



UNIVERSITA' DEGLI STUDI DI BERGAMO  
Facoltà di Ingegneria

## Informatica Industriale

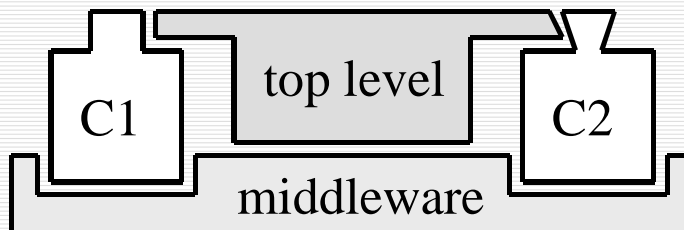
---

Prof. Davide Brugali

### 2.5 – Enterprise Frameworks

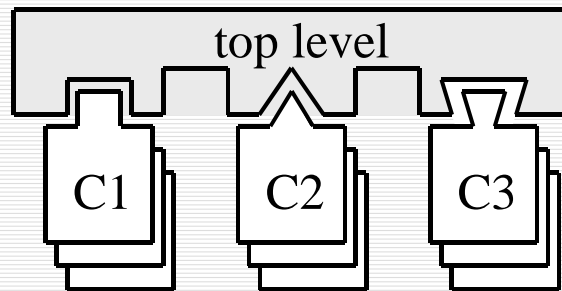
## Middleware Framework

---



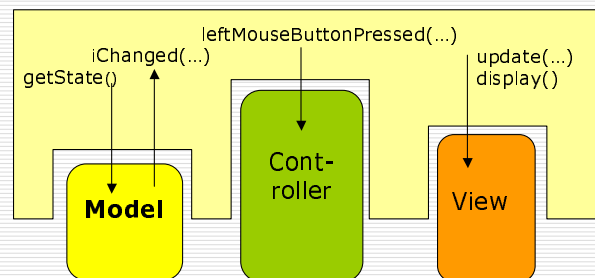
→ Un insieme integrato di componenti software che offre specifici servizi comunemente utilizzati da una famiglia di applicazioni simili

## Application Framework





→ La parte stabile di un'applicazione consiste nel disegno ad alto livello e nell'architettura del sistema che integra componenti selezionati dall'utente.

## Model-View Controller



# Application Framework: Definizione

Class Library	Framework Repository
	
Le Class Libraries sono strutture statiche	I Frameworks sono dinamici
Le Class libraries includono solamente: Design Codice	I Frameworks contengono: Processi Architetture Design Implementazione
Le Class Libraries non includono alcun tipo di flusso applicativo	I Frameworks sono come applicazioni in miniatura
Le Class Libraries forniscono specifiche funzionalità da utilizzare in un applicativo	I Frameworks includono flussi applicativi costruiti su un'architettura predefinita

# Application Framework: Definizione

- **Un set di oggetti interattivi che insieme realizzano un set di funzioni**
  - Il set di funzioni definisce l'area dell' "expertise" o delle "competencies" del framework; in seguito lo chiameremo "dominio del framework"
- **Un dominio può essere sia un set di business domain (problem space), o un set di domain computing (the solution space).**
  - Un framework per applicazioni bancarie implementa funzioni con un problem space in ambito bancario
  - Il framework Model View Controller (MVC), sviluppato per il linguaggio Smalltalk, copre un sottoinsieme di *computing domains*, in particolare il design domain, approcciando il problema di correlare la logica di business con la logica GUI in modo da minimizzare le dipendenze tra le due.

## Application Framework

---

- ❑ **Application framework = a blueprint + realizzazioni di componenti**
- ❑ Blueprint : analysis oppure design model
- ❑ Quando si parla di *business frameworks*, il framework identifica le *domain classes*, le loro *correlazioni*, e le loro *interazioni* (analysis level description), e se possibile il design e la parziale realizzazione di ogni classe, correlazioni e interazioni.

