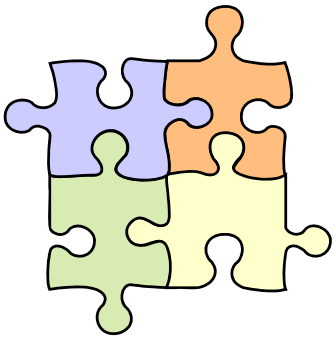
Luca Cabibbo

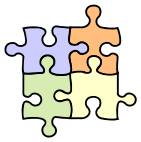


Architetture Software

Architetture basate su componenti

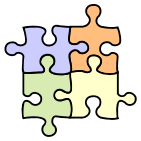
Dispensa PA 4

ottobre 2008



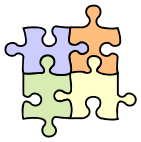
- Fonti

- [Cheesman&Daniels] UML Components – un semplice processo per la specifica di software basato su componenti
- [POSA4] Pattern-Oriented Software Architecture – A Pattern Language for Distributed Computing
- [CDK/4e] Distributed Systems – Concepts and Design



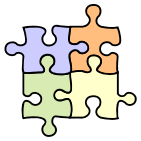
- Obiettivi e argomenti

- Obiettivi
  - conoscere alcuni aspetti relativi alle architetture basate su componenti ed alle relative tecnologie
- Argomenti
  - introduzione
  - componenti
  - nozioni legate ai componenti
  - componenti e contenitori
  - container [POSA4]
  - componenti e interfacce
  - discussione



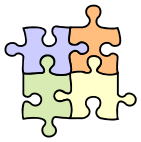
* Introduzione

- Le architetture client/server, ad oggetti distribuiti e di messaging
  - sono alla base di molte tecnologie per sistemi distribuiti
  - sono evolute per semplificare ulteriormente lo sviluppo dei sistemi distribuiti
- Middleware per componenti
  - evoluzione del middleware per oggetti distributi
  - i componenti vivono in contenitori (application server) in grado di gestire la configurazione e la distribuzione dei componenti, e fornire ad essi funzionalità di supporto
  - possibile sia la comunicazione nello stile procedurale che basata sullo scambio di messaggi



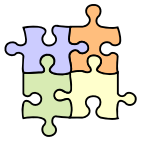
* Componenti

- Esistono varie accezioni legate ai componenti software
  - da una parte, le *tecnologie a componenti* rappresentano un'evoluzione delle tecnologie ad oggetti distribuiti
  - d'altra parte, lo *sviluppo del software basato su componenti* (*CBSD* o *CBSE*) è un approccio alla costruzione di grandi sistemi software basato sullo sviluppo e sull'integrazione di componenti software



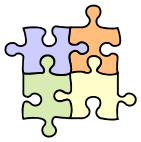
Tecnologie a componenti

- Le *tecnologie a componenti* rappresentano un'evoluzione delle tecnologie ad oggetti distribuiti
  - in generale, un *componente* è un fornitore runtime di servizi
  - un componente è un oggetto distribuito a grana grossa, che vive in un contenitore
    - i componenti possono essere composti al tempo del rilascio (deploy)



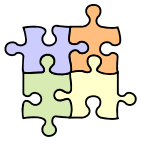
Sviluppo del sw basato su componenti

- Lo *sviluppo del software basato su componenti* (*CBSD* o *CBSE*) è un approccio alla costruzione di grandi sistemi software basato sullo sviluppo e sull'integrazione di componenti software
  - alcuni componenti potrebbe essere pre-esistenti (acquisiti da terze parti o sviluppati in precedenza) oppure sviluppati appositamente come tali
  - la specifica e l'integrazione di componenti rivestono un ruolo fondamentale



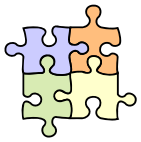
Definizione di componente

- Una possibile definizione – un *componente* è
  - un'unità software eseguibile indipendente
  - definita in termini di un modello standard per componenti (component model)
  - la cui interfaccia, pubblicata, definisce contrattualmente le sue modalità d'uso
    - le interazioni avvengono tramite l'interfaccia
  - può essere composto (senza modifiche) con altri componenti per creare un sistema software
    - anche la composizione avviene tramite l'interfaccia
    - la composizione può essere fatta anche da terze parti



