



Introduzione a UML 2.0 – 1^a parte



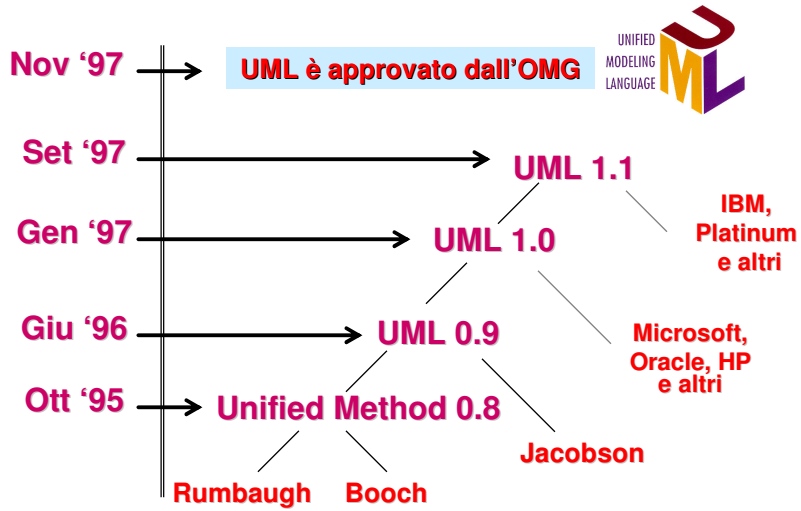
Per un'introduzione generale ai diagrammi UML si rimanda al tutorial "**Introduzione UML**" presente su questo sito



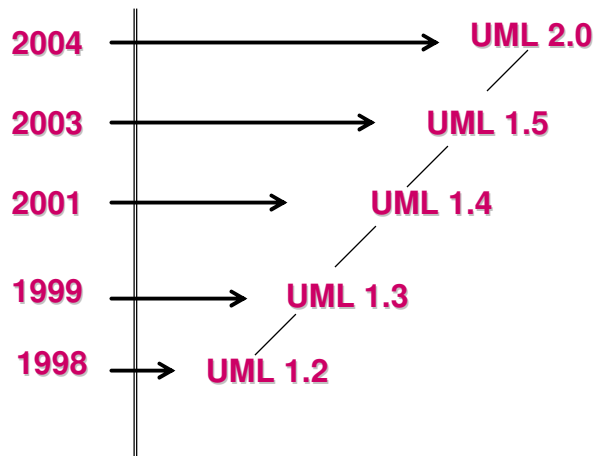
UML

- E' un linguaggio che serve per visualizzare – specificare – costruire – documentare un sistema (software, di business, ...) e gli elaborati prodotti durante il suo sviluppo
- Ha una semantica e una notazione standard, basate su un meta-modello integrato, che definisce i costrutti forniti dal linguaggio
- La notazione (e la semantica) è **estensibile** e **personalizzabile**
- E' utilizzabile per la modellazione durante tutto il ciclo di vita del software (dai requisiti al testing) e per piattaforme e domini diversi
- Combina approcci di:
 - Modellazione dati (Entity/Relationship)
 - Business Modeling (work flow)
 - Modellazione ad oggetti
 - Modellazione di componenti
- Prevede una serie di diagrammi standard, che mostrano differenti viste architettrurali del modello di un sistema

Storia di UML – 1ª parte



Storia di UML – 2ª parte



Obiettivi di UML 2.0

Alcuni tra i principali obiettivi di UML 2.0 sono stati:

- Rendere UML più facile da applicare, far evolvere e personalizzare
- Migliorare il supporto allo sviluppo basato sui componenti
- Arricchire i modelli architetturali
 - affinamento della descrizione del comportamento dinamico e possibilità di rappresentare la scomposizione gerarchica dei sistemi
- Migliorare la scalabilità, la precisione e l'integrazione dei diagrammi di comportamento
 - Arricchire e ottimizzare i diagrammi di sequenza
 - Migliorare il supporto per il business process modeling
 - Permettere l'esecuzione di modelli (simulare come il sistema lavorerà)
- Consentire lo scambio di modelli e diagrammi su tools diversi

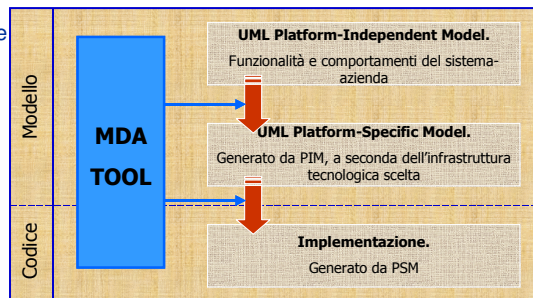
Compatibilità con le versioni precedenti: è comunque possibile continuare a utilizzare UML 1.x

Model Driven Architecture

- MDA di OMG è un framework per un insieme di standard che supportano in modo completo lo sviluppo basato sui modelli (**Model Driven Development**)
- Il focus è sui modelli, non sul codice (approccio Model Driven vs. approccio Code Centric)
- Il cuore di MDA è un insieme di standard (MOF, UML, CWM e XMI) e di tecnologie (CORBA, .NET, Java, ...). A partire da tali standard e servizi è possibile costruire profili specifici di dominio (finanza, e-commerce, biotecnologia, ...) e realizzare, nell'ambito di ogni dominio, specifiche applicazioni conformi agli standard di supporto

