successive. Il CD di prima analisi ha dunque lo scopo di rappresentare i concetti significativi del dominio del problema, le astrazioni significative, la terminologia del problema e il contenuto informativo del dominio. Esso rappresenta altresì un'ontologia del dominio del problema. Il CD di prima analisi è mostrato in Figura 2.

Per individuare le classi di analisi è possibile sfruttare svariate tecniche, tra cui la più utilizzata è l'analisi nome/verbo:

- Nomi: candidati per classi o proprietà delle classi
- Verbi: candidati per le responsabilità delle classi

Una buona classe di analisi è un'astrazione ben definita se i suoi elementi rispettano il principio di massima coesione e di minimo accoppiamento.

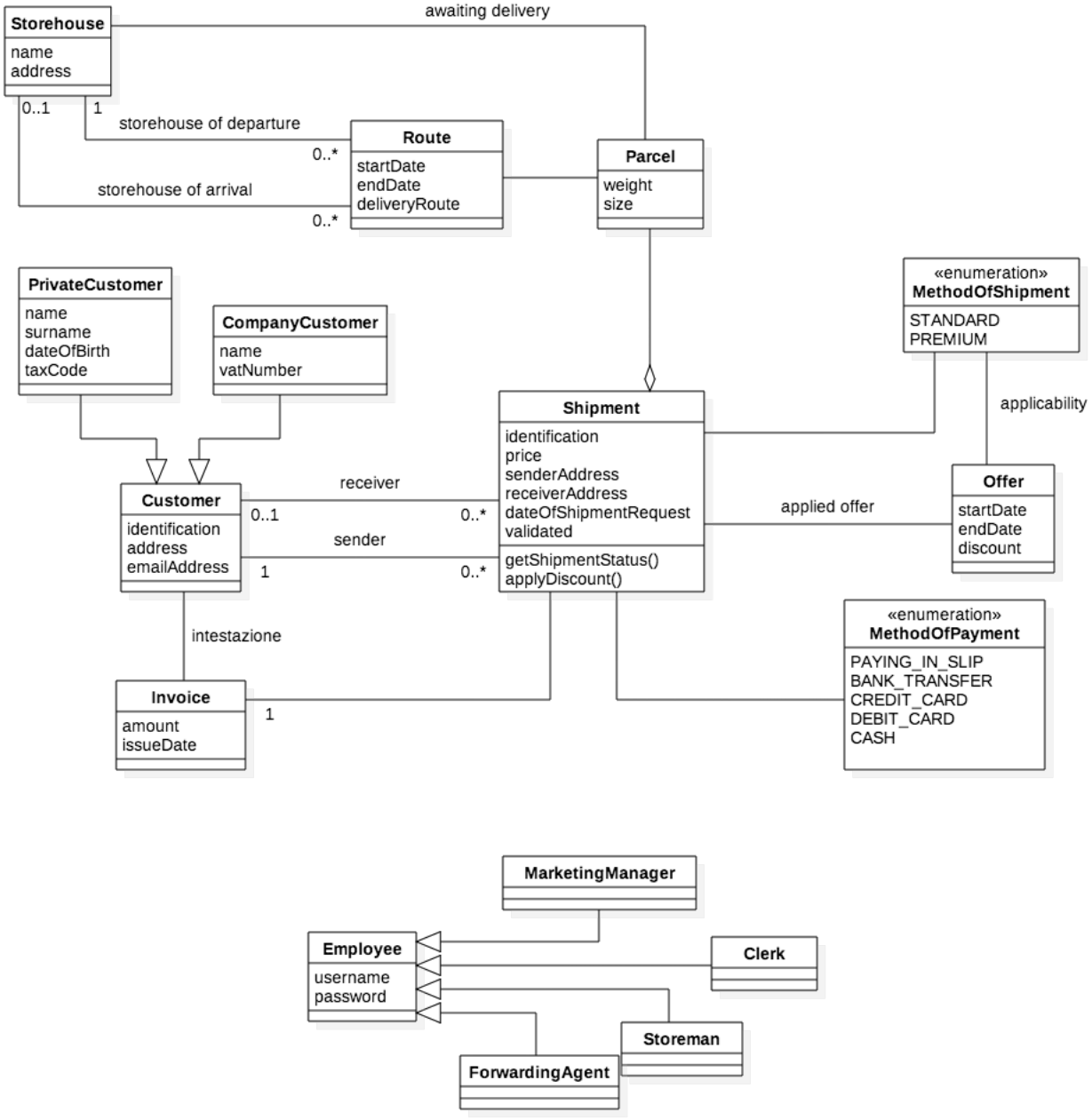

Figura 2 Class Diagram di Prima Analisi

Per modellare la *realizzazione dei casi d'uso*, si può usare il diagramma UML di sequenza. Il **diagramma di sequenza** (*sequence diagram* – SD) mostra la dinamica delle iterazioni tra gli attori e le entità del dominio per realizzare una funzionalità del sistema (uno o più scenari di un caso d'uso). I diagrammi di sequenza sono composti: da linee di vita (*lifeline*), che indicano i partecipanti di un'interazione; e da messaggi, che si scambiano le istanze durante l'interazione. Un SD di prima analisi è mostrato in Figura 3.