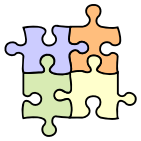
Caratteristiche dei componenti

- Alcune caratteristiche concrete dei componenti
  - un componente è un fornitore di servizi autonomo
    - è indipendente da dove il componente è in esecuzione o dal linguaggio con cui è implementato
    - è un'entità eseguibile – l'uso non richiede ulteriore compilazione – ma deve essere rilasciato (deploy) in un opportuno contenitore (application server)
  - i componenti sono basati su alcuni principi fondamentali del paradigma orientato agli oggetti
    - un componente unifica dati e funzioni
    - incapsulamento – fondamentale per gestire e ridurre le dipendenze tra i diversi componenti che formano un sistema
    - identità – i componenti hanno un'identità, indipendente dal loro stato



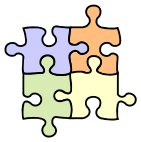
Caratteristiche dei componenti

- Alcune caratteristiche concrete dei componenti
  - i servizi offerti da un componente sono resi disponibili tramite interfacce e contrattualmente
    - l'interazione tra componenti avviene attraverso le loro interfacce
    - la rappresentazione esplicita delle dipendenze tra componenti, a livello di interfacce, è un aspetto centrale nella loro specifica, progettazione ed integrazione
    - l'avere dipendenze basate su interfacce sostiene la sostituibilità dei componenti – con un impatto minimo sui loro client

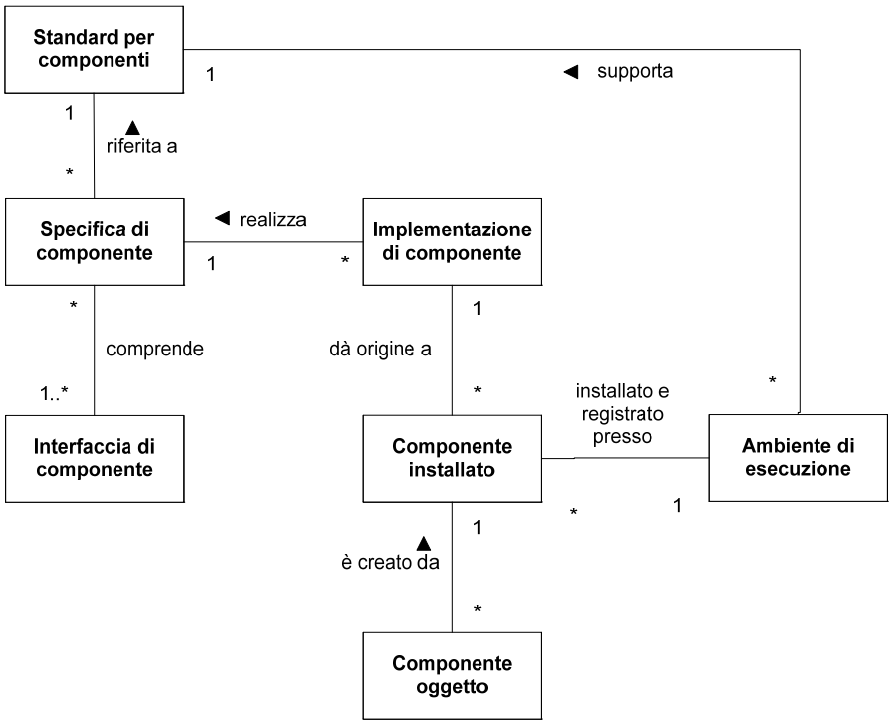


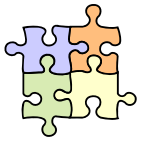
* Nozioni legate ai componenti

- Ci sono varie nozioni legate ai componenti – la parola "componente" può usata con tutti questi significati (correlati ma diversi)
  - standard per componenti
  - specifica di componente
  - interfaccia di componente
  - implementazione di componente
  - componente installato
  - componente oggetto



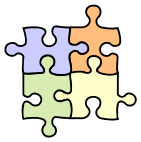
Nozioni legate ai componenti





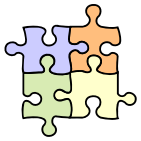
Standard per componenti

- Uno *standard* (o *modello*) *per componenti* definisce uno standard per l'implementazione, la documentazione e il rilascio di componenti
  - alcuni esempi notevoli
    - modello EJB – della piattaforma Java EE
    - modello COM+ – della piattaforma .NET
    - Corba Component Model