## Modeling Framework

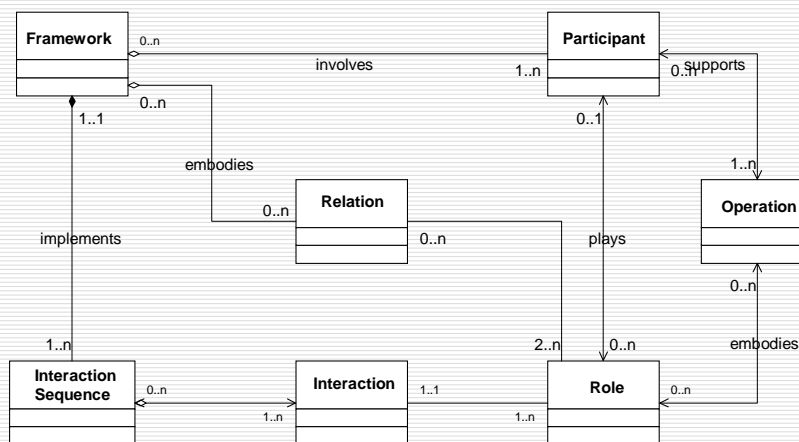
---

- ❑ Tipicamente riguarda costrutti a livello di analisi, senza assumere alcun vincolo su come i componenti del framework verranno progettati e implementati.
- ❑ Analysis-only frameworks, o *modeling frameworks* sono tipicamente un prodotto di *domain analysis*.
- ❑ L'iniziativa IBM San Francisco mira a sviluppare un modeling frameworks per una molteplicità di business domains.

# Structure of Frameworks

- La definizione di un framework deve comprendere almeno:
  - un set di *attori* del framework,
  - un set **di relazioni tra gli attori** del framework,
  - un set di **scenari di interazioni tra gli attori del** framework.
- Gli attori sono generalmente descritti in termini di "**obblighi**": ogni attore ricopre un particolare ruolo all'interno del framework. Tale ruolo è spesso descritto in termini di **interfaccia** che l'attore è tenuto a garantire. L'interfaccia consiste in un set di properties (raramente) and **method signatures** che un componente deve implementare per svolgere quel ruolo.

# Framework Meta-Model



# Architectures

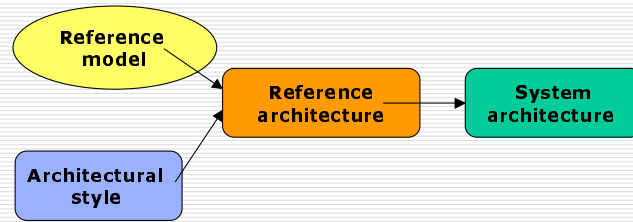
- Il disegno ad alto livello di un sistema software in termini di componenti software (moduli, sottosistemi, processi), le loro proprietà esterne (API, run-time behaviour), e le loro interrelazioni
- Poichè il software tende a essere complesso, abbiamo necessità di più descrizioni (**view**) che riflettano le differenti proprietà dei componenti software e le differenti relazioni tra essi, o differenti decomposizioni del software.

# Architectural Styles

- Uno stile architeturale è una classe di architetture caratterizzata da:
  - **Tipi di componenti:** sono classi di componenti caratterizzate sia da proprietà di sw packaging (e.g. "COM component") sia da ruoli funzionali (e.g. "transaction monitor"), o cocomputazionali ("persistence manager") in un'applicazione
  - **Communication patterns tra i componenti:** indicano il tipo di comunicazione tra i tipi di componenti
  - **Semantic constraints,** indicano proprietà comportamentali dei componenti, singolarmente o nel contesto delle loro interazioni
  - **Un set di connettori,** che sono artefatti software che ci permettono di implementare la comunicazione tra i componenti in modo da soddisfare i **Semantic constraints**

# Enterprise Frameworks

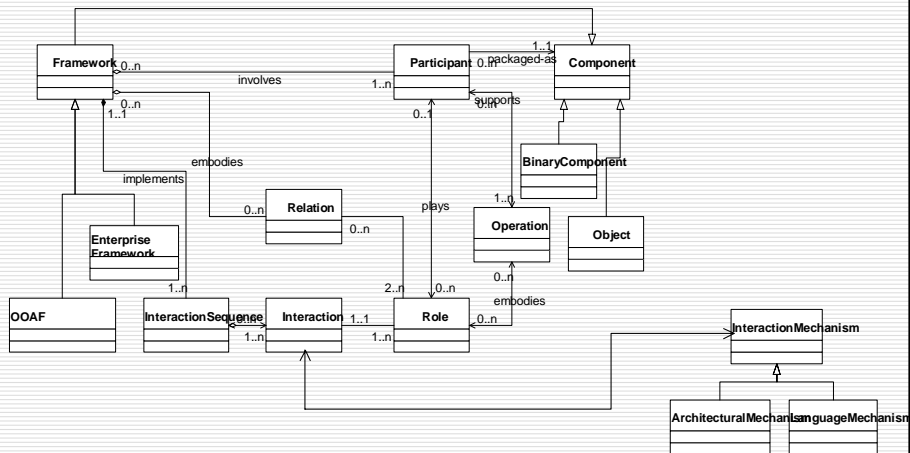
- ❑ Inglobare le semantiche di un dominio applicativo con un'architettura computazionale (o infrastruttura) per un'applicazione in quel particolare dominio.



