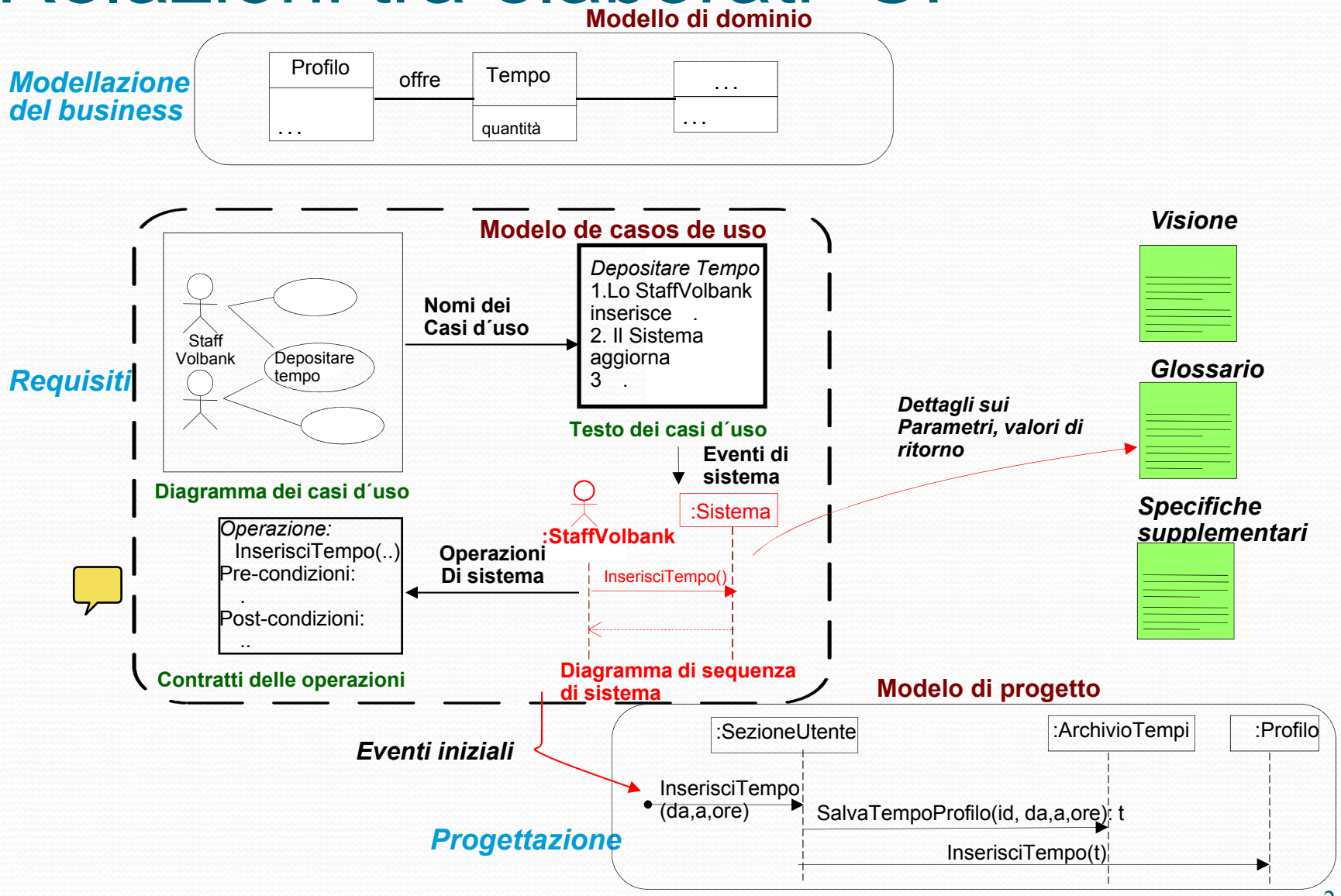
Sperimentazioni di Ingegneria del Software

- Elaborazione
- Disciplina: Requisiti
  - Diagrammi di sequenza di sistema
  - Contratti delle operazioni

Grazie a Simona Bernardi (Universita` di Saragoza, Spagna), che ha messo a disposizione il suo materiale