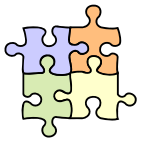
Piattaforma per componenti

- Un *ambiente* (o *piattaforma*) *di esecuzione* fornisce ai componenti i servizi previsti dal relativo standard
  - ad esempio, un *application server*
  - l'ambiente di esecuzione funge da *contenitore* per i componenti eseguibili
  - il relativo standard definisce, tra l'altro, le interazioni previste e/o consentite tra componenti e contenitori



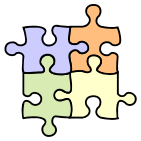
Specifica e interfaccia di componente

- La *specifica di un componente* definisce in modo preciso il comportamento di un componente
  - il comportamento di un componente è definito da un insieme di interfacce – un insieme di operazioni e i relativi contratti
- Gran parte della specifica di un componente consiste nella definizione delle *interfacce del componente*
  - questo insieme di interfacce è anche chiamato collettivamente come l'*interfaccia del componente*
  - definisce l'insieme dei comportamenti forniti da un componente



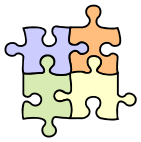
Implementazione di componente

- Un'*implementazione di un componente* è la realizzazione (software) di una specifica di componente
  - si tratta di un componente autonomo, che può essere distribuito ed installato in modo (possibilmente) indipendente da altri componenti
  - la separazione netta tra specifica e implementazione è una caratteristica fondamentale dei componenti
  - per una certa specifica di componente, possono esistere più implementazioni di componenti diverse che la realizzano
- Tre possibili versioni per un'implementazione
  - codice sorgente
  - codice binario
  - unità di distribuzione



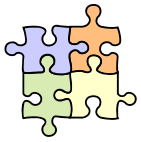
Componente installato

- Un *componente installato* è una copia di un'implementazione di componente che è stata installata su un particolare calcolatore
  - l'installazione avviene su un ambiente (piattaforma) di esecuzione – l'installazione comprende la registrazione del componente in tale ambiente
  - un'implementazione di componente può dar origine a più componenti installati

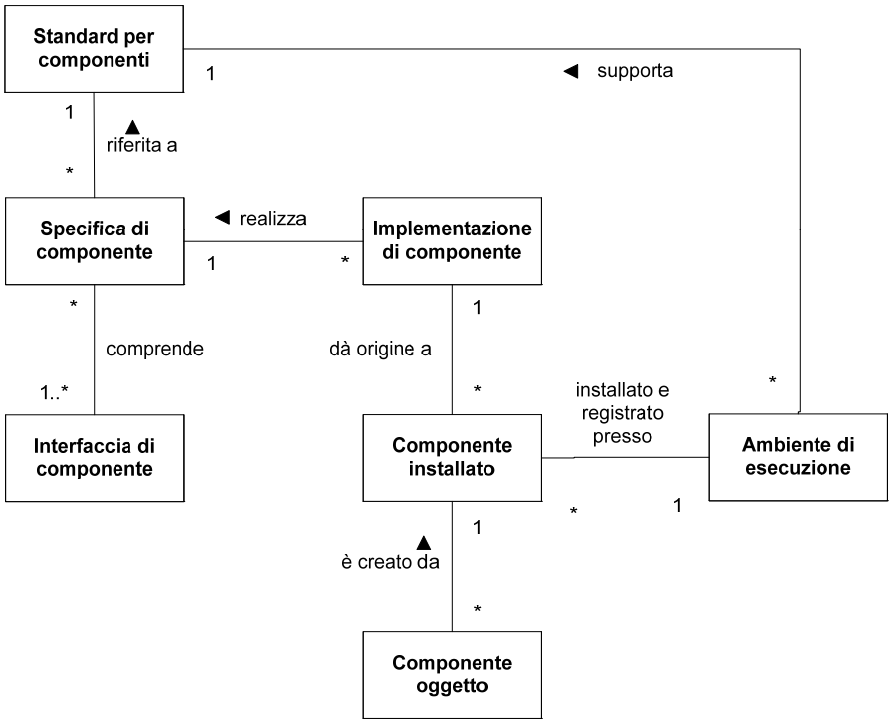


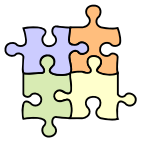
Componente oggetto

- Per *componente oggetto* si intende un'istanza creata a partire da un componente installato, nel contesto di un ambiente di esecuzione
  - un componente oggetto esiste solo durante l'esecuzione
  - i componenti oggetto hanno un'identità univoca, nonché un proprio stato (ovvero, dei dati)
  - un componente installato può avere uno o più componenti oggetto da esso creati
  - sono solo i componenti oggetto ad essere effettivamente in grado di offrire servizi