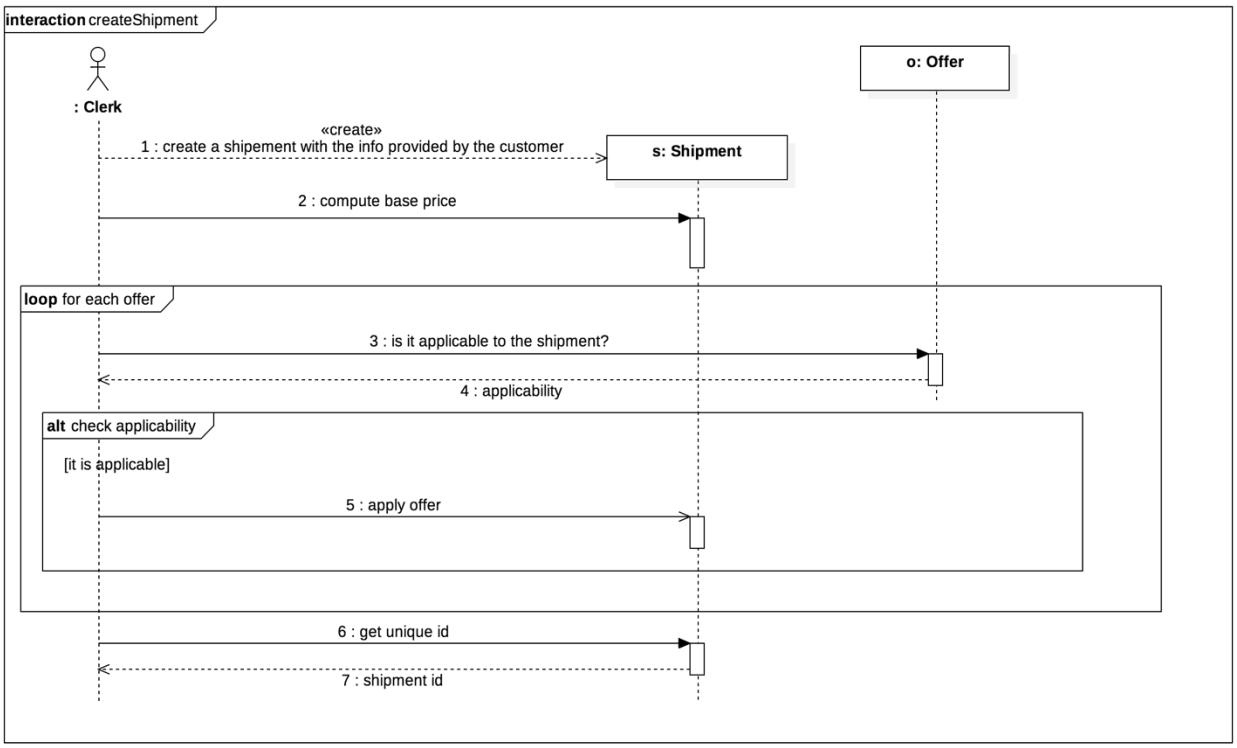

Figura 3 Sequence Diagram di Prima Analisi per la realizzazione del caso d'uso CreaSpedizione

Il modello di analisi si può raffinare in più iterazioni, aggiungendo dettagli, ornamenti UML, e rendendolo più completo, nella descrizione del dominio del problema e nell'interazione del sistema con gli attori. Ad esempio, in un raffinamento del **diagramma delle classi** aggiungiamo alle associazioni il loro tipo, la loro cardinalità, la navigabilità e i ruoli (Figura 4).