Relazioni tra elaborati UP



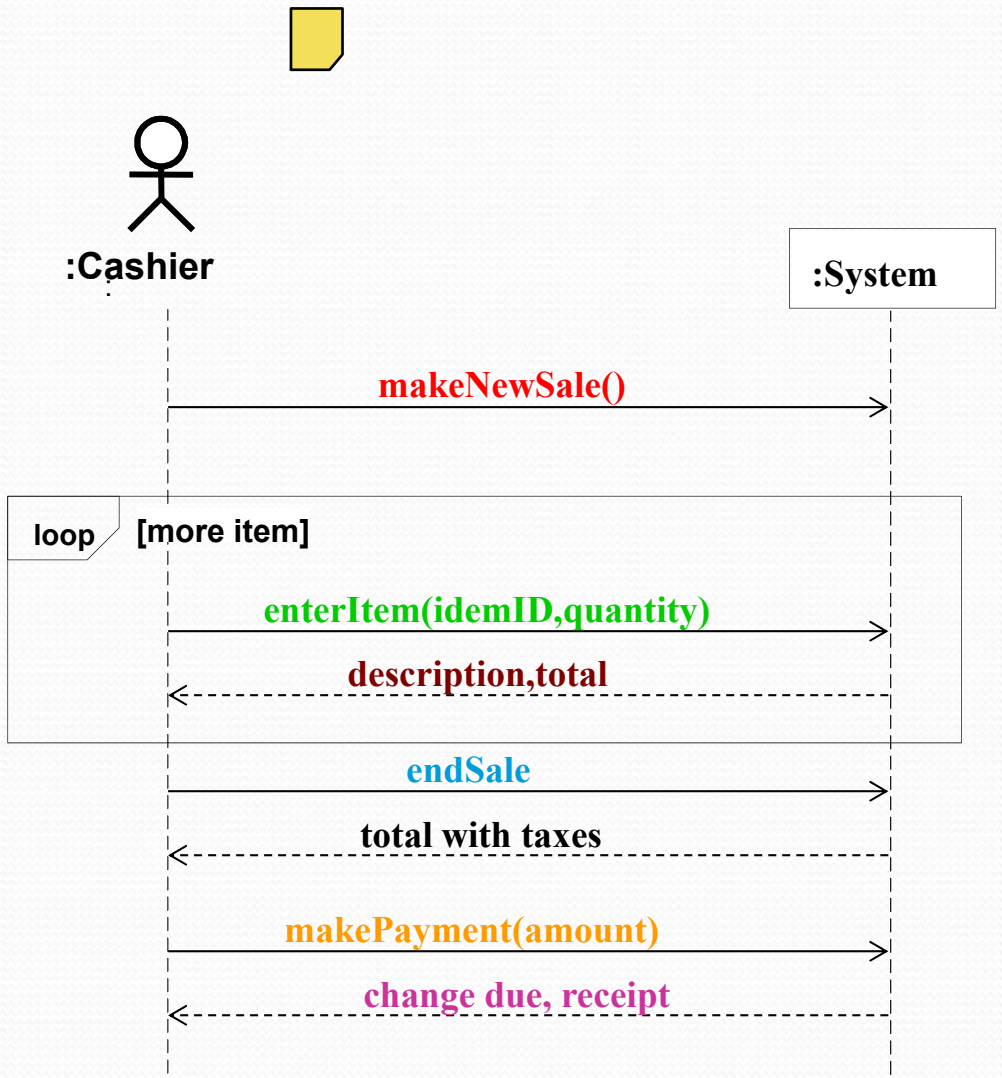
Diagrammi di sequenza di sistema (DSS)

- E´un elaborato che rappresenta gli eventi di input ed output del sistema
- Disciplina: Requisiti
  - Non è menzionato esplicitamente in UP
  - Si usa un diagramma di sequenza UML
- Attori
  - Il Sistema è modellato come una “scatola nera”
- Modellare un DSS per ogni caso d´uso
  - Per lo scenario principale
  - Per gli scenari alternativi complessi

DSS Volbank

Elabora Vendita
(Scenario principale):