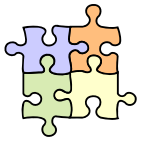
Nozioni legate ai componenti





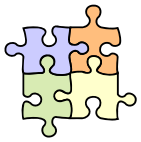
Esempio: MS Word

- MS Word comprende diversi componenti, di cui due “ovvi”
  - lo *standard* per componenti è COM
  - l’*ambiente di esecuzione* è un s.o. Windows, che supporta COM
  - i due *tipi di componenti* “ovvi” sono quelli che rappresentano rispettivamente l’applicazione (WordApp) e un documento (WordDoc)
    - non conosciamo né le loro *specifiche* né le relative *interfacce*
  - il file winword.exe “impacchetta” le *implementazioni* di queste due specifiche di componenti
  - al momento dell’installazione, il file winword.exe viene copiato sul disco, dando origine a due *componenti installati*, che vengono registrati nell’ambiente COM
  - quando viene eseguita l’applicazione, vengono creati due *componenti oggetto*: uno di tipo WordApp per l’applicazione e uno di tipo WordDoc per un nuovo documento
    - ogni nuovo documento aperto sarà rappresentato da un ulteriore *componente oggetto* di tipo WordDoc



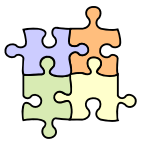
Esempio: un'applicazione web Java EE

- Lo *standard* per componenti è Java EE
- L'*ambiente di esecuzione* è un application server Java EE – ad es., Tomcat
- L'*implementazione* (unità di distribuzione) è il file war – costruito anche sulla base di un descrittore di deployment
- Per essere usata, l'unità di distribuzione deve essere *installata/deployata* sull'application server
- Richieste HTTP all'applicazione causano/possono causare
  - la creazione di *componenti oggetto* coinvolti – ad es., la creazione di oggetti istanza delle classi servlet
  - la richiesta di esecuzione di operazioni a questi oggetti – queste richieste vengono fatte dall'application server alle servlet
- Il ciclo di vita dei vari oggetti dell'applicazione è gestito dall'app. server
  - attenzione – lo stesso oggetto servlet potrebbe essere usato da più sessioni oppure, in una stessa sessione, in interazioni diverse, potrebbe venire usate istanze diverse di una stessa classe servlet



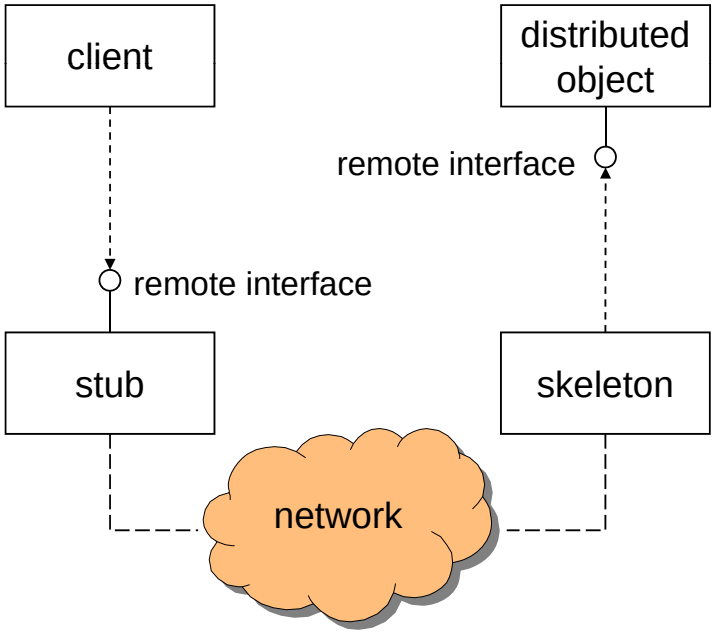
* Componenti e contenitori

- Un aspetto rilevante circa i componenti, è che i componenti "vivono" dentro contenitori
  - un contenitore gestisce il ciclo di vita dei componenti installati presso di lui
  - un contenitore è in grado di fornire alcuni servizi importanti ai suoi componenti
    - descriviamo, intuitivamente, l'evoluzione che ha portato ai contenitori