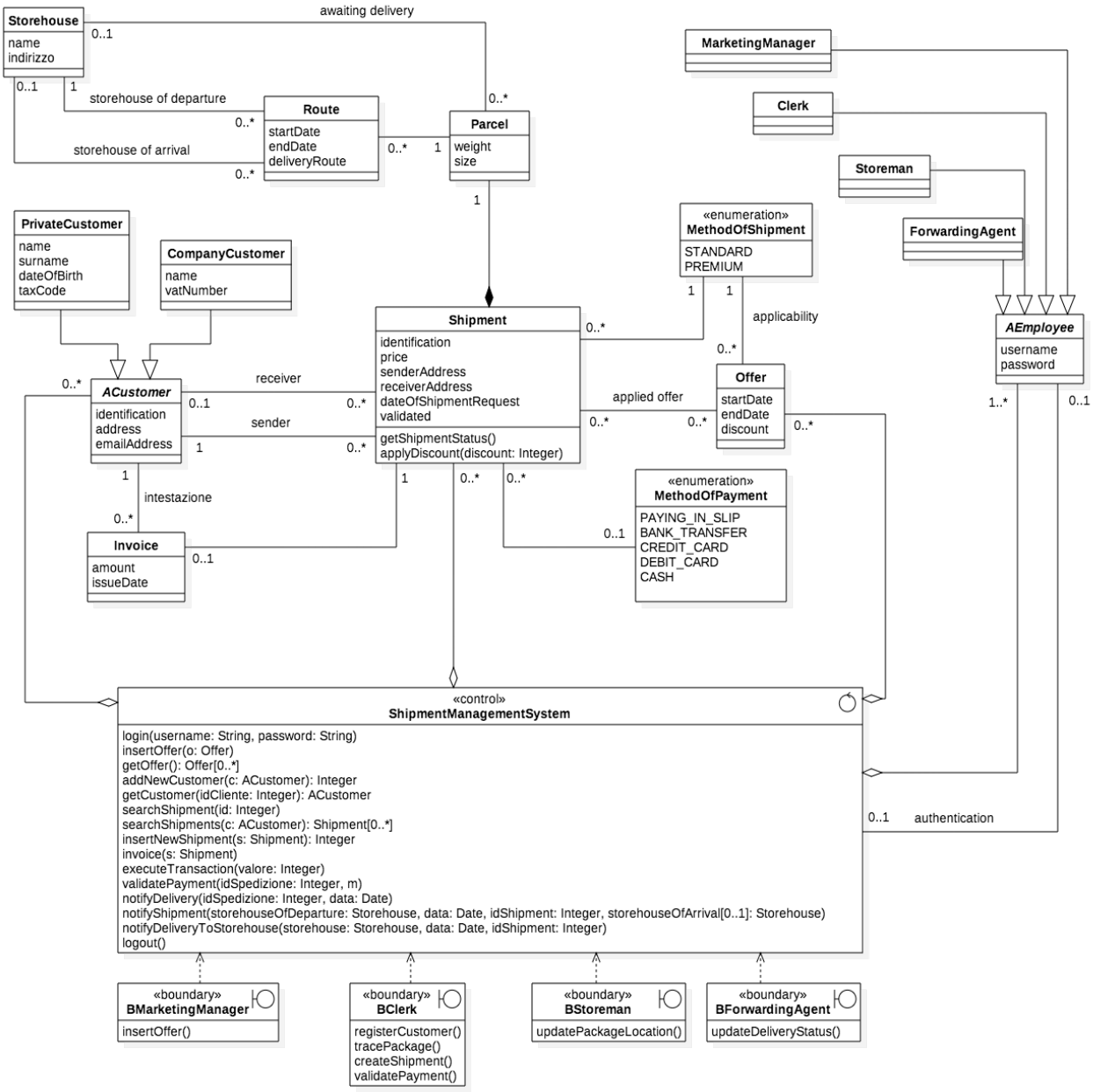

Figura 4 Refined Class Diagram

Raffinare un diagramma di sequenza (Figura 5) è utile per verificare se la modellazione proposta delle classi di analisi è adeguata per catturare i requisiti sui dati del sistema, ovvero se le associazioni e gli attributi identificati sono quelli sufficienti e necessari per permettere lo svolgimento dei casi d'uso (se non sono sufficienti, allora c'è qualcosa da aggiungere alla modellazione; se non sono necessari, probabilmente c'è qualcosa da rimuovere per migliorare la qualità del software).

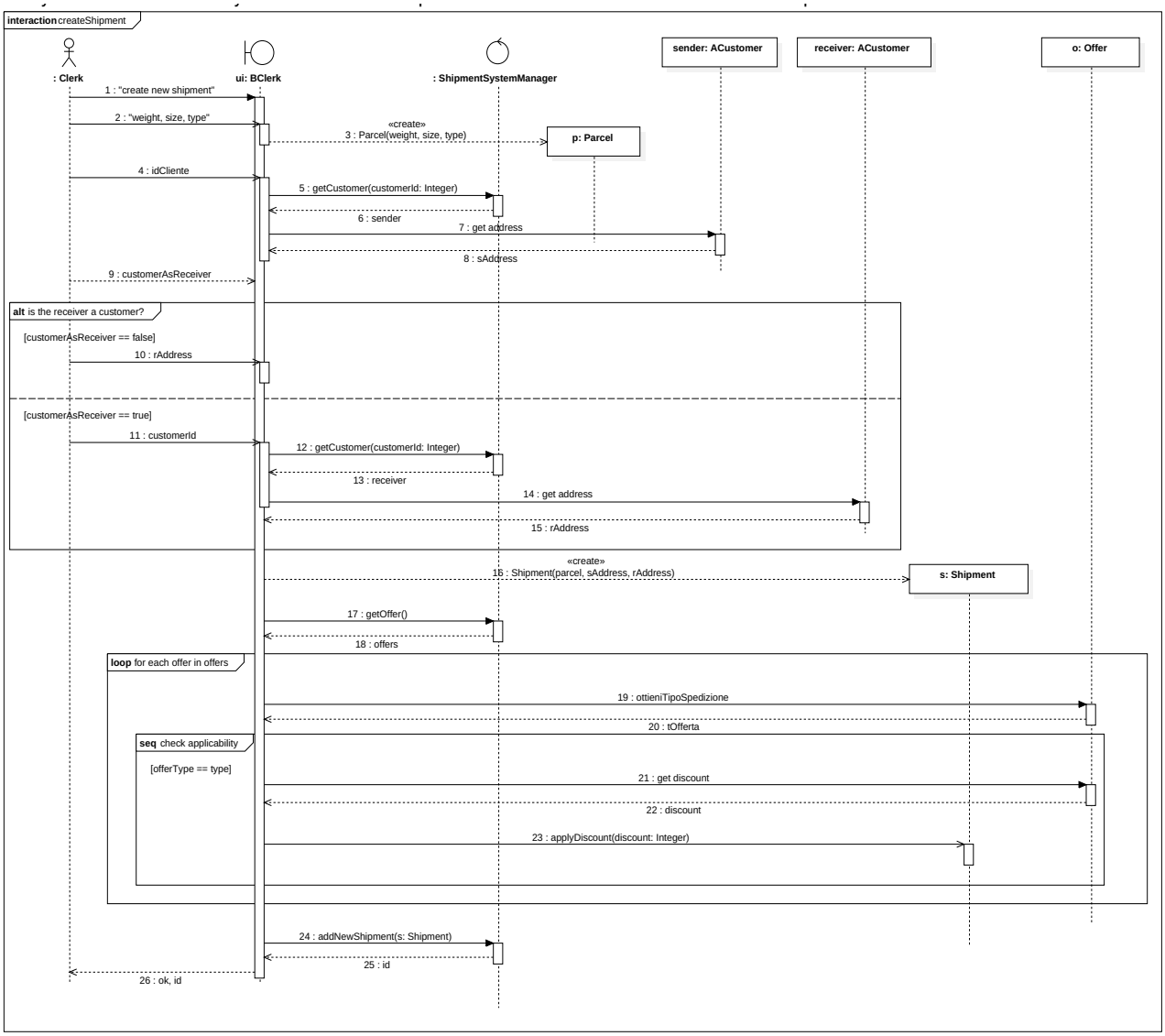

Figura 5 Refined Sequence Diagram del caso d'uso Crea Spedizione

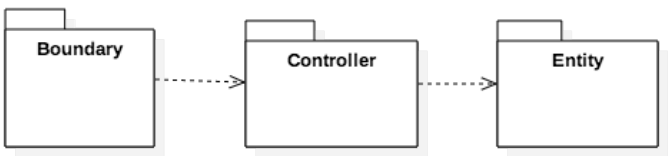

Figura 6 Package Diagram

4 MODELLO DI PROGETTAZIONE

Al termine della fase di analisi abbiamo una specifica del sistema da realizzare; ci concentriamo ora sulla progettazione: mentre l'analisi esplora il dominio del problema, la progettazione studia il dominio della soluzione, e mira ad offrire una soluzione che sia implementabile con specifiche tecnologie, ovvero linguaggi, librerie software, sistemi middleware, etc.