Model Driven Architecture

- MDA suddivide lo sviluppo del software in 3 passi principali:
 - creazione del **Platform Independent Model** (il PIM è il modello che definisce l'essenza del sistema, indipendentemente dalla sua implementazione tecnologica)
 - trasformazione del PIM in **Platform Specific Model** (il PSM è il modello specifico di piattaforma, che consente di "mappare" il PIM su specifiche tecnologie)
 - generazione del software



Model Driven Architecture

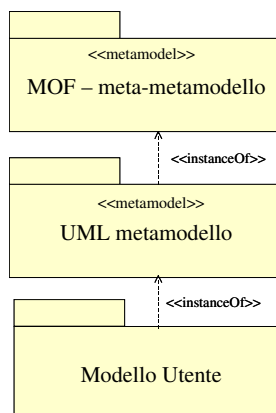
- La definizione di PIM e PSM si avvale di Patterns predefiniti, personalizzabili e riusabili, così come la trasformazione da PIM a PSM (patterns tecnologici) e la generazione del software (patterns di implementazione) a partire dal PSM
- Una volta inserite tutte le informazioni necessarie nel modello, gli strumenti MDA potrebbero *generare automaticamente* gran parte del *codice* necessario per implementare un sistema, grazie soprattutto alla potenza del linguaggio di modellazione (UML), che consente di specificare in dettaglio il comportamento di un sistema

Meta-Object Facility

- Il **MOF** è il cuore della strategia MDA
- Definisce un metalinguaggio comune per costruire altri linguaggi: UML, CWM
- Il Common Warehouse Model (**CWM**) definisce un metalinguaggio per descrivere i data warehouse e i sistemi correlati. Fornisce ad es. il "mapping" dei modelli PIM di MDA negli schemi di database
- **XMI** (XML Metadata Interchange) definisce un mapping da UML a XML, rendendo possibile convertire un modello UML in formato XML, distribuirlo e riconvertirlo in formato UML

Relazioni MOF-UML-modelli

La figura illustra le relazioni tra il meta-metamodello di MOF, il metamodello di UML ed un modello dell'utente



Esempio: la classe **Contratto**, presente nel Modello Utente, è un'istanza della metaclassa **Classe** presente nel metamodello di **UML**, che è a sua volta un'istanza della metaclassa **Classe** del metamodello di **MOF**

E' possibile visualizzare un ulteriore livello, dipendente dal Modello Utente, in cui mostrare le istanze (gli oggetti) della classe **Contratto** (es: contratto 1113, contratto 2002,...)

Overview su UML 2.0

UML 2.0 comprende:

- Infrastructure
- Superstructure

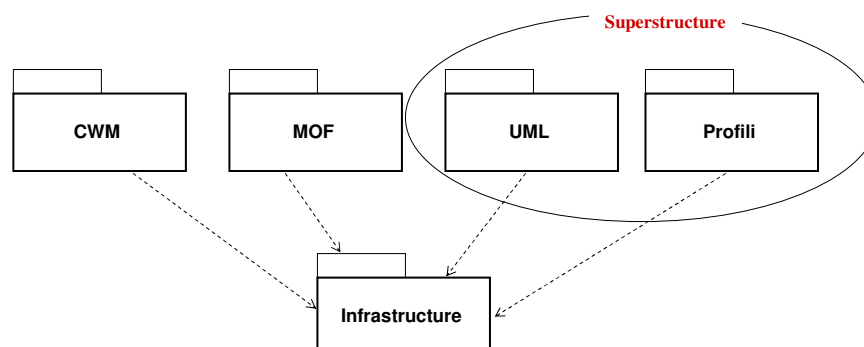
L'**Infrastructure** definisce il nucleo del metamodello usato per definire MOF, UML, CWM e i Profili

La **Superstructure** estende e specializza l'Infrastructure per definire il metamodello di UML

Il metamodello di UML è stato completamente rivisto per renderne più semplice l'evoluzione e la manutenzione e per ridurre ambiguità

Overview su UML 2.0

Infrastructure è il metamodello centrale





Superstructure

La **Superstructure** contiene tutti gli elementi utilizzati per costruire i diagrammi UML, suddivisi in tre categorie:

- elementi di comportamento (**Behavior**)
- elementi di struttura (**Structure**)
- elementi supplementari (**Supplement**)

Supplement definisce i costrutti ausiliari (information flow, modelli, tipi primitivi e template) e i Profili

I **Profili** rappresentano un tailoring di UML per utilizzarlo nell'ambito di una specifica piattaforma o di uno specifico dominio.