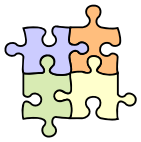


Oggetti distribuiti

- Le DOA consentono l'interazione tra oggetti distribuiti

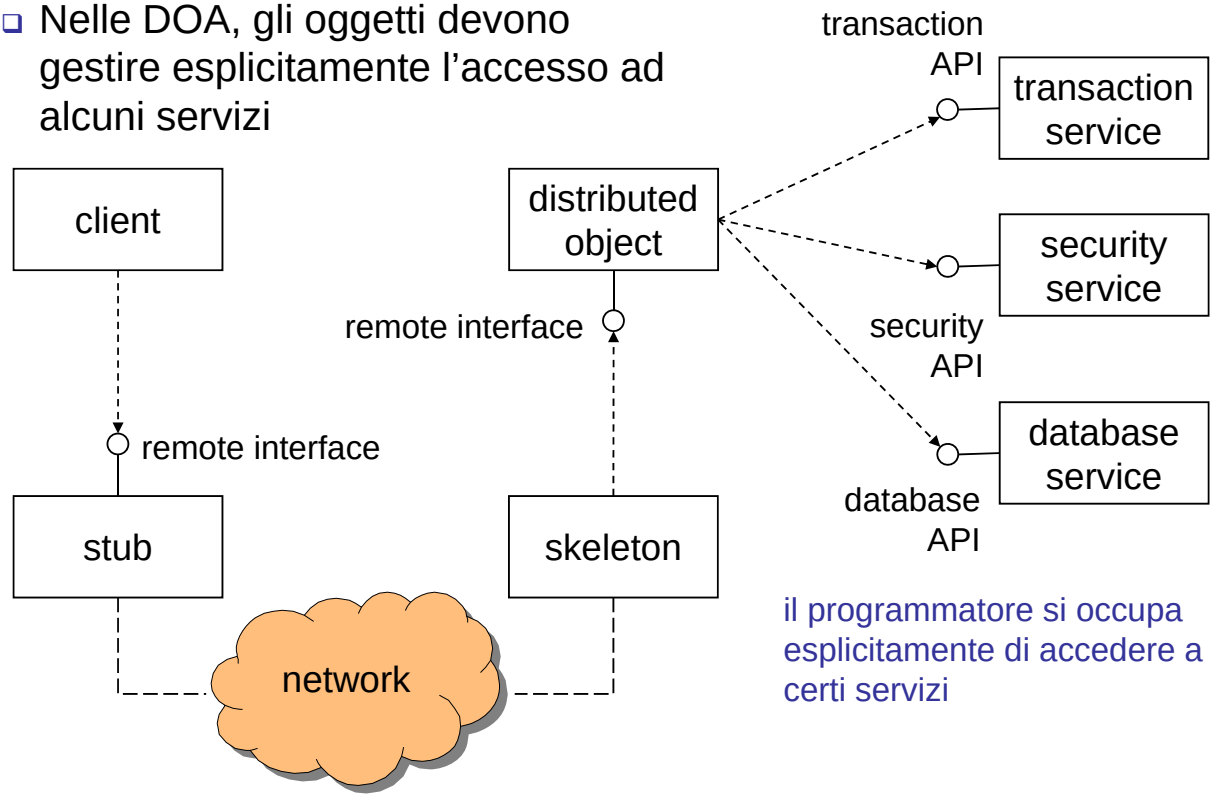


tutte le richieste remote passano attraverso il middleware



Oggetti distribuiti e servizi

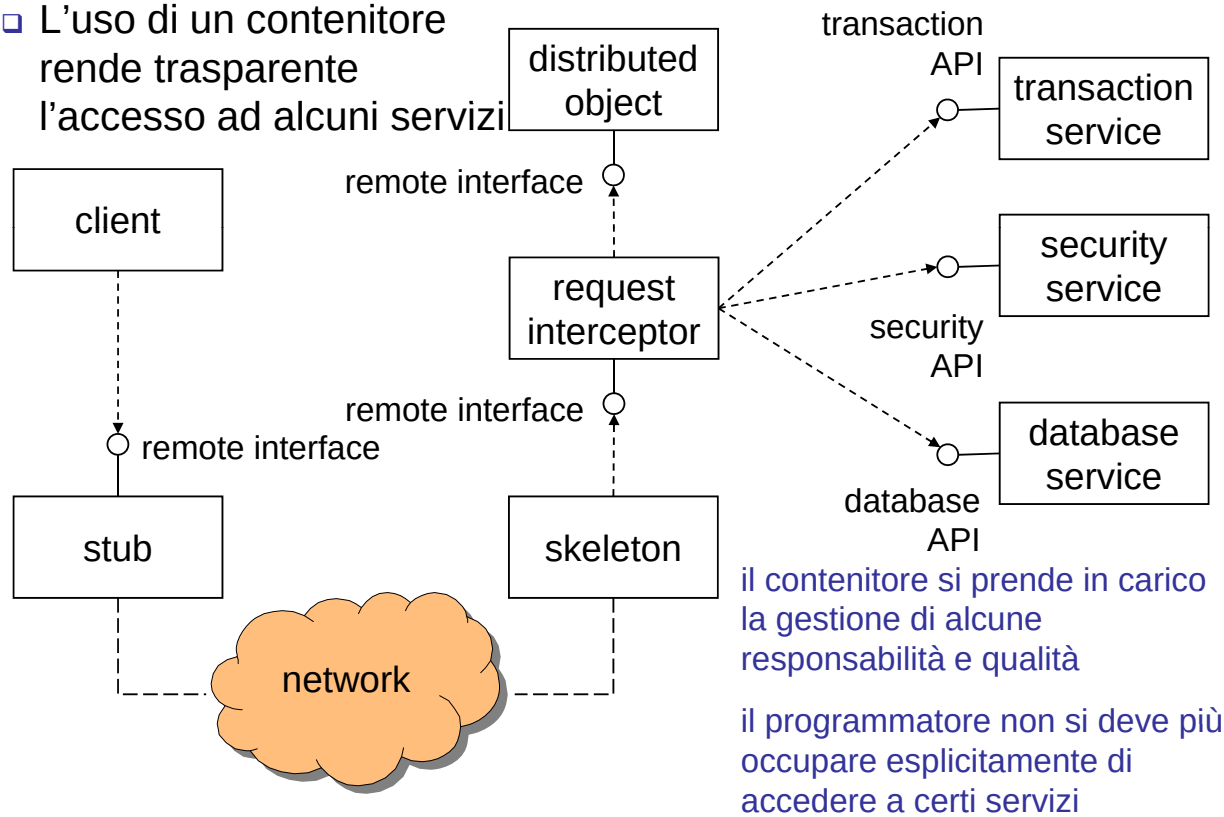
- Nelle DOA, gli oggetti devono gestire esplicitamente l'accesso ad alcuni servizi



il programmatore si occupa esplicitamente di accedere a certi servizi

Gestione trasparente dei servizi

- L'uso di un contenitore rende trasparente l'accesso ad alcuni servizi

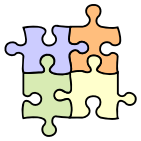


il contenitore si prende in carico la gestione di alcune responsabilità e qualità

il programmatore non si deve più occupare esplicitamente di accedere a certi servizi

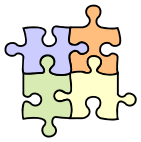
* Container [POSA4]

- Gli ambienti di esecuzione per componenti (application server), specifici per un modello a componenti, sono realizzati sulla base del pattern architetturale **Container**



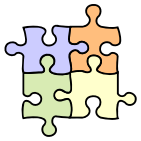
Container

- Contesto
  - ambiente di esecuzione per componenti
- Problema
  - i componenti implementano logica di business (o infrastrutturale) – possono essere usati per comporre applicazioni
  - per favorire la loro composizione, i componenti non devono essere accoppiati direttamente al loro ambiente tecnico di esecuzione o a particolari scenari di esecuzione
    - ad es., ad un particolare application server
    - ad es., uso transazionale o meno, sicuro o meno
  - deve piuttosto essere possibile comporre componenti, senza un esplicito intervento del programmatore
    - in varie piattaforme, con vari scenari di deployment



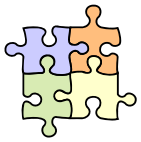
Container

- Soluzione
  - definisci un *container* (*contenitore*) per fornire un ambiente di esecuzione opportuno ai componenti
    - che gestisce il ciclo di vita dei componenti
    - in grado di fornire servizi di supporto ai componenti
      - ad es., persistenza, notifica di eventi, transazioni, replicazione, bilanciamento del carico, sicurezza, …
  - fornisci un mezzo per registrare/configurare/integrare dinamicamente i componenti nel contenitore
    - anche sulla base di specifiche dichiarative