# Componenti

- ❑ Un *well-designed* framework include un'implementazione dell'infrastruttura computazionale nell'architettura di riferimento, e concrete realizzazioni di alcune componenti funzionali.
- ❑ Tali realizzazioni assumono la caratteristica di templates di codice, oppure di componenti, i.e. componenti binari pronti all'uso come ad ex. COM o Enterprise Java beans.

# Framework Meta-Model



15/40

# Building Enterprise Frameworks

## ❑ Packaging Issues

- Separazione tra aspetti computazionali e aspetti di dominio
- Infrastrutture computazionali per integrare architetture eterogenee

## □ Process Issues

- Domain Engineering
- Sviluppo applicativo guidato dai prodotti
- Framework lifecycle

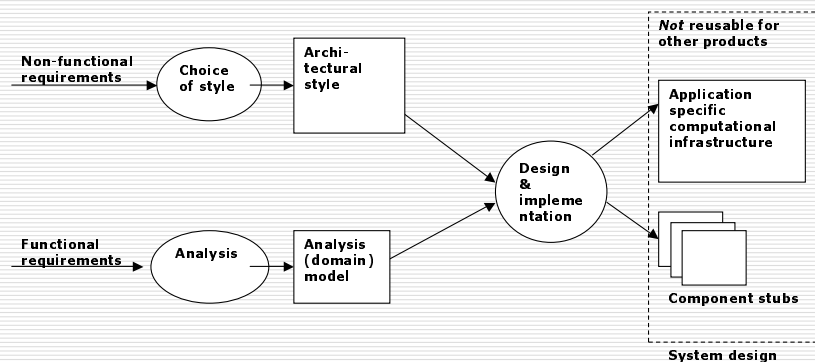
16/40



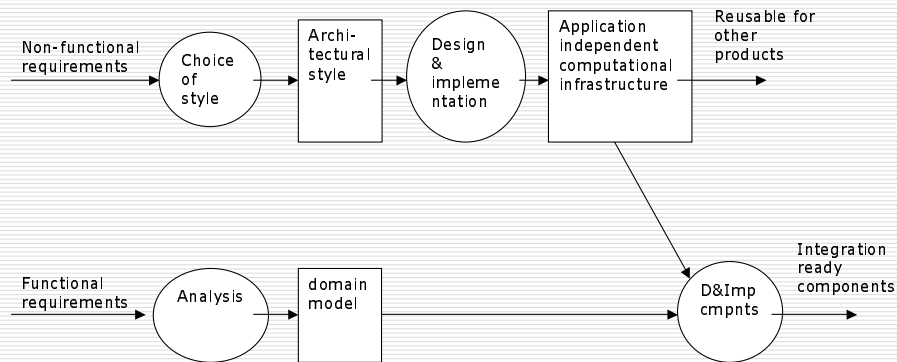
## Packaging Issues

- Un'architettura sw è uno specifico artefatto nel senso che il suo ciclo di vita è correlato ai cicli di vita di tutti i componenti del sistema
- Si trae vantaggio dalla separazione dell'artefatto e del suo ciclo di vita dagli altri artefatti e rispettivi cicli di vita