Profili

I meccanismi di estensione consentono di creare nuovi **Profili** di UML, ossia collezioni di stereotipi, proprietà e regole che specializzano lo standard UML per adattarlo opportunamente ad uno specifico dominio o scopo

Alcuni Profili sono già stati standardizzati da OMG, altri sono in corso di standardizzazione

Esempi:

- **profilo per il processo di sviluppo del software**
- **profilo per il business modeling**, per la modellazione dei processi di business aziendali e dei ruoli organizzativi coinvolti

Diagrammi di UML 2.0

Diagrammi di struttura:

- diagramma delle classi (class)
- diagramma degli oggetti (object)
- diagramma delle strutture composite (composite structure)
- diagramma dei componenti (component)
- diagramma di deployment (deployment)
- diagramma dei package (package)

Diagrammi di comportamento:

- diagramma dei casi d'uso (use case)
- diagramma di stato (state machine)
- diagramma delle attività (activity)

diagrammi di interazione:

- diagramma di comunicazione (communication)
- diagramma dei tempi (timing)
- diagramma di sintesi delle interazioni (interaction overview)
- diagramma di sequenza (sequence)

Diagramma delle classi

UML 2.0 distingue due tipi di interfaccia: fornita (**provided interface**) e richiesta (**required interface**)

Un'interfaccia fornita è un servizio che una classe rende disponibile ad altre classi

Un'interfaccia richiesta è un servizio fornito da altri e di cui la classe necessita per operare opportunamente in un particolare ambiente

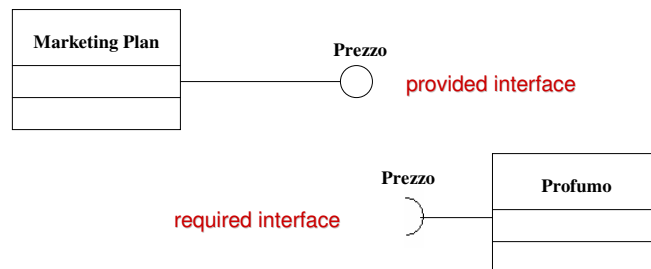


Diagramma delle classi

UML 2.0 aggiunge il nuovo concetto di **PORTA**

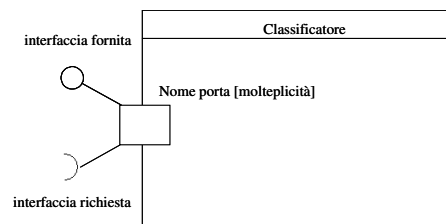
La **porta** è una caratteristica di un classificatore che specifica un punto di interazione tra il classificatore e il suo ambiente o tra il (comportamento del) classificatore e le sue parti interne in termini di interfacce richieste e fornite

Una classe può essere utilizzata in differenti modi, poiché i diversi interlocutori possono richiedere insiemi di servizi differenti. Una porta consente di raggruppare le interfacce appartenenti o destinate a particolari interlocutori, fornendo una "vista" specifica di una classe.

Le porte sono connesse con altre porte tramite connettori (connector)

Una porta connette le parti interne (internals) di una classe al suo ambiente.

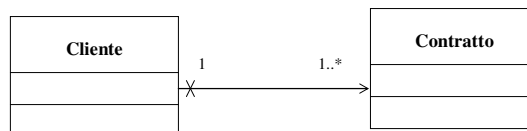
I concetti di porta, interfaccia richiesta e fornita, connettore, ... si applicano anche ai componenti.



Pag. 17

Diagramma delle classi

Per le opzioni di navigabilità dell'associazione è stata aggiunta la "X" per indicare l' **assenza di navigabilità**



La figura modella un'associazione in cui la classe *CLIENTE* può accedere alle istanze di *CONTRATTO*, ma un *CONTRATTO* non può accedere al *CLIENTE* associato

Pag. 18

Diagramma delle classi

In UML 2.0 è stata eliminata l'aggregazione semplice, rimane solo la composizione (is-part-of)

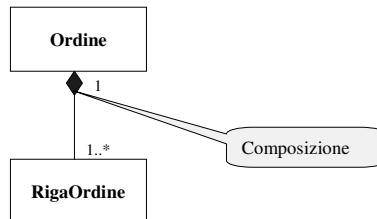


Diagramma delle classi

UML 2.0 ha aggiunto il nuovo concetto di "substitution dependency" (dipendenza di sostituzione)

E' un tipo di realizzazione che implica sostituibilità al runtime

Non è basata sul concetto di specializzazione, perché non implica eredità della struttura, ma solo conformità ai contratti pubblici disponibili

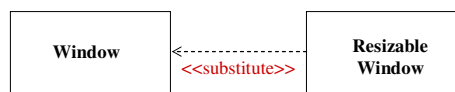


Diagramma delle classi

In UML 2.0 è cambiata la notazione della **classe attiva**

La classe attiva è una classe le cui istanze sono oggetti attivi.