Il nostro progetto è relativo ad una classe di sistemi detta **enterprise applications** (anche "information systems"), ovvero sistemi che soddisfano le esigenze di una impresa, piuttosto che di un singolo individuo. Questi sistemi hanno degli aspetti caratteristici che la progettazione deve tenere in conto, tra cui:

1. Grandi insiemi di dati persistenti (*business entities*) che si assume non possano risiedere tutti in memoria centrale quando l'applicazione è in uso, e dei quali occorre garantire l'accesso nel tempo, perché costituiscono patrimonio aziendale;
2. Accesso concorrente ai dati, poiché questi sistemi sono multiutente;
3. Uso dei dati da parte di persone non tecniche (es. sportellisti, impiegati), che richiedono interfacce semplici, e offerte in più forme di presentazione;
4. Necessità di integrazione con sistemi esterni (es.: sistemi bancari per il pagamento);
5. Variabilità delle procedure (la *business logic*) che svolgono i processi aziendali (casi d'uso).

Per i sistemi enterprise si applicano scelte progettuali consolidate nel mondo dell'ingegneria del software. Per favorire la manutenibilità del software e l'interoperabilità dei dati non seguiamo strettamente il paradigma object-oriented (OO) – che suggerisce di accoppiare le responsabilità di business con le entità del dominio, ma suddividamo le classi in: **Business Object**, **Controller Class** e **Boundary Class**.

- **Le (Entity) Business Objects** modellano gli oggetti del dominio (derivano dai Domain Objects, e sono come Spedizione, Pacco, Tratta, …), sono persistenti e memorizzate in una sorgente dati esterna (es. RDBMS). Generalmente offrono solo metodi accessor e mutator per i loro attributi (ovvero i metodi *get e set*).
- **Le Controller Classes** implementano la business logic, realizzando le transazioni associate ai processi aziendali. Esse modificano le basi dati manipolando le classi entity, ed applicando vincoli di integrità (le business rules).
- **Le Classi Boundary** permettono agli attori per raggiungere le classi controller e di svolgere le transazioni, occupandosi della presentazione dei dati.

In questa architettura le classi entity sembrano avere poche responsabilità, tanto che il modello delle classi di dominio è definito da alcuni come *anemico* ("anemic domain model"). Tuttavia, le classi entity hanno la responsabilità di risolvere problemi di memorizzazione e comunicazione verso sistemi esterni: esse sono generalmente classi serializzabili e grazie ai *framework* di persistenza (ORM o i moderni OGM) diventano strumenti per effettuare trasparentemente operazioni sulla sorgente dati.

Inoltre, il separare le classi controller dalle entity enfatizza l'importanza della business logic, e migliora la manutenibilità, siccome in un sistema enterprise cambiano con maggiore frequenza i processi piuttosto che le classi di dominio. È importante aggiungere che il progetto può includere classi che non appartengono a questi tre gruppi: ad esempio, una classe controller può usare

i servizi di una classe “Parser”, “UUIDGenerator”, “Validator”, “CSVReader”, etc. Queste classi offrono servizi in maniera trasversale ai componenti del sistema, e sono da sviluppare secondo i principi dell’ingegneria del software della progettazione orientata agli oggetti.

Seguendo l’architettura indicata, le interazioni si svolgono prevalentemente in questa sequenza:

1. Le classi *boundary* dialogano con l’utente e accettano richieste per svolgere processi aziendali (es. spedizione di un nuovo pacco).
2. Le richieste sono inviate ad una classe controller che ha la responsabilità di realizzare il caso d’uso. La *boundary* può servirsi delle classi *entity* per scambiare informazioni verso il controller, ma questa scelta può avere impatto sulla manutenibilità e leggibilità. Il progettista può scegliere anche di assegnare alle classi controller parte dello stato dell’interazione, spostandolo dalle classi *boundary*: ciò facilita il collegamento di più elementi di presentation alla stessa interazione.
3. Il controller svolge il caso d’uso, che generalmente è associato a transazioni da eseguire sulla base dati. Il *controller*: (1) preleva dati dalla sorgente dati sotto forma di oggetti entity, (2) li modifica, (3) li salva, tutto all’interno di una transazione che garantisce le proprietà ACID (*Atomicity, Consistency, Isolation, Durability*). La transazione può essere limitata all’invocazione di un metodo del controller (che deve avere tutte le informazioni necessarie per completare la transazione) o ad una successione di metodi, coordinata dallo strato *boundary* o da un elemento *façade*, che fa fluire in più passi le informazioni, modificandole in base all’interazione con l’utente.
4. Le operazioni di aggiornamento della base dati sono eseguite dal controller dialogando direttamente con lo strato di persistenza (classi DAO o framework di persistenza) e manipolando ad arte le classi entity (il controller non usa direttamente istruzioni SQL).

Possiamo vedere le informazioni manipolate dal sistema enterprise come raggruppate nei seguenti insiemi di dati:

1. Nei dati degli oggetti *entity* memorizzati nella sorgente dati. Questi formano il patrimonio aziendale, di cui occorre garantire durabilità e consistenza.
2. Nei dati associati agli oggetti controller. Questi dati consentono di identificare lo stato raggiunto nello svolgimento di un processo business, e permettono di applicare le business rules: ad esempio, “GestoreTransazioni” deve tenere traccia che l’utente ha accettato i termini legali di pagamento prima di inserire i dati della carta di credito, e conosce i tassi aggiornati che deve applicare come “commissioni sul pagamento”.
3. Nei dati posseduti dalle classi boundary. Questi sono associati all’input utente, e alla logica di presentazione e reporting dei dati.

Gli stereotipi[1] RUP aggiungono alle classi UML la semantica della nostra suddivisione logica:

- Classe *«boundary»*: indica una classe che media l’interazione tra il sistema e il suo ambiente;
- Classe *«control»*: identifica una classe che incapsula un comportamento specifico di un caso d’uso;
- Classe *«entity»*: rappresenta una classe che modella informazioni persistenti.

[1] La stereotipazione è uno dei tre meccanismi di estensione del linguaggio UML, in cui gli elementi stereotipati estendono un elemento esistente, aggiungendo una variazione.

Un modello di progettazione è composto dalla descrizione dei sottosistemi, delle classi di progettazione, delle interfacce e dalla realizzazione dei casi d'uso. Nel nostro progetto identifichiamo dapprima l'architettura di alto livello, definendo i componenti di sistema tramite un **diagramma dei componenti**.