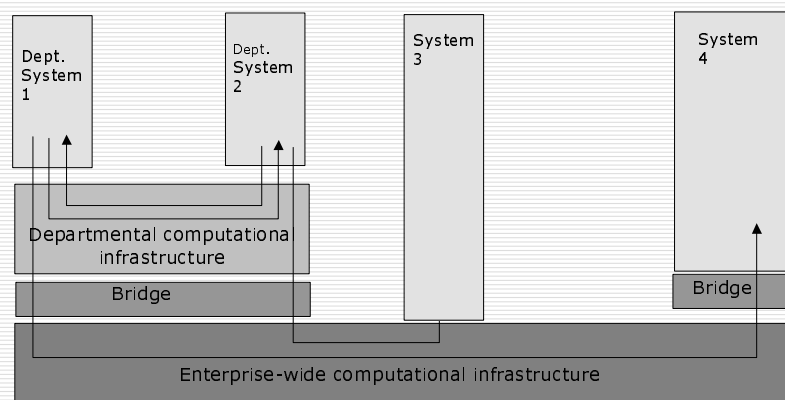
## Binding of domain aspects with computational aspects



## Separation of domain aspects from computational aspects



## Architetture eterogenee



## Processo di sviluppo

---

- Il primo passo nella costruzione di un framework è identificarne i requisiti funzionali
- Sui successivi passi vi sono due scuole di pensiero:
  - **Analisi di tipo top down:** sviluppiamo un framework ricorrendo ad un processo di domain engineering , partendo da una domain analysis, quindi design, e implementazione
  - **Analisi di tipo bottom up:** si parte sviluppando un'applicazione nel dominio del framework, e successivamente introducendo variabilità in essa

## Domain Engineering

---

- La Domain engineering è un set di attività volte a sviluppare artefatti riusabili in un reale dominio funzionale.
  - Attraverso un'analisi per identificare aspetti che sono comuni a tutte le applicazioni nel dominio e quegli aspetti che differenziano un'applicazione dall'altra
  - Derivando in modo incrementale descrizioni concrete descriptions per quegli aspetti/concetti/componenti, partendo da descrizioni a livello di analisi per arrivare al codice.

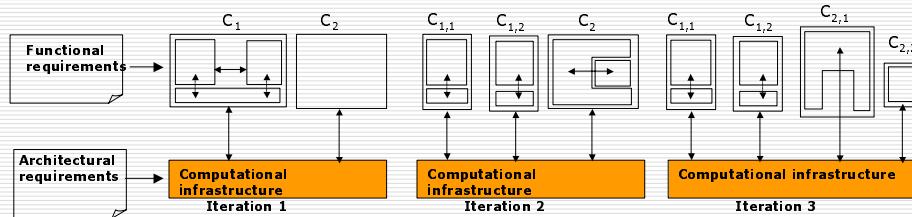
## Infrastruttura computazionale

- Un'applicazione mission-critical su web richiede:
  - **Distribuzione:** la logica applicativa dovrebbe essere "*location transparent*", ciò significa utilizzare un'infrastruttura distribuita,
  - **Sicurezza:** la comunicazione attraverso la rete deve essere sicura, ciò richiede gestione degli utenti, autorizzazioni, certificazioni, encrypting, etc.,
  - **Transaction services:** ciò richiede l'utilizzo di transaction monitors per transazioni distribuite
  - **Recoverability:** ciò significa usare distributed logging, mirror sites, etc...

## Hot-spots variability

- Il processo inizia con l'identificazione del dominio del framework, e i requirements per una prima applicazione.
- Tale implementazione si evolve in una prima iterazione del framework in cui un primo punto di variabilità (hotspot) viene sviluppato completamente.
- Progettare un hotspot significa cambiare un particolare legame di un punto di variabilità in:
  - Una descrizione astratta (generalizzazione) del particolare aspetto, e
  - Un numero di realizzazioni complete (compresa quella esistente).
- In un tipico incremento, una classe concreta all'iterazione  $n$  è sostituita all'iterazione  $n+1$  da una classe astratta o un'interfaccia e classi concrete.

# Framework Life Cycle



L'infrastruttura computazionale è sviluppata nella "prima iterazione", e segue l'approccio top down, partendo dai requirements, scegliendo uno stile architetturale (analysis), selezionando i relativi connettori (design), implementando l'infrastruttura. Oltre all'infrastruttura computazionale, si sviluppano anche componenti applicativi grezzi, e inizialmente con pochi gradi di libertà, e che sono raffinati via via con le successive iterazioni. In figura vediamo il componente ( $C_1$ ) che ha una sua struttura interna e i suoi meccanismi di interazione. Nella seconda iterazione, gli elementi interni del componente sono esternalizzati, e i suoi sottocomponenti ( $C_{1,1}$  e  $C_{1,2}$ ) ora interagiscono attraverso la comune infrastruttura computazionale. Per esempio,  $C_1$  potrebbe rappresentare un sistema legacy, con una sua architettura locale. Nella prima iterazione, un bridge API è per connettere il sistema all'infrastruttura comune. Le successive iterazioni esternalizzeranno i sottocomponenti del sistema legacy, fornendo maggiori gradi di libertà, e supportando l'intercambiabilità dei componenti.