Un oggetto è attivo se “ha un proprio thread di controllo”; è un oggetto che, come diretta conseguenza della sua creazione, inizia ad eseguire il suo comportamento (e non cessa fino a che non ha completato tale comportamento oppure non è cancellato da qualche altro oggetto)

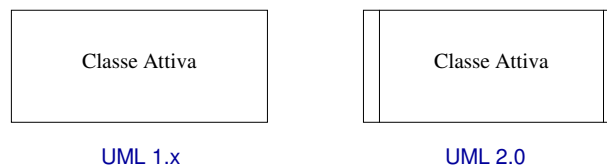
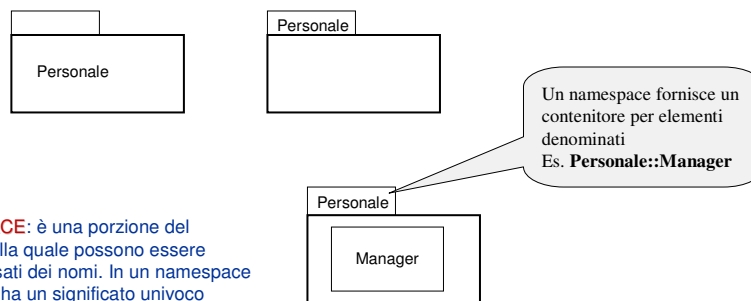


Diagramma dei Package

In UML 1.x non si faceva riferimento esplicito ad un diagramma dei package

Un **package** è utilizzato per raggruppare elementi e fornire loro un **namespace**

Un package può essere innestato in altri package

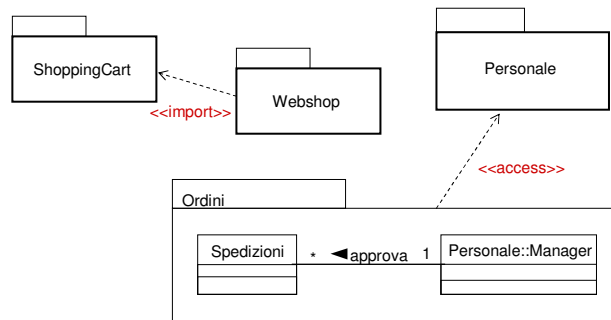


NAMESPACE: è una porzione del modello nella quale possono essere definiti e usati dei nomi. In un namespace ogni nome ha un significato univoco

Diagramma dei Package

IMPORT : è una relationship di dipendenza tra package, che indica che un package contiene una copia del classificatore appartenente ad un altro package

ACCESS : è una relationship di dipendenza tra package, che indica che tutti i classificatori del package origine (nell'es. *Ordini*) possono accedere ai classificatori pubblici del package target (*Personale*)

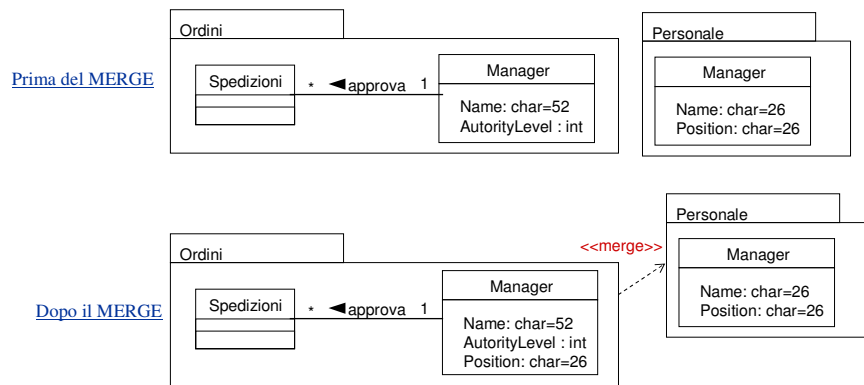


Pag. 23

(c) TECNET DATI

Diagramma dei Package

MERGE (introdotto con UML 2.0) : è una relationship di dipendenza tra package, dove i contenuti del package target (nell'es. *Personale*) sono "uniti" a quelli del package origine (*Ordini*) tramite specializzazione e ridefinizione, dove applicabile



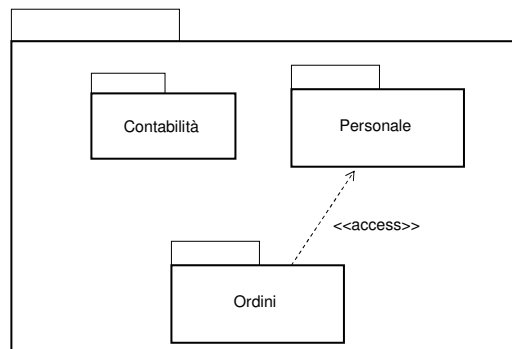
Pag. 24

(c) TECNET DATI

Diagramma dei Package

Un **Package Diagram** è un diagramma che illustra come gli elementi di modellazione sono organizzati in package e le relazioni (dipendenze) tra i package

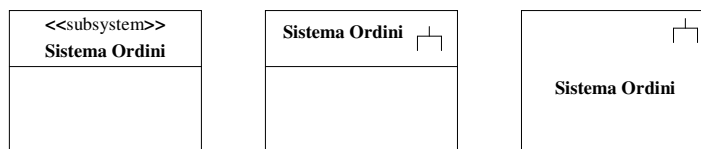
E' simile ad un Class Diagram



Modellare subsystem

Mentre in UML 1.x un **subsystem** è un tipo di package, in UML 2.0 è un tipo di componente

E' quindi possibile specificare per un subsystem le interfacce richieste e quelle fornite, per evidenziare le relazioni con altri subsystem