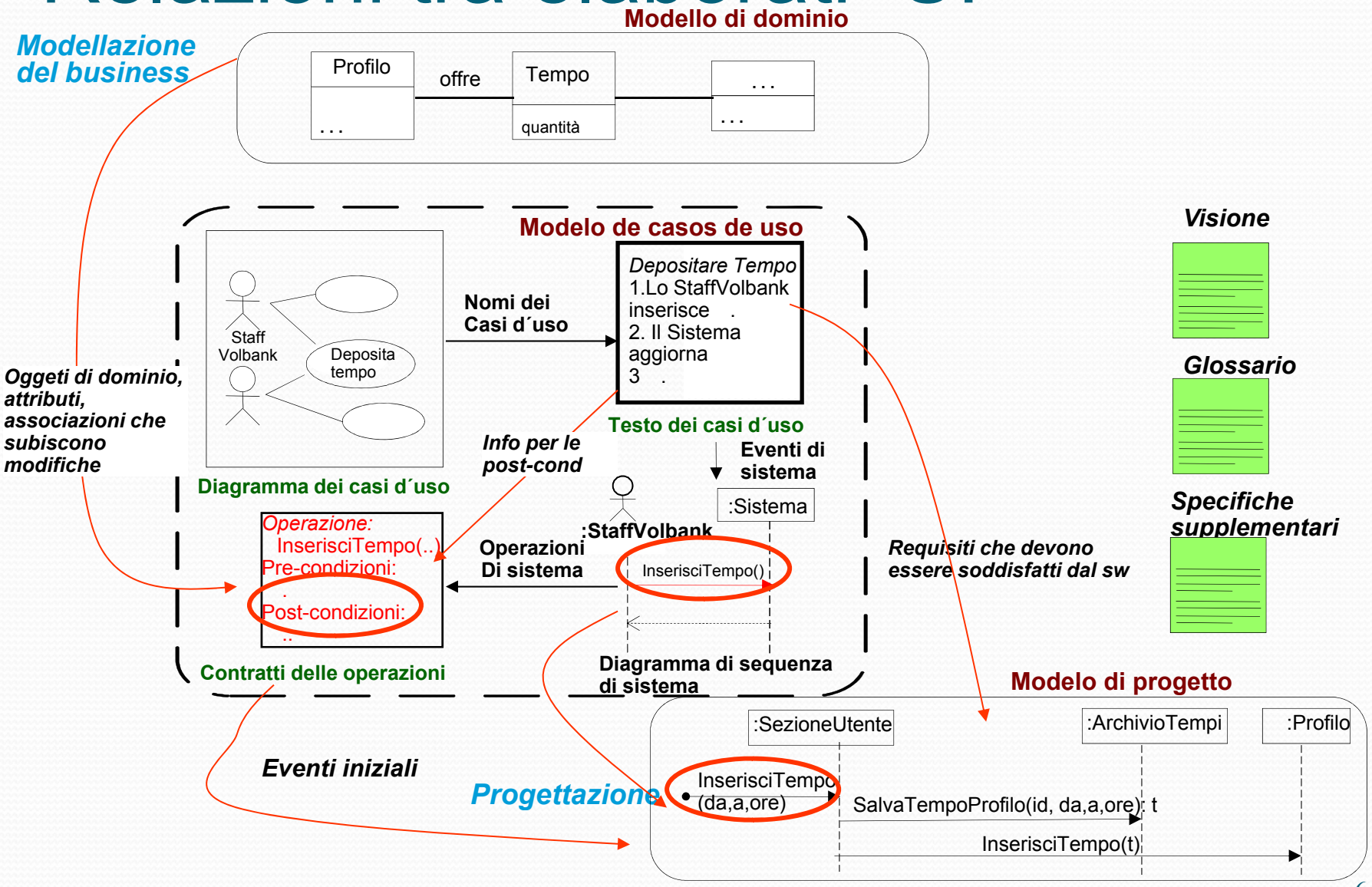
1. Il Cliente arriva alla cassa POS
2. Il Cassiere inizia una nuova vendita
3. Il Cassiere inserisce il codice identificativo dell´articolo e la qtà
4. Il Sistema registra la riga di vendita per l´articolo e mostra la descrizione dell´articolo, il suo prezzo,
5. Il Cassiere ripete i passi 2-3 fino a che non indica che ha terminato
6. Il Sistema mostra il totale con le imposte calcolate
7. Il Cassiere riferisce il totale al Cliente e richiede il pagamento
8. Il Cliente paga ed il Sistema gestisce il pagamento
9. Il Sistema registra la vendita completata
10. Il Sistema mostra il cambio e genera la ricevuta
11. Il Cliente va via con la ricevuta e gli articoli acquistati



Sperimentazioni di Ingegneria del Software

- Elaborazione
- Disciplina: Requisiti
  - Diagrammi di sequenza di sistema
  - Contratti delle operazioni