## Selezionare un Enterprise Framework

- ❑ Lo sviluppatore dovrebbe analizzare le caratteristiche dei frameworks esistenti sulla base dei requirements dell'applicazione.
- ❑ Se non si individua un framework soddisfacente, possibili soluzioni sono:
  - I requisiti dell'applicazione sono rivisti sulla base delle caratteristiche dei frameworks disponibili. In accordo a Boehm [Boehm, 1981], "in the old process, system requirements drove capabilities. In the new process, capabilities will drive system requirements... it is not a requirement if you can't afford it".
  - Il miglior framework tra quelli esistenti è sicuramente adatto a soddisfare i requisiti dell'applicazione.
  - L'applicazione è sviluppata da zero.

## L'adeguatezza di un framework

- Tipiche domande sono:
  - Che evoluzione avrà l'applicazione?
  - Quali parti dell'applicazione sono le più stabili?
  - Quali standard sono disponibili?

## Framework customization

- *Livello infrastruttura di sistema → Black-box customization*
  - Consiste nell'aggregazione dei componenti elementari (ad ex. meccanismi di comunicazione logica e concorrenza).
- *Livello dominio di business → White-box customization*
  - I componenti base sono classi intermedie, che per loro natura sono abbastanza indipendenti dall'applicazione e devono essere adattate per ogni specifica applicazione
- *Livello applicativo → Grey-box customization*
  - Consiste nell'interconnessione componenti "pluggabili" attraverso lo starto middle ware. Questi componenti sono il risultato dell'adozione del framework per lo sviluppo di più applicazioni e in alcuni casi sono *commercial components-off-the-shelf (COTS)*

## Caratteristiche, Sfide e Criteri

- ❑ Enduring Business Themes (EBTs)
- ❑ Raggiungimento della stabilità software
- ❑ Supporto alla separazione dei concetti
- ❑ Funzionalità run-time mature
- ❑ Un esempio di workflow management ed Enduring Business Processes (EBPs)
- ❑ Supporto all'integrazione
- ❑ Indipendenza dalla piattaforma
- ❑ Ottima documentazione del framework
- ❑ Framework adequacies

## Enduring Business Themes

- ❑ L' EBT è concepito come una caratteristica del business. Letteralmente si intende "**che cosa** si sta facendo" e non "**come** si sta facendo". Ad esempio, nell'industria di trasporto, l'EBT è considerato lo spostamento di materiale da un posto ad un altro.
- ❑ La maniera migliore per identificare gli EBT è osservare l'organizzazione dal **punto di vista del cliente**. Il cliente fa più attenzione a **cosa** viene effettuato piuttosto che a **come** viene effettuato.
- ❑ Quindi, gli EBT sono legati allo scopo del business e gli oggetti di business sono delle istanze che specificano l'attività del business stesso in un certo periodo discreto di tempo.

## Stabilità del Software

- ❑ I **difetti** e il **deterioramento** di un software sono causati dalle modifiche apportate al software stesso.
- ❑ Una caratteristica richiesta ad un enterprise framework è la **stabilità a lungo termine**.
- ❑ E' necessario concentrare la fase di progettazione di un software, su quelle parti che dovrebbero rimanere stabili.
- ❑ "Enduring Business Themes" (EBTs) e "Enduring Business Processes" forniscono una base per la stabilità del software.