Un componente rappresenta una *parte modulare del sistema che incapsula il proprio contenuto e la cui manifestazione è sostituibile all'interno del proprio ambiente*: affinché i componenti possano essere sostituiti (a design time o a runtime), il loro comportamento è definito in termini di interfacce offerte e richieste. L'architettura interna dei componenti è definita poi in termini di uno o più classificatori che realizzano il comportamento richiesto[2]. Successivamente, nella fase di implementazione, ad ogni componente corrisponderanno uno o più manufatti (artifact). I file eseguibili (.exe) sono un esempio di manufatto che può realizzare un componente: essi sono componenti sostituibili a runtime, che richiedono le interfacce di un sistema operativo ed offrono servizi verso gli altri processi di sistema tramite i meccanismi di inter-process communication (IPC). Lo stereotipo «subsystem» applicato ad un componente lo identifica come un'unità di scomposizione gerarchica, ed è utile per scomporre sistemi di grandi dimensioni.

Individuiamo tre *componenti* (Figura 7): il primo realizza lo strato di boundary (GUI), il secondo incapsula lo strato dei controller, il terzo è un RDBMS che usiamo per la persistenza. I primi due interagiscono tramite una sola interfaccia, che chiamiamo IShipmentAgencyEIS. Gli ultimi due comunicano grazie allo standard JDBC, che ci permette di usare un sistema di base dati già disponibile sul mercato (Component Off-The-Shelf, COTS). L'architettura proposta permette la sostituzione di questi componenti a tempo di progettazione (*design-time*).

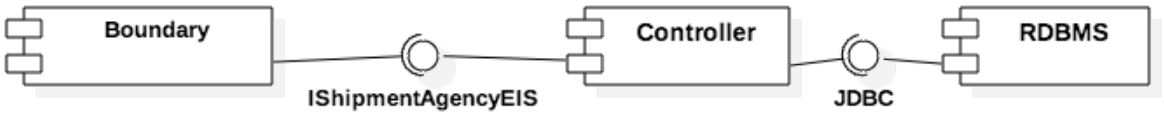

Figura 7 Component Diagram iniziale

Il **diagramma delle classi di progetto** (*design class diagram* – design-CD) è ottenuto raffinando il diagramma di analisi, aggiungendo i dettagli tralasciati nel precedentemente, ed offrendo una soluzione al problema. Il dettaglio offerto dai design-CD deve essere tale da specificare l'implementazione nella fase successiva di sviluppo. In particolare:

- Le classi devono specificare l'insieme completo dei loro attributi (dettagliando nome, tipo, visibilità, etc.) e delle operazioni (dettagliando nome, nome e tipi dei parametri, tipo del valore restituito, visibilità, etc.);
- Tutte le associazioni vanno raffinate e ben definite (tipo, cardinalità, direzione e ruolo).

Rispettando i principi dell'ingegneria del software, la progettazione deve offrire:

[2] In UML la classe è un tipo di classificatore: il classificatore è un concetto più astratto di classe che include ulteriori elementi tra cui i componenti, le interfacce e i casi d'uso.

- **classi *complete*** (che offrono *almeno* ciò che i loro utilizzatori attendono), ***sufficienti*** (che offrono *solo* ciò che i loro utilizzatori attendono), ed ***essenziali*** (i servizi offerti devono essere primitivi, semplici, atomici e distinti);
- **moduli** (classi, componenti, package, etc.) con ***massima coesione*** **e** ***minimo accoppiamento*** (poche *relazioni di dipendenza* tra gli elementi);
- **ulteriori entità** (classi, oggetti, componenti, etc.) che sono introdotte per rispondere ad esigenze e scelte progettuali (es.: un componente che offre servizi di caching per migliorare le performance).

Le classi di progettazione sono raggruppate logicamente in *package*, come mostrato in Figura 8. Identifichiamo un package per contenere gli elementi che appartengono allo strato boundary, un package per le classi controller, ed un terzo per le classi entity. Infine, introduciamo un package per contenere le implementazioni delle classi controller, ed uno per le classi DAO.

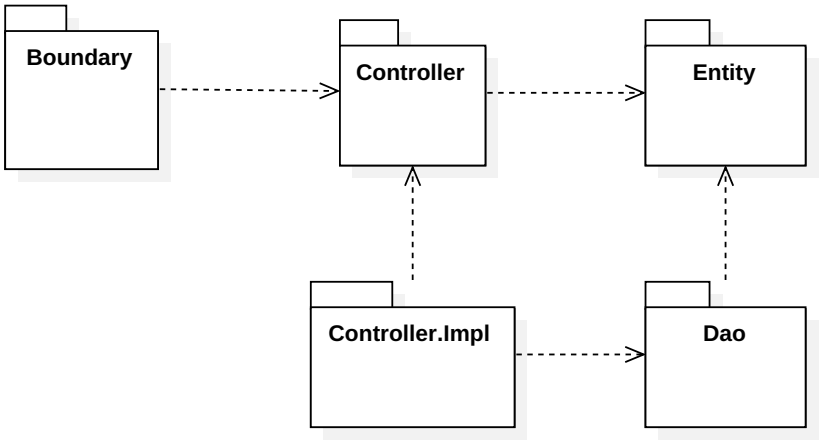

Figura 8 Design Package Diagram

Per ogni package possiamo sviluppare un design-CD che comunica visivamente il contenuto. La Figura 9 mostra un diagramma delle classi di progetto con i relativi *package*, per indicare la logica progettuale. La Figura 10 mostra un Design Class Diagram parziale dei package Controller e ControllerImpl.

Per migliorare la qualità interna del software la progettazione deve preferire l'uso di interfacce a classi, in modo da avere dipendenze di servizi, piuttosto che di implementazione[3]. Ad esempio, abbiamo sviluppato i controller basandoci su interfacce specifiche (CustomerManager, ShipmentManager, AuthenticationManager) che sono state raggruppate in un façade. L'implementazione concreta dei controller è realizzata nel package ControllerImpl: il façade è composto a runtime grazie ad una classe Builder offerta da PackageImpl. In questo modo lo strato boundary ha una sola dipendenza nel package controller, ed i controller sono facilmente sostituibili: possiamo isolarli e sostituirli durante il testing, migliorando l'evolvibilità, la manutenibilità e la testabilità.