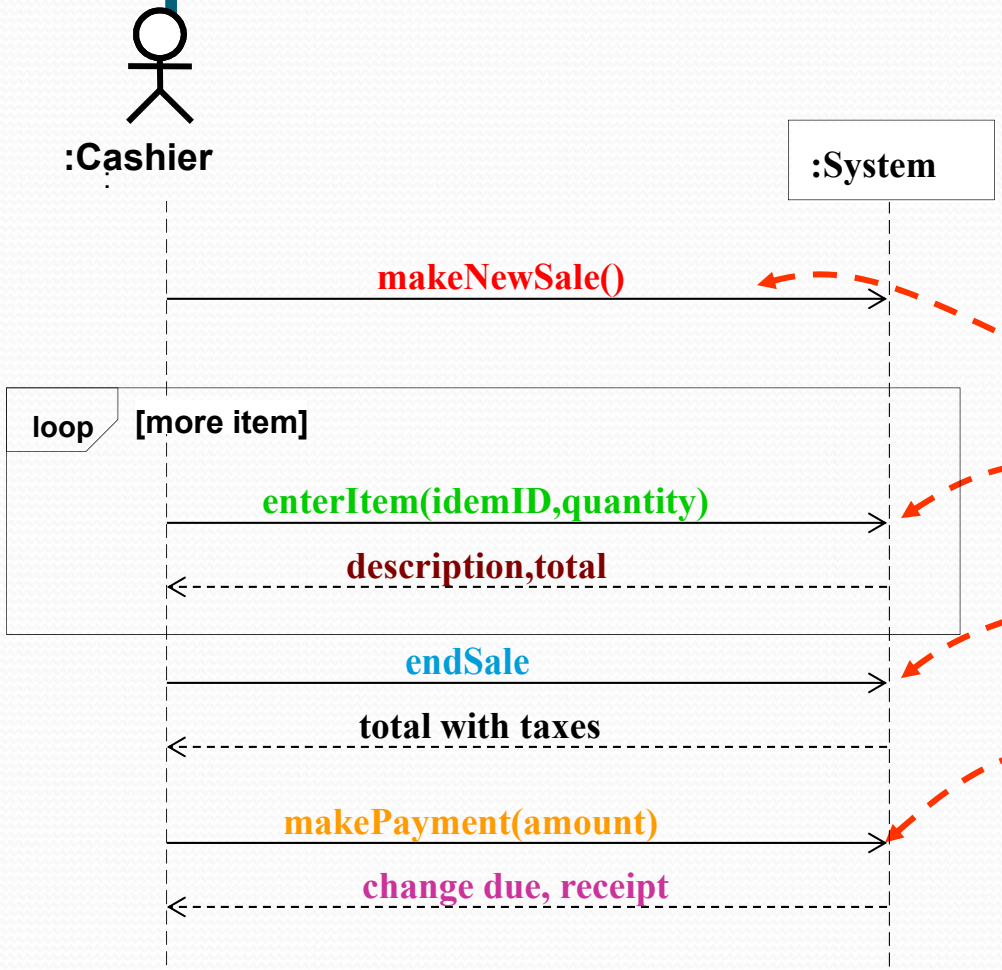
Relazioni tra elaborati UP



Contratti delle operazioni

- Descrizioni testuali, con pre- e post- condizioni, sui cambi di stato degli oggetti del modello di dominio a seguito di una operazione di sistema
- Operazione di sistema: operazione offerta dal sistema (visto come scatola nera) nella sua interfaccia pubblica
- Disciplina: Requisiti
  - Non menzionati esplicitamente in UP

Operazioni di sistema



Questi eventi di sistema in input sono chiamate alle operazioni di sistema

L´ evento di sistema makeNewSale è una chiamata all´operazione di sistema makeNewSale()

Template per i contratti

Operazione	Nome e parametri dell´operazione
Riferimenti	Casi d´uso in cui si può verificare (chiamare) questa operazione
Pre-condizioni	Ipotesi non banali sullo stato del sistema o degli oggetti nel modello di dominio prima dell´esecuzione dell´operazione.
Post-condizioni	**Lo stato degli oggetti nel modello di dominio dopo il completamento dell´operazione.**

Come creare e scrivere i contratti - I

- Identificare le operazioni di sistema nel DSS
- Creare un contratto per ogni operazione di sistema complessa
- Scrivere le post-condizioni usando le seguenti categorie
  - Creazione-cancellazione di istanze di classi
  - Modifica attributi
  - Creazione-cancellazione istanze di associazioni (link)

Esempio POS

Contratto C01:

Operazione: makeNewSale()

Riferimenti: Caso d´uso ElaboraVendita

Pre-condizioni: Nessuna

Post-condizioni:

- **È stata creata una istanza s di Sale (creazione istanza)**
- **s è stata associata con Register (associazione formata)**
- **Gli attributi di s sono stati inizializzati (modifica attributi)**

Contratto C02:

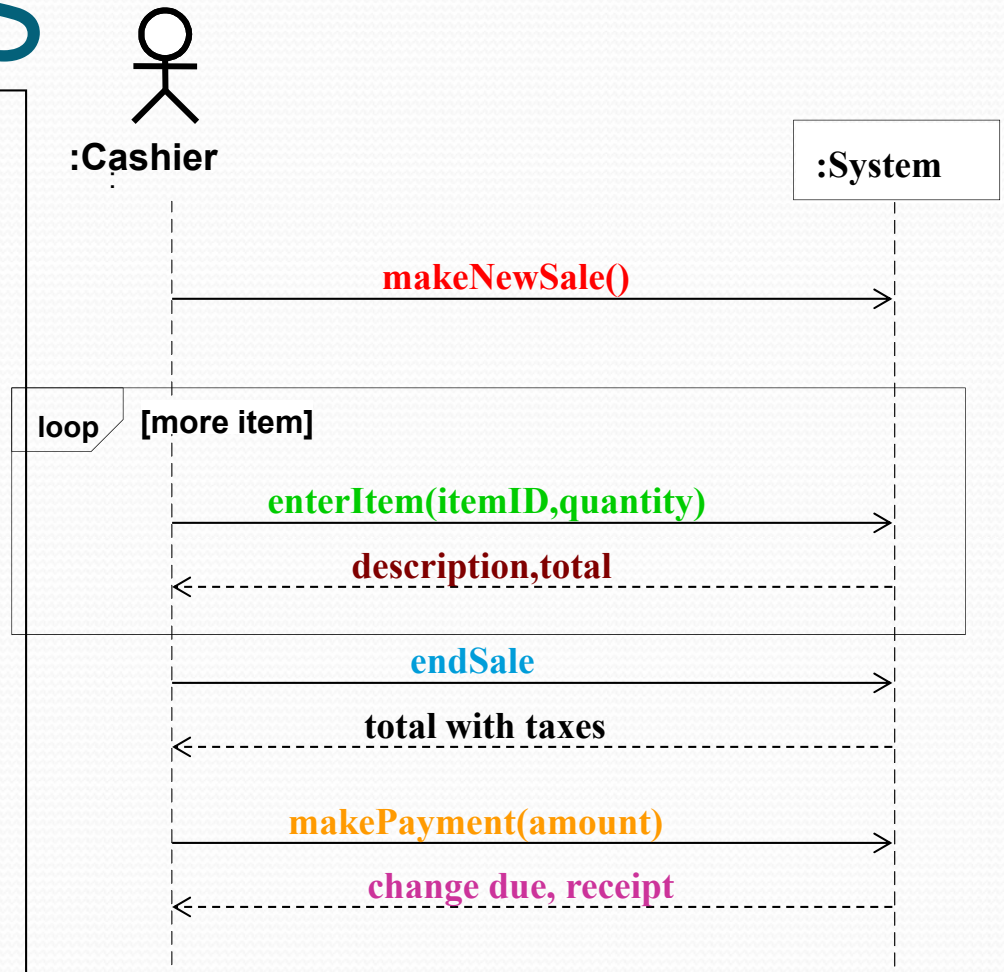
Operazione: enterItem(itemID,quantity)