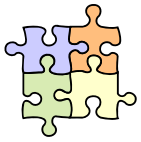
Container

- Il pattern *Container* sostiene e semplifica lo sviluppo e la composizione di componenti
  - consente di separare gli aspetti relativi allo sviluppo dei componenti da quelli relativi al loro deployment e composizione
  - consente al programmatore di focalizzarsi sulla logica applicativa
    - altri interessi possono essere gestiti, spesso con approcci dichiarativi, in sede di deployment
  - fornisce funzionalità di registrazione dei componenti
    - il contenitore funge da “manager” per i componenti registrati, gestendone il ciclo di vita
    - funge inoltre da intermediario nell’accesso ai componenti



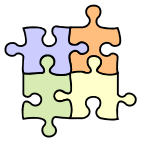
* Componenti e interfacce

- Secondo UML
  - un *componente* è una parte modulare di un progetto di un sistema che nasconde la sua implementazione dietro un insieme di interfacce esterne
- Definizione molto generale
  - ad es., un modulo, un package, un data store, un sottosistema completo
- Un aspetto rilevante dei componenti
  - una separazione netta tra interfaccia ed implementazione
  - le dipendenze tra componenti sono solo in termini di interfacce



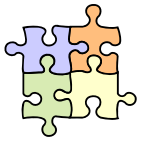
Notazione UML per i componenti

UML 1.x



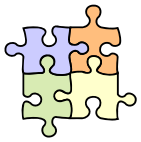
Componenti e interfacce

- Per “interfaccia” si intendono le assunzioni che i componenti possono fare circa gli altri componenti
  - queste assunzioni comprendono nomi delle operazioni e parametri – ma anche i loro contratti, in termini di pre- e post-condizioni, effetti collaterali, consumo di risorse globali, vincoli di coordinamento, service level agreement, ...

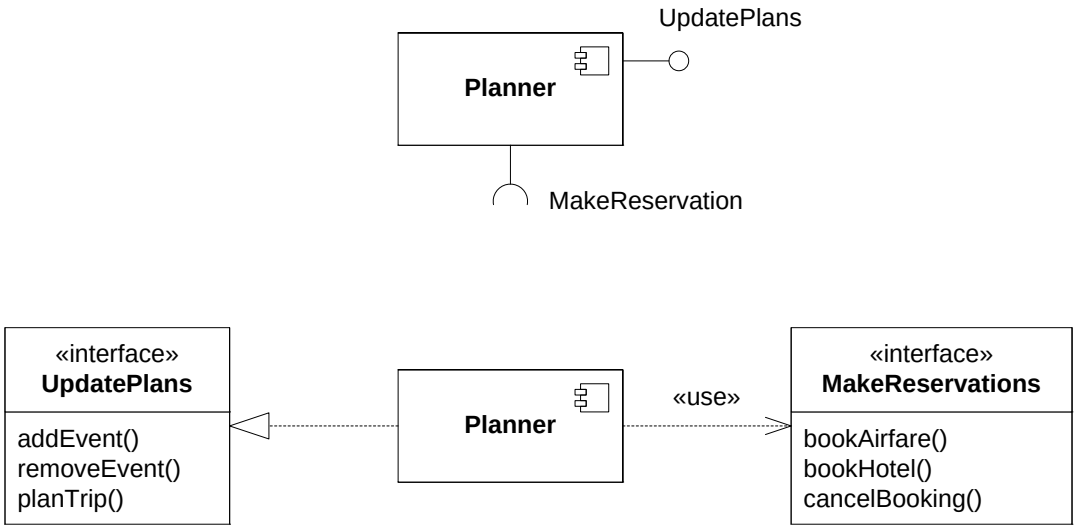


Componenti e interfacce

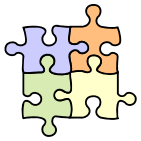
- I componenti sono dotati di due tipi di interfacce
  - *interfacce fornite*
    - specifica dei servizi forniti da un componente ad altri componenti
  - *interfacce richieste*
    - specifica dei servizi che devono essere resi disponibili ad un componente affinché possa funzionare come specificato
    - se non sono disponibili – il componente non funzionerà



Notazione UML per le interfacce

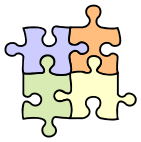


- Un componente
  - fornisce zero o più interfacce (anche più di una)
  - richiede zero o più interfacce (anche più di una)