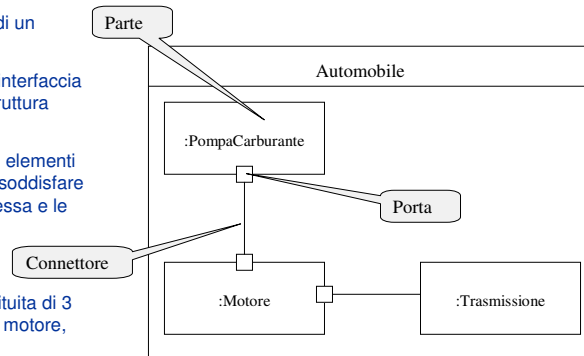
Notazioni alternative per il meccanismo di raggruppamento **subsystem**

Composite Structure Diagram



Il **Diagramma di Struttura Composita** ha l'obiettivo di rappresentare la struttura interna (le parti) di un classificatore (classe, componente, collaborazione), inclusi i punti di interazione (**porte**) utilizzati per accedere alle caratteristiche della struttura

- Mostra la struttura interna di un classfier complesso
- mostra in modo separato l'interfaccia di un classfier dalla sua struttura interna
- descrive i ruoli che i diversi elementi della struttura giocano per soddisfare l'obiettivo della struttura stessa e le interazioni richieste



La classe *Automobile* è costituita di 3 parti: pompa del carburante, motore, trasmissione

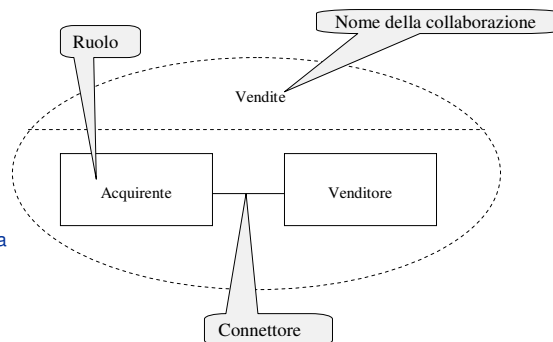
Composite Structure Diagram

Il Composite Structure Diagram può essere utilizzato anche per modellare la struttura di una collaborazione

Una **collaborazione** descrive un comportamento, le risorse utilizzate per fornire il comportamento e i ruoli che le risorse partecipanti assumono durante il comportamento stesso. In UML 2.0 una collaborazione è un tipo di classificatore

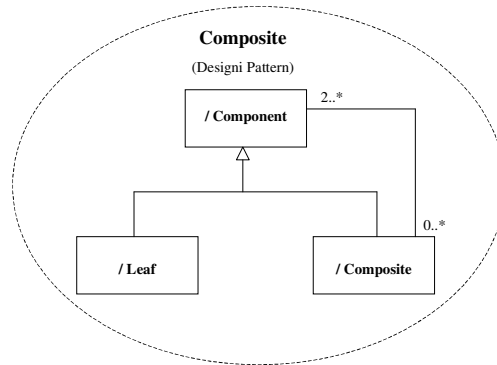
La figura modella la collaborazione tra due ruoli: Acquirente e Venditore.

Un **ruolo** (**ClassifierRole**) definisce un insieme di caratteristiche (attributi e comportamenti) che il classificatore deve possedere per rivestire una particolare responsabilità nell'ambito di una collaborazione



Composite Structure Diagram

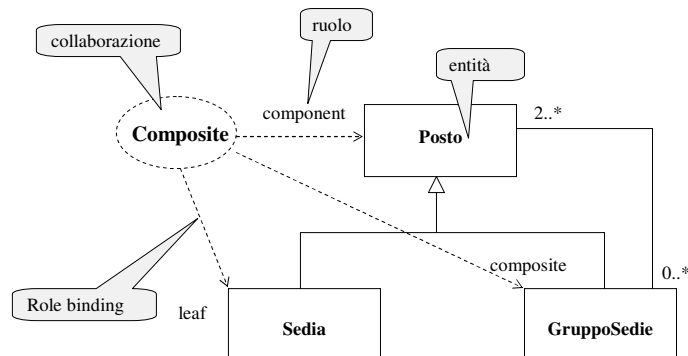
Le collaborazioni sono spesso utilizzate per modellare i pattern di progettazione



Composite Structure Diagram

Un'occorrenza di collaborazione è un particolare uso di una collaborazione per spiegare i legami tra le parti di un classificatore o le proprietà di un'operazione. Definisce un insieme di ruoli e connettori che cooperano nell'ambito del classificatore o dell'operazione.