[3] In particolare, il principio della segregazione delle interfacce (*interface segregation principle*, ISP) suggerisce di scomporre le interfacce in molte e specifiche, riducendo le dipendenze di ciascun *client* da metodi in cui può non avere interesse.

Ad esempio, in una architettura a *microservizi,* le classi controller potrebbero essere trasformate in web service, e contattati dal *façade* tramite protocolli standard di *service discovery* e comunicazione. Tale soluzione offrirebbe la sostituibilità dei controller a runtime, limitando in maniera molto fine le parti del software da modificare in ogni *release* (*design for change*).

La Figura 11 mostra un **diagramma di sequenza** di progettazione relativo al caso d'uso CreaSpedizione.

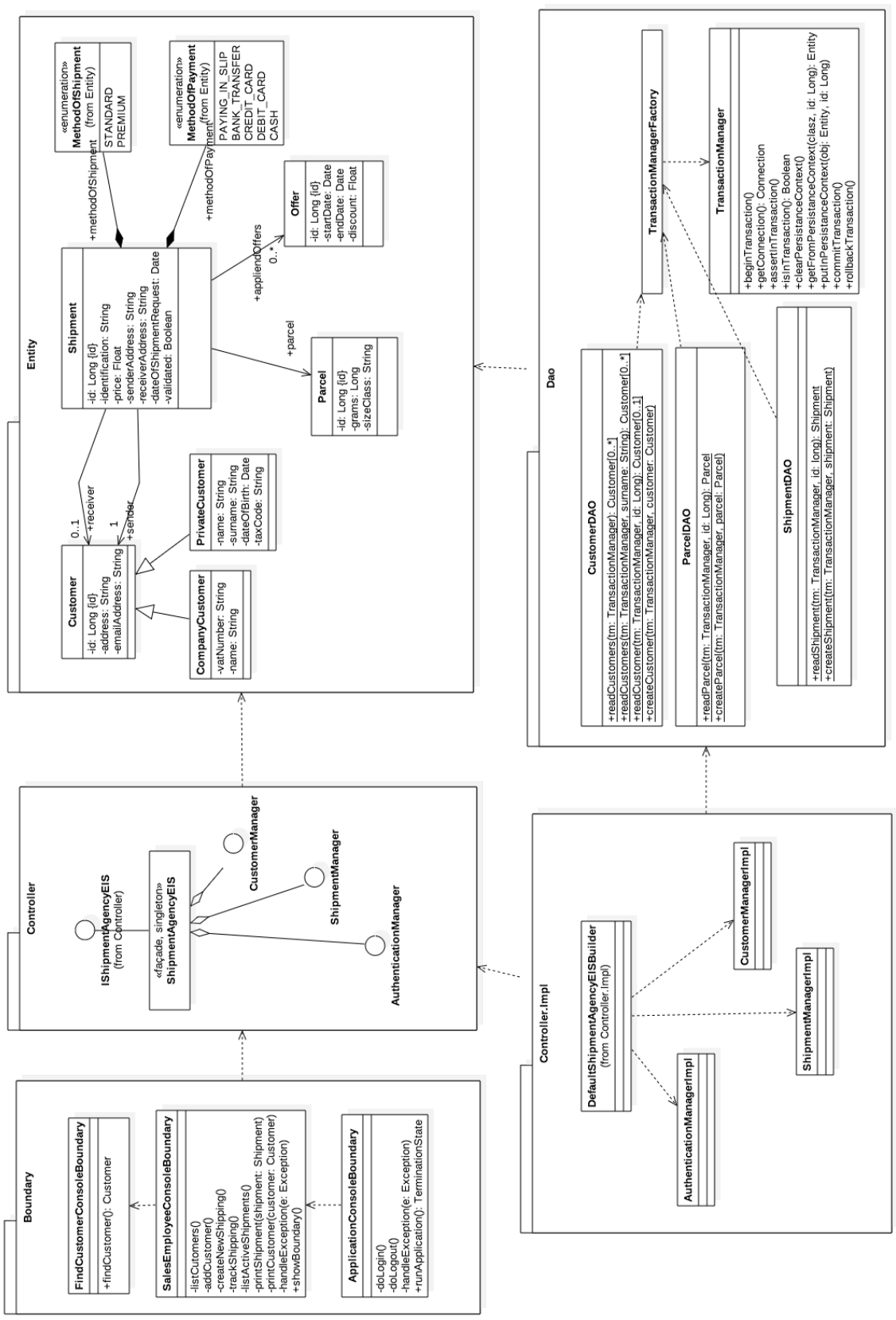

Figura 9 Design Class Diagram completo con package