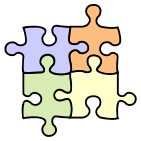
Stili per le interfacce

- I componenti possono interagire tra loro sulla base di due modalità ("stili") di comunicazione
  - stile procedurale
    - comunicazione sincrona – nello stile RPC/RMI
  - stile orientato ai messaggi
    - comunicazione asincrona – nello stile del messaging
- In corrispondenza, l'interfaccia sarà evidentemente espressa con modalità diverse
- Possibili componenti dotati sia di interfacce procedurali che di interfacce orientate ai messaggi



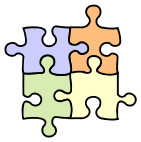
Interfacce e composizione

- La composizione di un certo numero di componenti può essere fatta
  - sulla base delle interfacce fornite e richieste dei componenti
    - specificando quali componenti oggetto servono
    - specificando come sono tra loro collegati – in termini di collegamenti/dipendenze tra componenti, con riferimento alle loro interfacce fornite/richieste
  - in sede di deployment
  - la composizione sulla base delle interfacce consente la sostituzione di componenti – con altri che abbiano le stesse interfacce



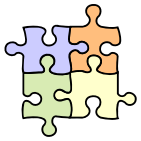
* Componenti e riuso

- Un obiettivo dello sviluppo basato su componenti
  - aumentare la velocità di sviluppo del software – anche mediante il riuso di componenti
- Perché il riuso di componenti (eseguibili) dovrebbe essere migliore/preferibile al riuso di moduli/librerie (codice sorgente)?
  - i benefici della modularizzazione del software sono ben noti
  - con i componenti (eseguibili) la modularizzazione viene sostenuta in modo più rigoroso – che non al livello del codice sorgente
  - la dipendenza a livello delle interfacce sostiene riuso/sostituzione/composizione di componenti



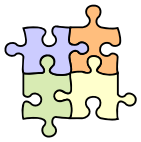
Componenti e riuso

- Lo sviluppo basato sul riuso di componenti pone diversi problemi
- I componenti da riusare vengono normalmente scelti perché forniscono alcune funzionalità
  - queste vengono erogate nel contesto di certe assunzioni architetturali, e dunque di qualità
  - queste assunzioni possono vincolare l'architettura o parte dell'architettura
  - l'architetto deve verificare che queste assunzioni siano accettabili e compatibili (tra i vari componenti)



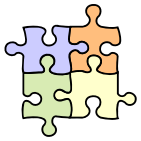
Componenti e riuso

- Per completare rapidamente un prodotto, è possibile scegliere inizialmente di usare componenti anche se non completamente compatibili con la visione definitiva del sistema
  - può essere accettabile, se l'architettura consente un piano di sostituzione dei componenti
    - sostituzione con altri componenti commerciali – se il componente implementa delle interfacce standardizzate
    - altrimenti, sostituzione con componenti costruiti in casa



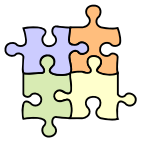
Componenti e riuso

- Le interfacce richieste e fornite di alcuni componenti che devono essere "collegati" potrebbero non corrispondere – si parla di un interface mismatch – quali tecniche per gestirli?
  - cambiare i requisiti – potrebbe essere una soluzione
  - evitare i mismatch
    - specificare con cura l'interfaccia dei componenti – ed ispezionare i componenti scelti per verificare la loro corrispondenza o meno
  - identificare i mismatch che non sono stati evitati
    - ad es., facendo dei prototipi di integrazione
  - riparare i mismatch mediante adattamento dei componenti
    - modificarne l'implementazione non è normalmente possibile
    - va piuttosto definito del "collante" per i componenti
    - adattatori/wrapper, bridge, mediatori, ...



* Discussione

- L'uso delle tecnologie a componenti richiede opportune considerazioni di natura metodologica
  - per guidare quanto meno le seguenti attività – da svolgere in modo congiunto
    - progettazione "statica" – identificazione dei componenti – quali componenti? in quali strati? con quali responsabilità?
    - progettazione "dinamica" – specifica dei componenti – quali interazioni tra componenti? quali interfacce per i componenti?
  - collettivamente, questa attività viene chiamata di "specifica" dell'architettura a componenti



Discussione

- Per aspetti relativi alla programmazione di componenti, vedi anche
  - dispensa su Java EE
- Per aspetti metodologici, vedi anche
  - tutorial su UML Components