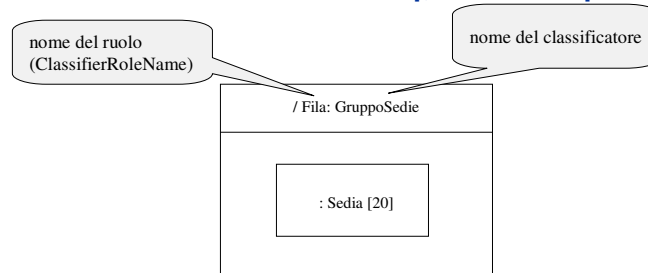
Rappresenta una singola istanza della collaborazione, legata ad uno specifico insieme di elementi (classificatori e associazioni)



Composite Structure Diagram

La sintassi completa per il ClassifierRole è la seguente:

ObjectName '/' ClassifierRoleName ':' ClassifierName [';' ClassifierName]



Nella fig. la classe *GruppoSedia* assume il ruolo di *Fila*. La fila è composta di 20 istanze di *Sedia*

Diagramma dei componenti

- Un **componente** è un'unità modulare sostituibile in un dato ambiente
- Specifica un contratto formale di servizi offerti e richiesti in termini di interfacce (eventualmente esposte tramite porte)
- Un componente è tipicamente specificato da uno o più classificatori (ad es. classi) e può essere implementato da uno o più artefatti (file eseguibile, script, ...)
- Gli **internals** (parti interne) sono inaccessibili se non attraverso le interfacce

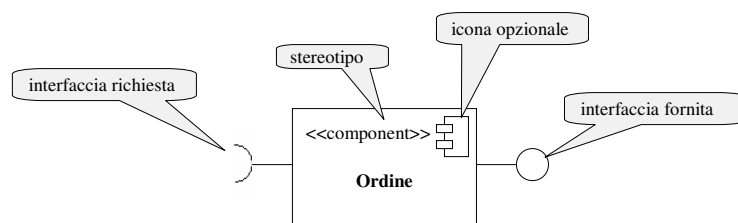


Diagramma dei componenti

Mentre in UML 1.x il componente è una specializzazione della metaclassa Classifier, in UML 2.0 è una specializzazione della metaclassa Class.

Quindi un componente può avere attributi e metodi, una struttura interna, porte e connettori.

In UML 2.0 l'icona del componente è cambiata

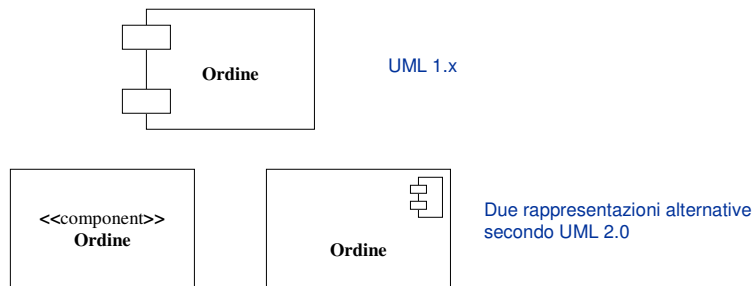
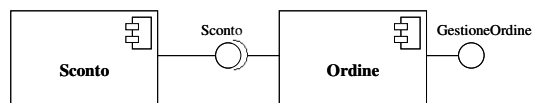


Diagramma dei componenti

Relazioni tra componenti



Interfaccia fornita e interfaccia richiesta devono essere compatibili a livello di tipo (attributi e associazioni) e di vincoli su comportamento (operazioni, eventi)

La fig. illustra una notazione alternativa, in cui le interfacce sono mostrate in modo esplicito.

La relationship tra il componente *Ordine* e l'interfaccia richiesta *Sconto* è una dependency di tipo <<uses>>

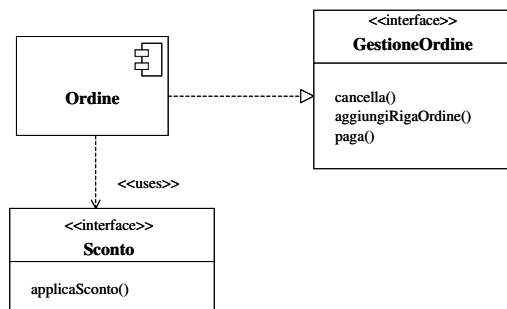


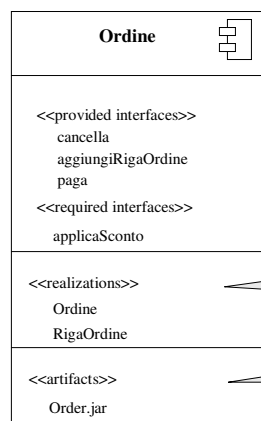
Diagramma dei componenti

I diagrammi Component e Deployment possono essere utilizzati anche per modellare i **sistemi di business**, inclusi i processi manuali

I componenti rappresentano i **processi di business** (es. acquisire un ordine) e i **prodotti** di questi processi (es: contratti, pagamenti, ...), mentre i nodi del deployment diagram rappresentano le **unità organizzative** (es: un reparto) o le risorse di business (es: un magazzino)

Diagramma dei componenti

Segue una rappresentazione **white-box** di un componente:



La notazione è detta **white-box**, perché mostra le proprietà interne del componente

sono le classi o i componenti che lo costituiscono

sono gli artefatti che lo implementano

Diagramma dei componenti

I classificatori interni (**internals**) che realizzano un componente possono essere mostrati in due modi:

- innestati nel componente

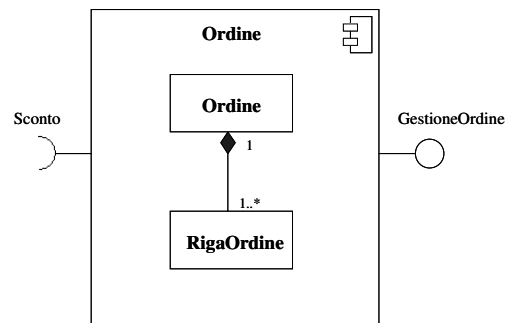
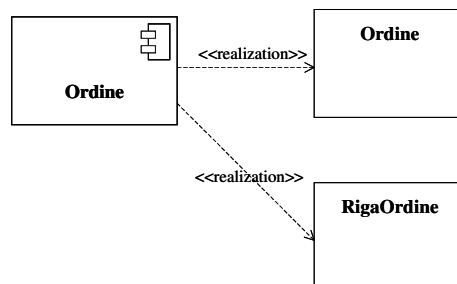


Diagramma dei componenti

- in modo esplicito tramite le relationship di **realization**

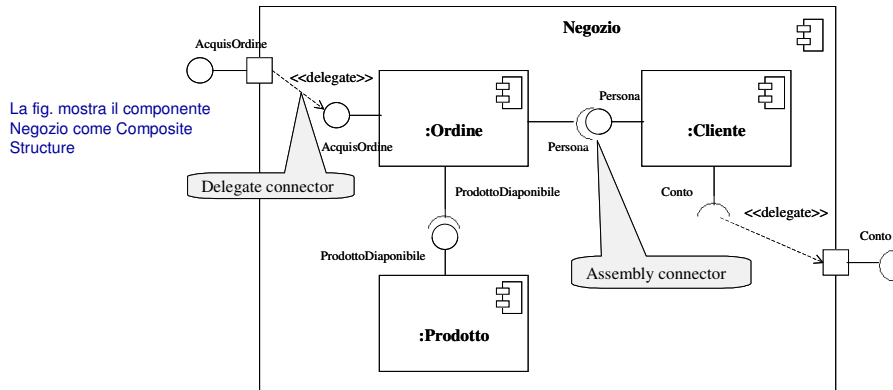


Il componente *Ordine* è implementato istanziando le classi *Ordine* e *RigaOrdine*

La **realization** è una relationship di dipendenza specializzata tra due insiemi di elementi di modellazione, di cui uno rappresenta la **specificata** e l'altro una sua **implementazione**

Per un componente la realization definisce i classificatori che realizzano il contratto offerto dal componente stesso in termini delle sue interfacce offerte e richieste

Diagramma dei componenti



Le parti interne sono collegate direttamente tra loro (**assembly connector**) oppure connesse a porte sul confine del componente (**delegated connector**). I delegated connector sono utilizzati per esporre servizi di una "parte" all'esterno del container.

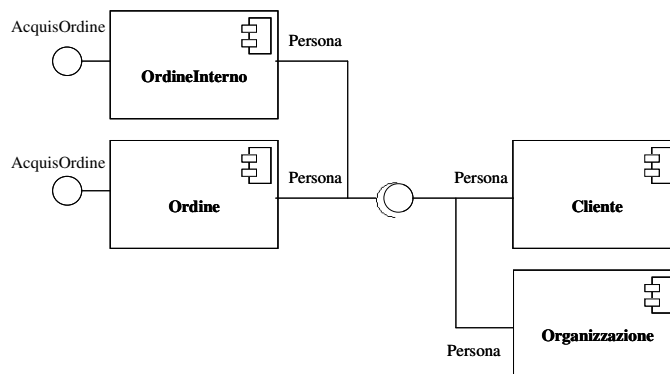
Pag. 39

(c) TECNET DATI

Diagramma dei componenti

La fig. illustra connettori del tipo **Multiple Wiring**

Entrambi i componenti, *Ordine* e *OrdineInterno*, richiedono l'interfaccia *Persona*: l'applicazione non conosce, fino al momento dell'esecuzione, quale componente, *Cliente* o *Organizzazione*, fornirà il servizio richiesto. Si tratta di un'interazione polimorfica.

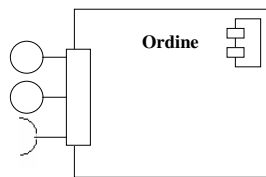


Pag. 40

(c) TECNET DATI

Diagramma dei componenti

UML 2.0 permette di connettere alla stessa porta più interfacce



Il componente ha una porta complessa, con interfacce offerte e interfacce richieste

Deployment Diagram

UML 2.0 ha definito meglio i concetti di Node, Device e Artifact.

Node

- Il nodo è un'unità sulla quali risiedono e/o sono eseguiti componenti / artifact
- I nodi comunicano tra loro tramite CommunicationPath

Artifact

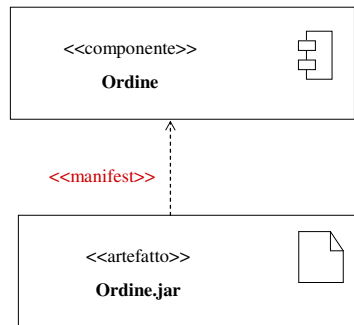
- Rappresenta una specifica porzione fisica di informazioni utilizzata o prodotta dal processo di sviluppo del software. Esempi di artifact (manufatti): i modelli (un diagramma dei casi d'uso, un diagramma delle classi, ...), file sorgenti, script, file eseguibili, ...