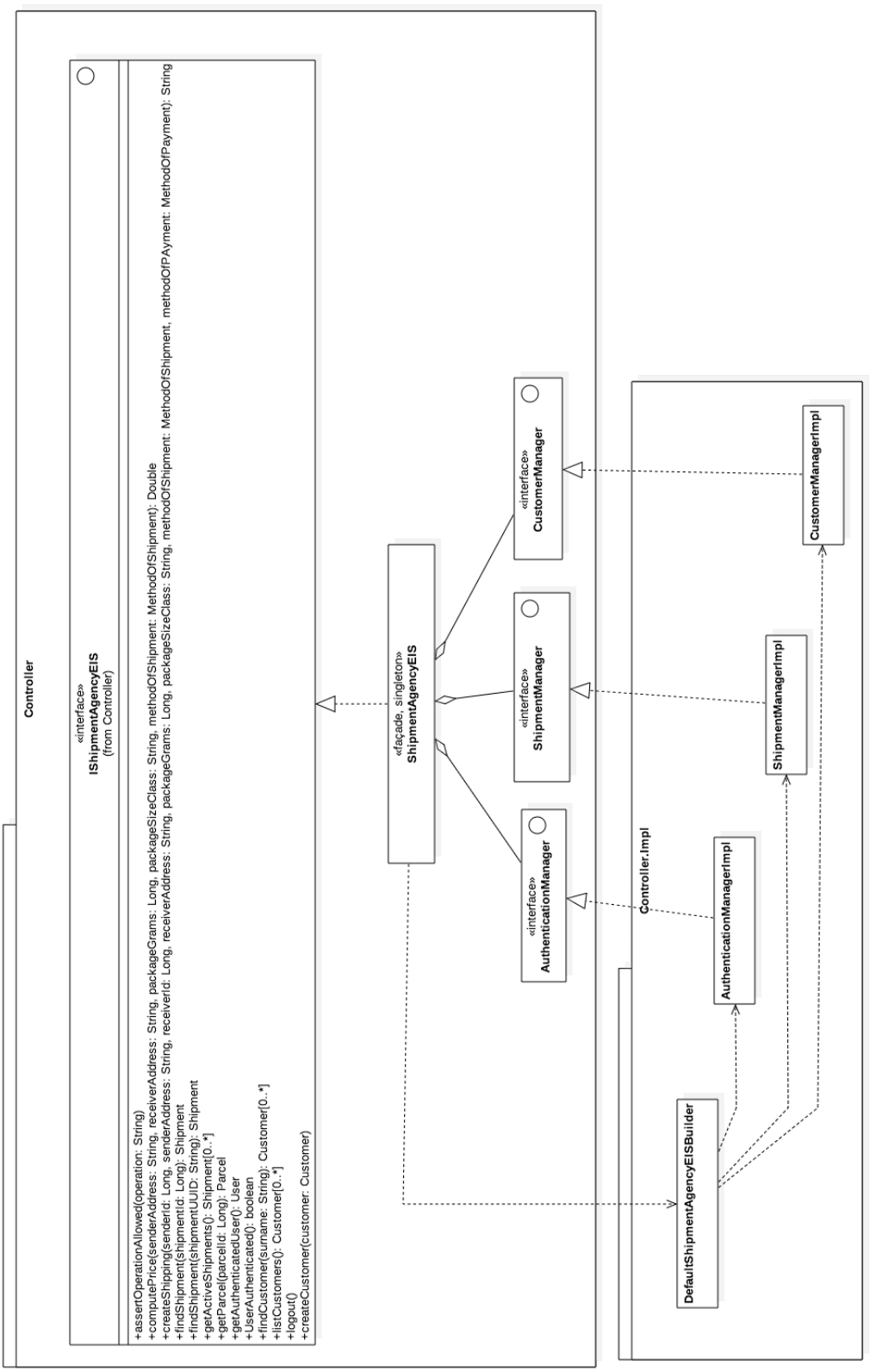

Figura 10 Design Class Diagram parziale dei package Controller e ControllerImpl

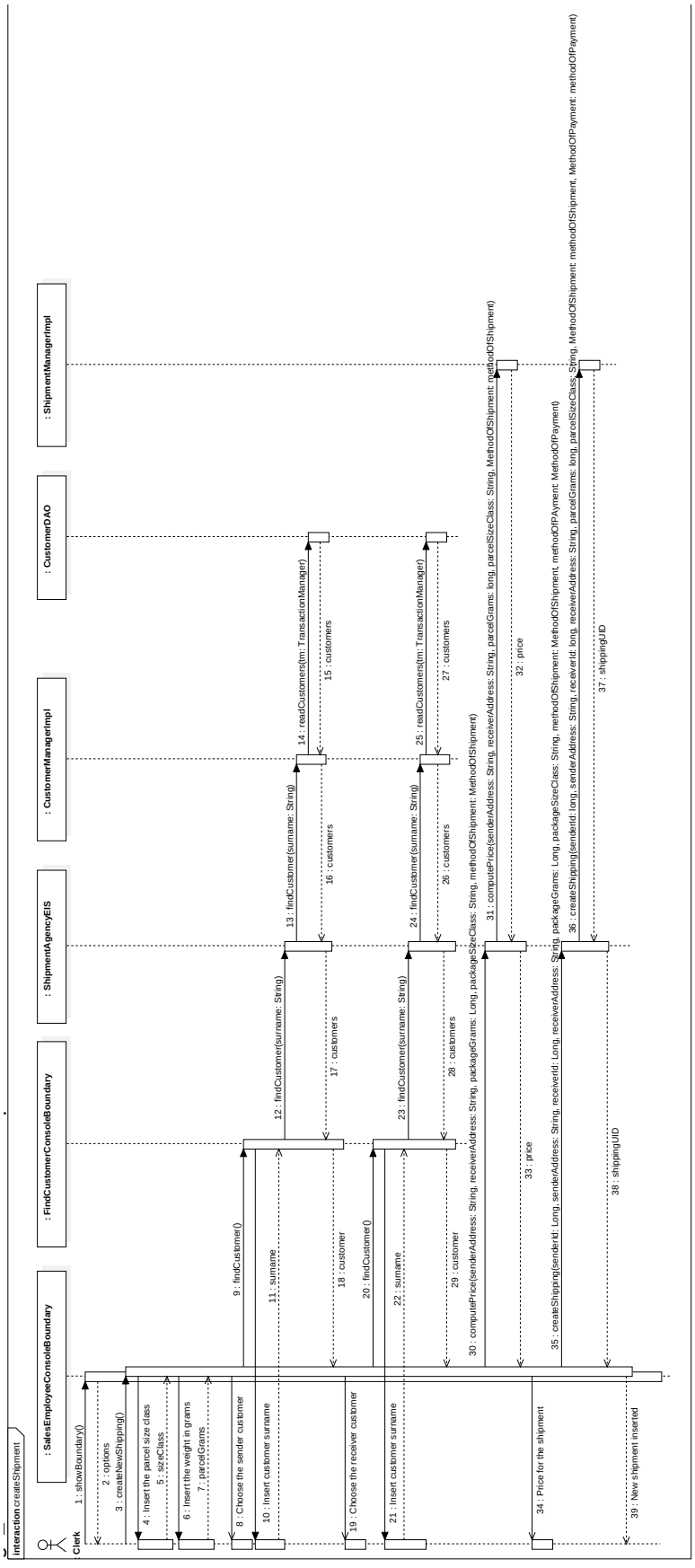

Figura 11 Design Sequence Diagram della realizzazione del caso d'uso CreaSpedizione

In progettazione si aggiungono inoltre diagrammi per modellare il comportamento dinamico degli elementi del sistema, come **diagrammi degli stati** (*statechart*). Le macchine a stato ci permetto di modellare il ciclo di vita di un'unica entità reattiva: concetto fondamentale è quello di **stato**, che è una condizione interna di funzionamento dell'oggetto che racchiude in sé tutta la sua storia. Lo stato è quindi un'astrazione di tutta la storia dell'entità. Esistono due tipi di macchine a stati in UML 2: le macchine a stato *del comportamento* e le macchine a stato *del protocollo*.