Riferimenti: Caso d´uso ElaboraVendita

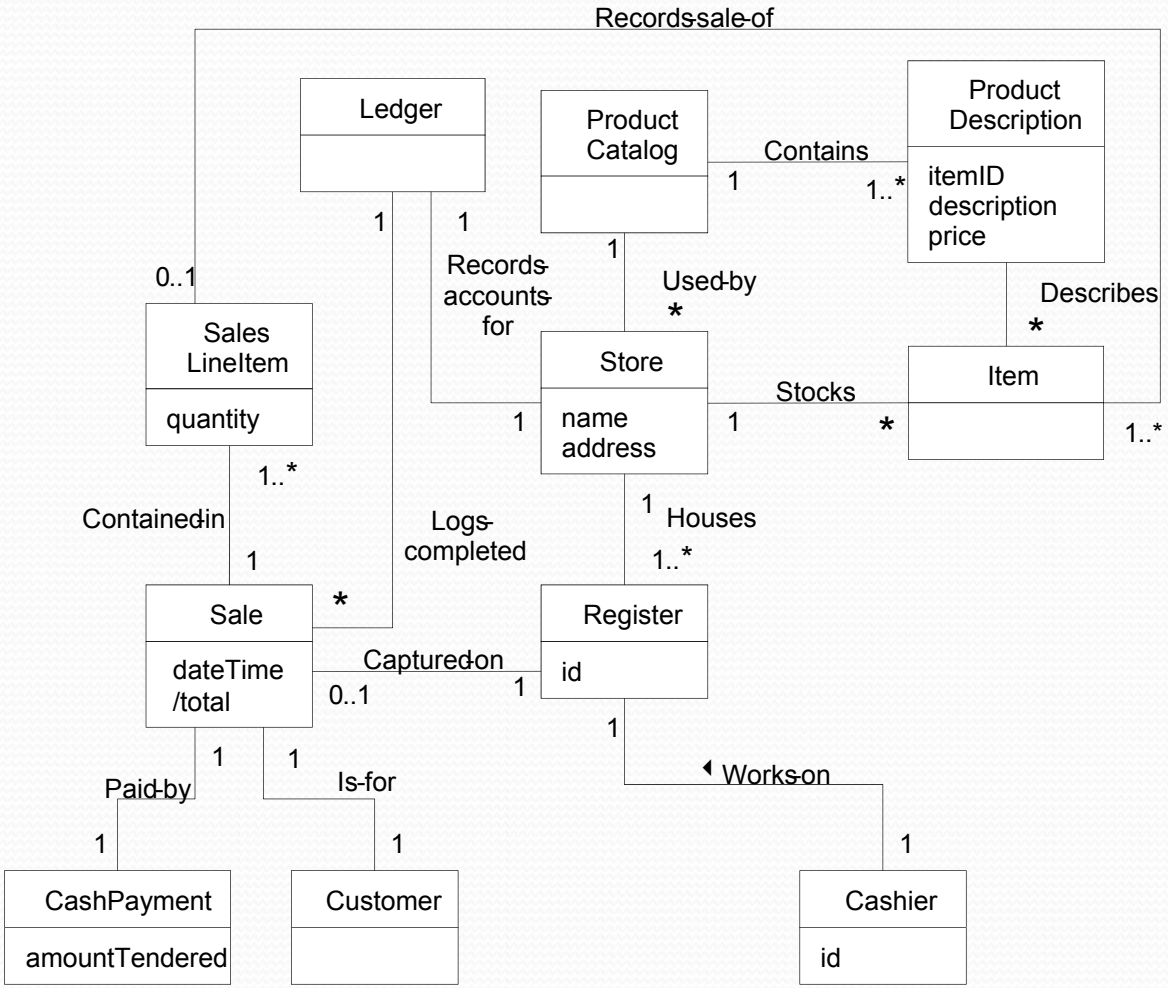
Pre-condizioni: è in corso una vendita

Post-condizioni:

- **È stata creata una istanza sli di SalesLineItem**
- **sli è stata associata con Sale corrente s**
- **sli.quantity è diventata quantity**
- **sli è stata associata a una ProductDescription in base alla corrispondenza con itemID**



Modello di dominio POS (parziale)



Come creare e scrivere contratti - II

- Bisogna scrivere un contratto per ogni evento di sistema trovato nel DSS ?
  - Non è necessario: consideriamo quelli più complessi
- Se si scoprono nuove classi, attributi, si possono aggiungere nel modello di dominio?
  - Ovvio! UP è incrementale
- Le post-condizioni devono essere in ogni momento il più complete possibili ?
  - Non è necessario: UP iterativo ed incrementale

Modifiche nel modello di dominio

Operazione	endSale()
Riferimenti	ElaboraVendita
Pre-condizioni	È in corso una vendita
Post-condizioni	**- Sale.isComplete è diventato true**

Aggiunto nuovo attributo

Sale
dateTime \total **isComplete:Bool**

Contratti nella progettazione: DBC

- Design by Contract (DBC) è una metodologia introdotta da Bertrand Meyer (anni ’80) ed integrata in Eiffel
- Consiste nel considerare un’interfaccia (insieme di metodi esposti da una classe) come un contratto fra chi la invoca (client) e chi la implementa (classe)
- Termini del contratto:
  - Pre-condizioni: requisiti minimi che devono essere soddisfatti dal cliente affinchè il metodo invocato possa andare a buon fine
  - Post-condizioni: requisiti per cui l’esecuzione del metodo si possa ritenere corretta (requisiti che il metodo deve soddisfare)
  - Invarianti: condizioni che devono essere sempre valide

Operazioni/metodi in UML

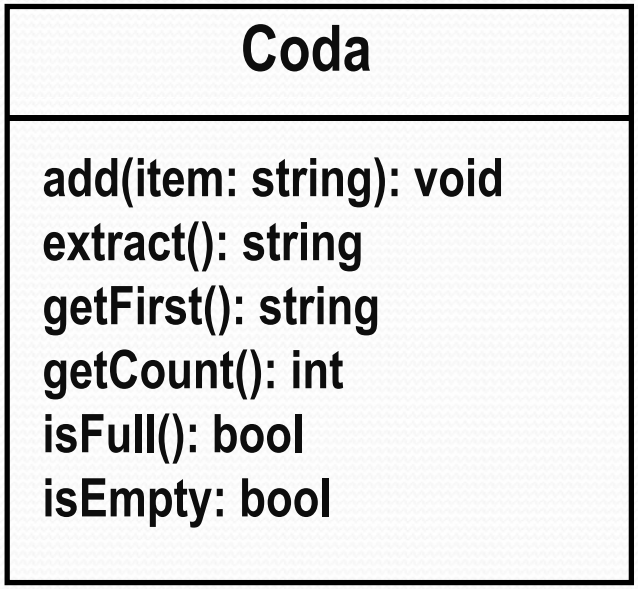
- Operazione
  - trasformazione o interrogazione che un oggetto può essere chiamato ad eseguire
- Un´operazione è una astrazione non una implementazione
- Metodo
  - implementazione di un´operazione
- Un´operazione ha una firma (signature) con nome e parametri
- In UML possono essere associati vincoli ad una operazione (pre e post condizioni)

DBC: Termini del contratto

- Le pre- e post- condizioni riguardano un' operazione:
  - Le pre-condizioni sono requisiti per il client (chiamante)
  - Le post-condizioni sono requisiti per l'operazione
- Gli invarianti costituiscono un vincolo per l' intera classe
  - Devono valere prima dell'invocazione dell'operazione/metodo ed alla fine dell' esecuzione
  - Possono non essere "temporaneamente" validi durante l'esecuzione dell'operazione/metodo