## Separazione dei Concetti

- ❑ Il paradigma Object-Oriented funziona bene soltanto se il problema può essere descritto con un'interfaccia relativamente semplice fra oggetti.
- ❑ La complessità più importante risiede nel fatto che i sistemi concorrenti/distribuiti hanno più di una dimensione.
  - Caratteristiche come scheduling, sincronizzazione, fault tolerance, sicurezza, testing, verifica e validazione sono sempre espresse in maniera tale da coinvolgere più oggetti.
  - Quindi, una semplice interfaccia ad oggetti viene violata e si perdono i benefici tradizionali dell'object-oriented.
- ❑ Un tentativo per risolvere il problema è l'architettura aspect-oriented. Le **architetture Aspect-oriented** sono sviluppate per realizzare l'architettura multi dimensionale di qualsiasi framework. Per questo, dobbiamo distinguere fra componenti e aspetti (funzioni).



## **Funzionalità Run-Time Mature**

- The framework life span
  - Architettura formata da elementi di base indipendenti da qualsiasi applicazione enterprise specifica
  - Attraverso il suo utilizzo in un numero crescente di applicazioni, il framework matura con la realizzazione di componenti sempre più concreti, i quali permettono la progettazione e l'implementazione di soluzioni per problemi sempre più difficili.
  - Gli oggetti di alto livello, che rappresentano le astrazioni principali identificate nel dominio del problema, possono maturare insieme al framework

## **Workflow Management**

- Il Workflow Management gestisce le **interazioni complesse fra oggetti** che appartengono ad applicazioni object-oriented di grosse dimensioni.
- Gli **Enduring Business Processes** catturano i workflow stabili nel tempo. Sono concepiti come una sequenza ad alto livello di ciò che è riportato nello state diagram di alto livello.
- Lo stesso concetto di workflow fornisce idee per rappresentare **processi di business dettagliati e dinamici**, visti come una sequenza di attività utili al completamento di operazioni abituali.

## Supporto all'integrazione (1)

---

### □ Principio di Hollywood

- Riguarda il concetto di inversione del controllo ("Non chiamarci, ti chiameremo noi!"). Un ciclo di controllo del framework invoca il codice dell'applicazione o un metodo al momento opportuno.

### □ Principio di Overlapping

- Questo problema si presenta quando due o più framework hanno gli stessi componenti concreti ma con differenti rappresentazioni e servizi

## Supporto all'integrazione (2)

---

### □ Principio del Gap

- Questo problema si presenta quando due o più frameworks, e l'architettura risultante, non realizzano tutti i requisiti richiesti all'applicazione. Questo è uno dei framework gap.

### □ Principio di impedenza

- Questo problema si presenta quando due o più framework integrati con differenti architetture falliscono nella loro interazione. Ciò è dovuto al fatto che più componenti del framework assumono molteplici forme che non possono efficacemente essere condivise o trasmesse ad altro framework

## Indipendenza dalla piattaforma

---

- ❑ L'indipendenza dalla piattaforma e la portabilità assicurano che il framework supporti tutte le piattaforme utilizzate da un'applicazione e garantisce l'interoperabilità tra le applicazioni costruite con il framework.
- ❑ L'indipendenza dalla piattaforma è specialmente importante per i venditori di framework; se un framework non è "open" sarà certamente una mancata vendita se il cliente non vorrà essere limitato nella scelta della piattaforma.

## Documentazione del Framework

---

- ❑ I fornitori di framework devono sicuramente fornire una documentazione esaustiva dei loro prodotti. La documentazione del framework deve descrivere il suo **scopo**, il suo **utilizzo** e la sua **struttura dettagliata**.
- ❑ I Pattern sono una tecnica documentata di progettazione che hanno la caratteristica di spiegare i motivi di una certa decisione progettuale e non solo il risultato.

## Documentazione del Framework(2)

---

- Un **pattern language** diventa un'effettiva documentazione tecnica dal momento in cui illustra i molteplici aspetti di un framework:
  - Il dominio applicativo per cui il framework è stato progettato; questo serve per scegliere il miglior framework sulla base di specifiche esigenze applicative.
  - La struttura del framework in termini di oggetti e loro correlazioni; questo aiuta il progettista nell'estensione del framework.
  - Le specifiche e l'implementazione delle classi del framework; questo fornisce il punto di partenza per l'implementazione di reali applicazioni.

## Adeguatezza del Framework

---

- L'adeguatezza del Framework è legata alla bontà della copertura delle esigenze
- Un framework non dev'essere né troppo generalizzato né troppo specifico
- Un buon framework per un particolare dominio applicativo riflette un maturo ed approfondito studio di un particolare problema o gruppo di problemi.