La Figura 12 illustra il diagramma degli stati degli oggetti della classe Spedizione (in particolare, l'esempio modella una macchina a stati del protocollo).

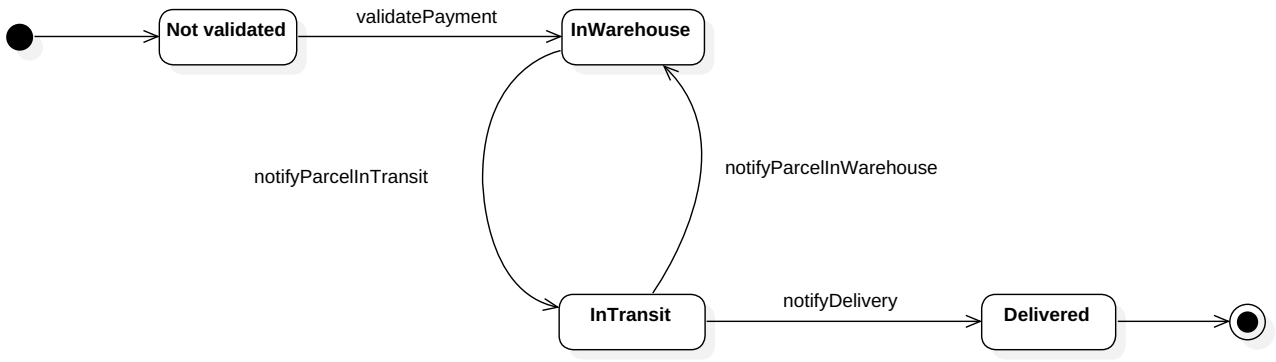

Figura 12 Statechart della classe Spedizione

5 MODELLO DI IMPLEMENTAZIONE

L'implementazione riguarda la trasformazione della soluzione (ovvero, del modello di progettazione) in elementi fisici di informazione (file sorgente, file eseguibili, file di configurazione, etc.) che sono realizzati o sfruttati nel processo di sviluppo software, ovvero – secondo la definizione UML – riguarda la trasformazione della soluzione in manufatti (*artifact*).

Il modello di implementazione:

- raffina il modello di progettazione aggiungendo dettagli specifici di una particolare soluzione implementativa (es.: può adattare la soluzione per risolvere la mancanza dell'ereditarietà multipla in Java);
- comunica quali manufatti devono essere sviluppati durante la fase di implementazione (es. quali tabelle relazionali occorre produrre, file .jar, file .exe, etc.);
- indica come i manufatti sono messi in esercizio nel sistema (*deployment*).

Un modello di implementazione aggiunge diagrammi dei componenti, per modellare quali manufatti manifestano i componenti (Figura 13); e diagrammi di *deployment* (Figura 14), per specificare le risorse computazionali (i *nodi* o *node*) su cui devono essere allocati i manufatti, e rappresentare la forma con cui il software verrà messo in esercizio.

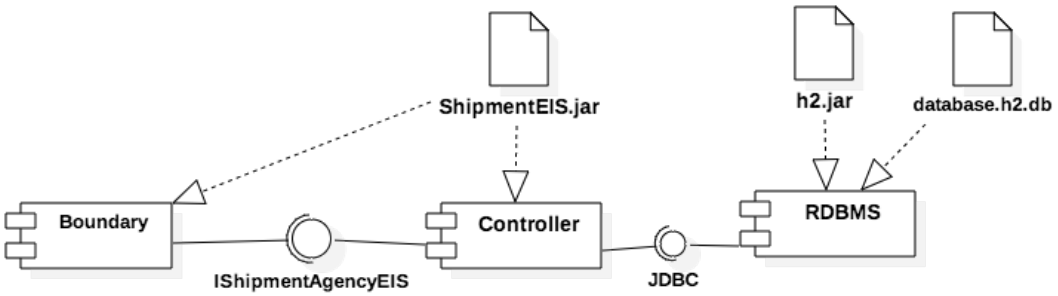

Figura 13 Implementation Component Diagram

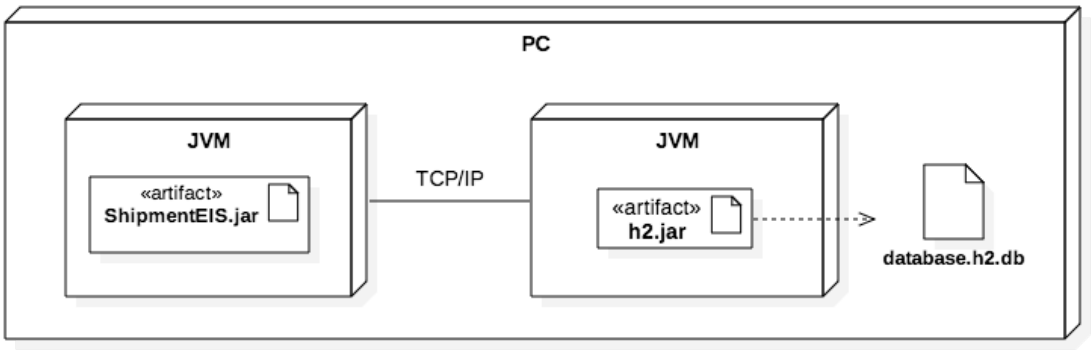

Figura 14 Deployment Diagram