Esempio: coda di messaggi

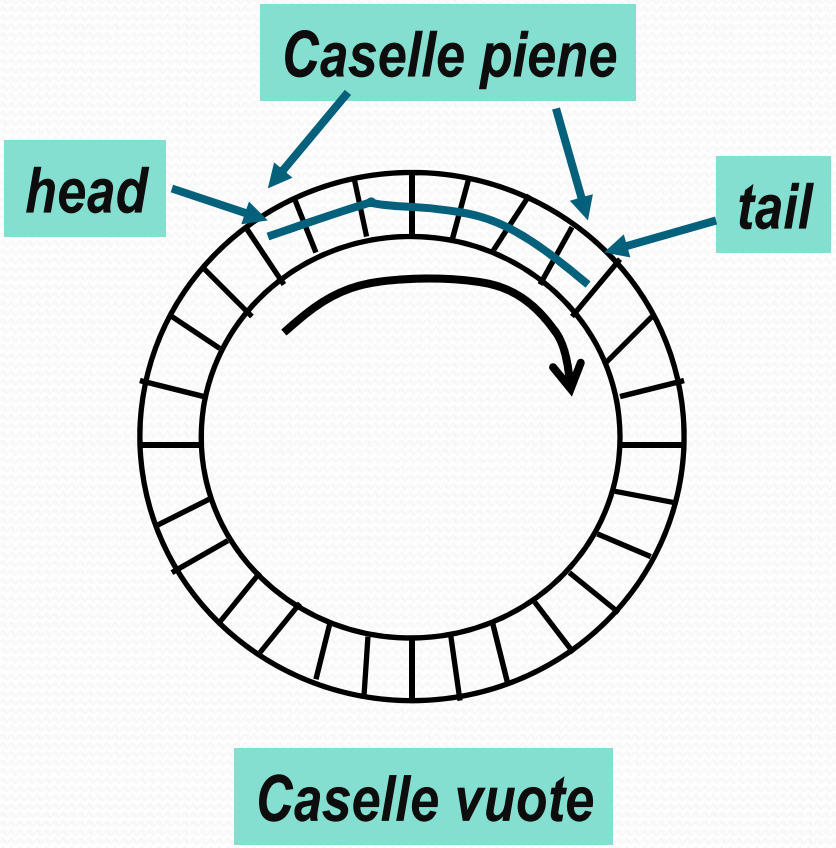
- Coda di messaggi (stringhe)
- Comportamento FIFO
- Operazioni:
  - Accodamento di un elemento
  - Estrazione del primo elemento in testa
  - Lettura del primo elemento in testa
  - Lettura dimensione coda
  - Test coda piena/vuota



Scelte di implementazione

- Buffer circolare che fa uso di un array con due indici
  - ***head***: punta all'elemento in testa
  - ***tail***: punta all'elemento in coda
- La dimensione del buffer (n.max elementi) viene specificata nel costruttore
- Quanto di aggiunge un elemento si incrementa ***tail***, quando si estrae un elemento si incrementa ***head***
- Indici gestiti in modo "circolare", quando si arriva alla fine dell'array si riparte da zero

Buffer circolare



BufferCircolare
-items[]: string -head: int -tail:int -count:int
+add(item: string): void +extract(): string +getFirst(): string +getCount(): int +isFull(): bool +isEmpty: bool

Classe BufferCircolare

```
public class BufferCircolare{
private string[] items;
private int head, tail, count;
public BufferCircolare(int capacity)
{items = new string[capacity];
count=0; head=0; tail=0;}
public void add(string item)
{items[tail] = item;
tail = (tail + 1) % items.length;
count++;
}
```

```
public string extract()
{string s = items[head];
head = (head + 1) % items.length;
count--;
return s;}
public string getFirst()
{return items[head];}
public int getCount()
{return count;}
public boolean isFull()
{return count >= items.length;}
public boolean is Empty()
{return count==0;}
```

Termini del contratto

- L'interfaccia Coda è un contratto tra chi implementa la classe e chi la usa
- Clausole implicite
  - Non si può estrarre un elemento da una coda vuota
  - Non si può inserire un elemento in una coda piena
  - Dopo aver inserito un elemento in una coda vuota la coda non è più vuota
  - Dopo aver estratto un elemento da una coda piena la coda non è più piena
  - I valori di testa e coda devono essere sempre indici validi dell' array (tra 0 e length-1)

Pre-condizioni

- Le due clausole
  - Non si può estrarre un elemento da una coda vuota
  - Non si può inserire un elemento in una coda piena
- Devono essere rispettate da chi invoca le operazioni (client)
- Sono pre-condizioni
- Se vengono violate il metodo deve bloccare l'esecuzione e sollevare l'eccezione