Device

- E' una risorsa fisica computazionale con capacità elaborative sulla quale possono essere allocati artefatti per l'esecuzione

Deployment Diagram

“Manifest” è la relationship di dipendenza che illustra gli elementi di modellazione utilizzati nella costruzione o generazione di un artefatto



Il componente rappresenta un **tipo di implementazione** fisica, l'artefatto è l'attuale implementazione

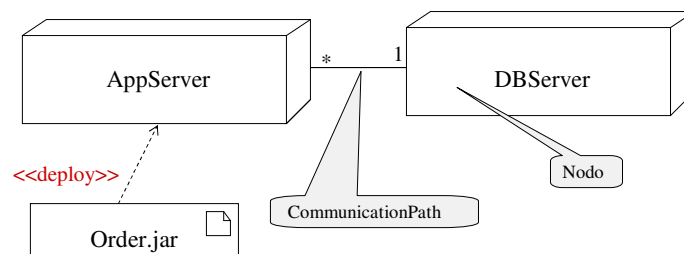
In base ai principi di MDA, lo stesso tipo di componente può essere implementato in differenti tecnologie e quindi in differenti artefatti fisici

Pag. 43

Deployment Diagram

Un nodo può rappresentare qualsiasi cosa che può eseguire un lavoro: un server, un device oppure un essere umano o un'unità organizzativa

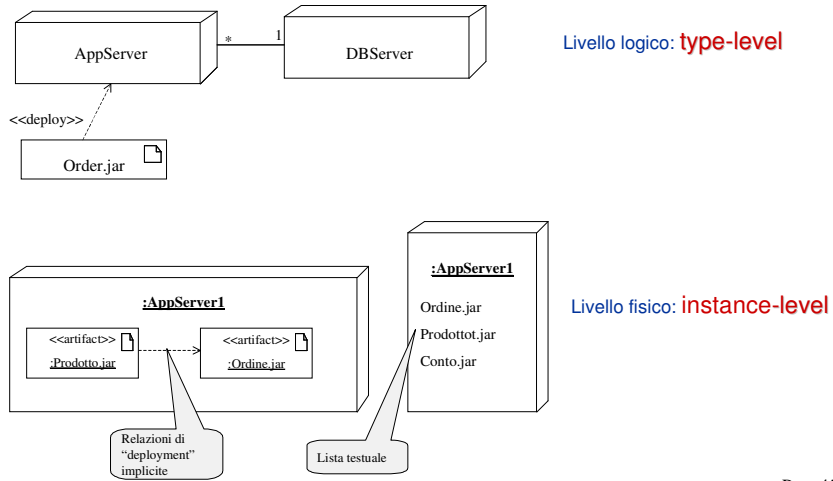
E' una risorsa su cui gli artefatti possono essere allocati per l'esecuzione, questo fatto viene rappresentato con una relationship di dipendenza con lo stereotipo <<deploy>> tra il nodo e l'artefatto



Un **communication path** è un'associazione tra due nodi tramite la quale i nodi possono scambiarsi segnali e messaggi

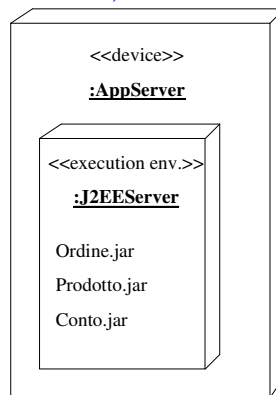
Pag. 44

Deployment Diagram



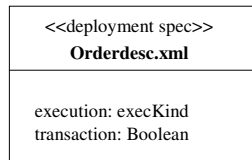
Deployment Diagram

L' **Execution Environment** è un nodo che offre l'ambiente per l'esecuzione di specifici tipi di componenti che sono allocati su di esso (<<OS>>, database system>>, <<J2EE container>>, ...)

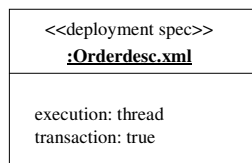


Deployment Diagram

Deployment Specification: è un insieme di proprietà che determinano i parametri di esecuzione di un artefatto allocato su un nodo



Type-level (Specification level)



Instance-level

Alcune delle novità più importanti

- Nuove possibilità di modellare l'architettura dei sistemi e maggiore supporto allo sviluppo basato sui componenti
 - Interfacce richieste e fornite
 - Porta
 - Scomposizione gerarchica e incapsulamento di componenti composti
- Arricchimento dei diagrammi di sequenza
 - Riferimento ad altre interazioni
 - Schema di flusso delle interazioni (Interaction Overview Diagram)
 - Frame di interazioni riusabili
 - Operazioni applicabili alle interazioni (alternativa, parallelismo, iterazione,...)
- Potenziamento dei diagrammi di attività
 - Nuovi fondamenti semantici (Reti di Petri vs. diagramma di stato) per una maggiore flessibilità nell'utilizzo
 - Miglior supporto per il business process modeling



Documentazione UML

Per i documenti ufficiali di UML si rimanda al sito di Object Management Group: www.omg.org