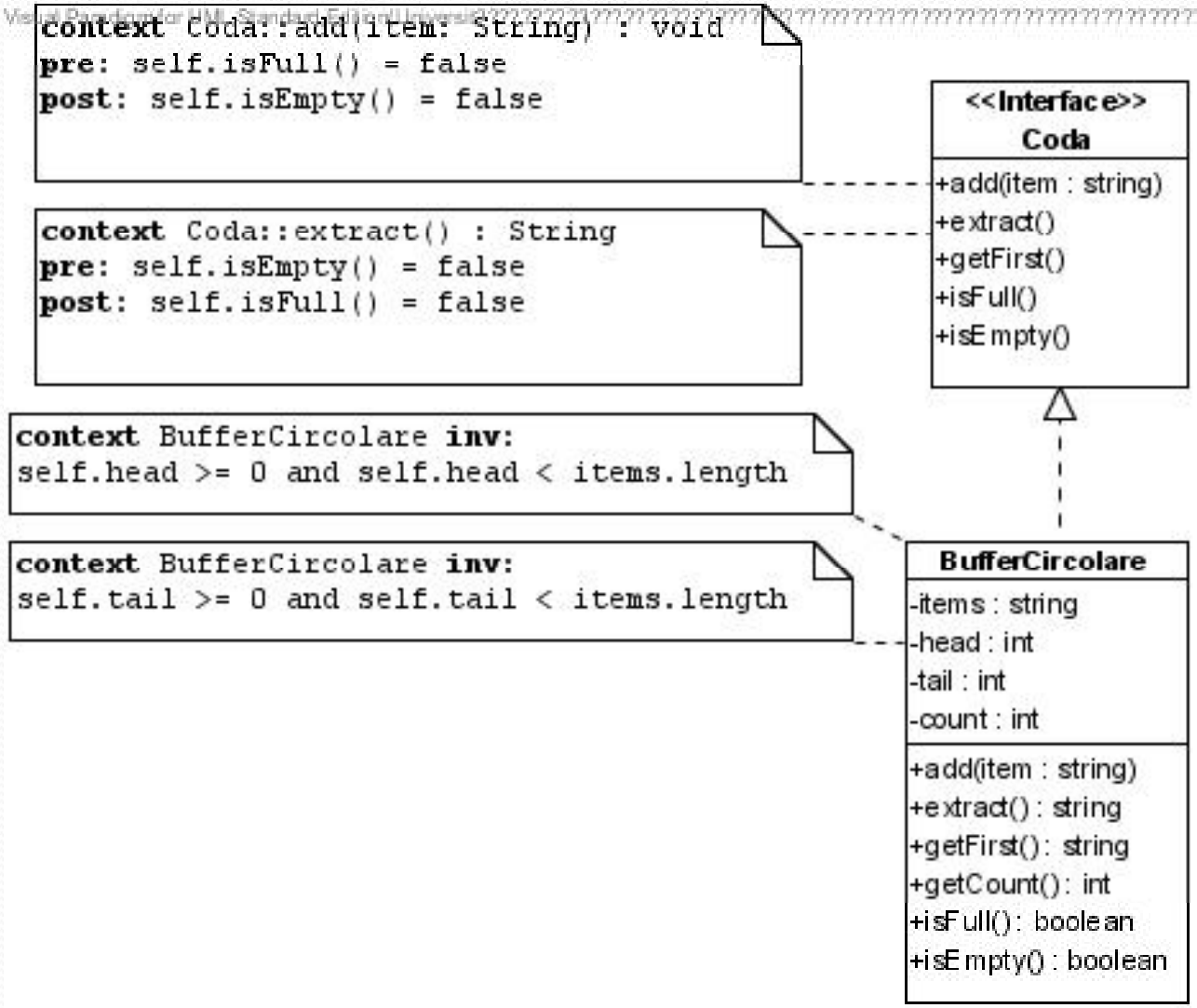
Post-condizioni

- Le due clausole
  - Dopo aver inserito un elemento in una coda vuota la coda non è più vuota
  - Dopo aver estratto un elemento da una coda piena la coda non è più piena
- Devono essere garantite dalle operazioni add() e extract() purchè le pre-condizioni siano valide
- Sono post-condizioni

Invarianti

- La doppia clausola
  - I valori di testa e coda devono essere sempre indici validi dell' array (tra 0 e length-1)
- Definisce due condizioni che devono valere sempre per la classe nel suo insieme
- Sono invarianti di classe: devono valere per ogni istanza della classe
  - Devono essere veri prima e dopo l'invocazione di un metodo
  - Possono essere temporaneamente violati durante l'exec

UML: Uso di OCL per i contratti



Come si esprimono le pre e post condizioni

- In un linguaggio dotato di una sintassi adeguata si potrebbe scrivere ad es.

```
public void add(string item)
precondition: !isFull();
postcondition: !isEmpty();
{items[tail] = item;
tail = (tail + 1) % items.length;
count++;}
```

```
public string extract()
precondition: !isEmpty();
postcondition: !isFull();
{string s = items[head];
head = (head + 1) %
    items.length;
count--; return s;}
```

Le precondizioni devono poter essere verificate dal chiamante e quindi devono essere espresse in base a soli elementi pubblici

Come si esprimono gli invarianti

- In un linguaggio dotato di una sintassi adeguata si potrebbe scrivere ad es.

```
public class BufferCircolare
{private string[] items;
private int head, tail, count;

invariant: head >=0 && head < item.length;
invariant: tail >=0 && tail < item.length;

}
```

Gli invarianti vengono verificati all'interno della classe e quindi si possono basare su elementi privati

Caso VolBank: 1 iterazione

- Durante la prima iterazione dell' elaborazione i requisiti selezionati sono:
  - Scenario di base del caso d'uso **Combinare disponibilità/richiesta**
- Creazione del modello di dominio
- Diagramma(i) di sequenza di sistema
- Contratti delle operazioni