Model-Driven Architecture (MDA)

Andrea Baruzzo
e-mail: baruzzo@dimi.uniud.it

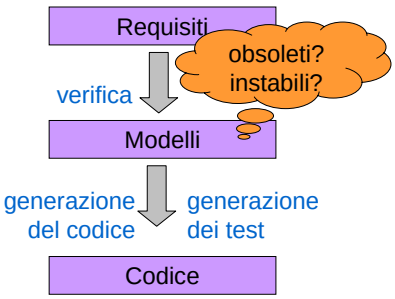
Dipartimento di Matematica e Informatica
Università degli Studi di Udine

Agenda

- **Model-Driven Development (MDD)**
- Model-Driven Architecture (MDA)
- Verifica di modelli UML
- Modelli eseguibili
- Argomenti di tesi

Model-Driven Development

- Un concetto base (modello come deliverable), ma diverse **interpretazioni**
- Scrittura del codice
  - manuale vs. automatica
- Elaborazione del **modello**
  - manuale vs. automatica
- **Verifica** di requisiti e specifiche:
  - a livello di codice (asserzioni, test)
  - a livello di modello (inspection informali, esperti)
  - entrambe
  - nessuna



L'approccio automatico tradizionale

- Ogni modifica nel codice viene aggiornata automaticamente nel diagramma
- Risolve **sincronizzazione** codice-modello
- Diagrammi aggiornati ma ricchi di dettaglio
- I dettagli **nascondono** gli aspetti essenziali di progetto
- I **cambiamenti** sono nei dettagli!
- Ancora instabilità!

Automazione nella direzione sbagliata …

- … infatti:
  - Nessun **controllo** sulla **qualità** del modello
  - Modelli difficili da leggere, da capire (troppi dettagli di implementazione)
  - Dominio e Business Logic **oscurati**
  - Modelli che sono solamente una **vista sintattica alternativa** del codice, non più uno strumento di progettazione!

5

Automazione in una direzione diversa …

- L'automazione dovrebbe aiutarci a:
  - mitigare instabilità e obsolescenza dei modelli
  - mantenere **separazione** tra i **livelli di astrazione** (dominio, tecnologia, codice)
  - fornire **strumenti di verifica** del modello
- Automazione verso i concetti di:
  - **Trasformazione** di modelli (per la separazione)
  - **Esecuzione** di modelli (per la verifica)

6

Agenda

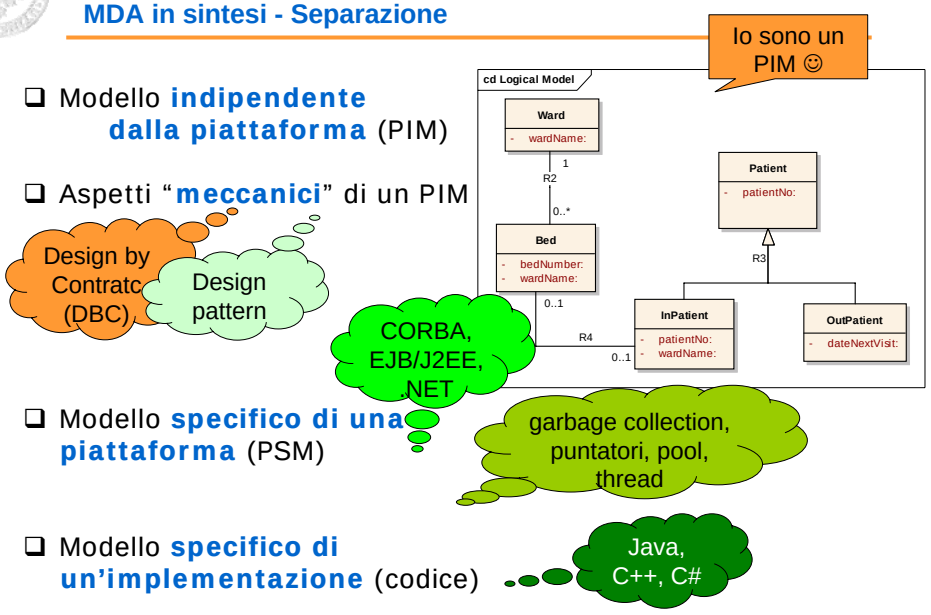
- Model-Driven Development (MDD)
- **Model-Driven Architecture (MDA)**
- Verifica di modelli UML
- Modelli eseguibili
- Argomenti di tesi

Che cos'è Model-Driven Architecture?

- Non è un'architettura software, ma …
- … un'architettura del processo di sviluppo del software
- È **standardizzata** dall'OMG
- È basata sui **modelli** (UML)
- È basata sulla **separazione in livelli di astrazione**
- È basata sulle **trasformazioni** di modello
- Il codice non è più un deliverable indispensabile
- Vediamo di capire meglio …

MDA in sintesi - Separazione

- Modello **indipendente dalla piattaforma** (PIM)
- Aspetti "**meccanici**" di un PIM
- Modello **specifico di una piattaforma** (PSM)
- Modello **specifico di un'implementazione** (codice)



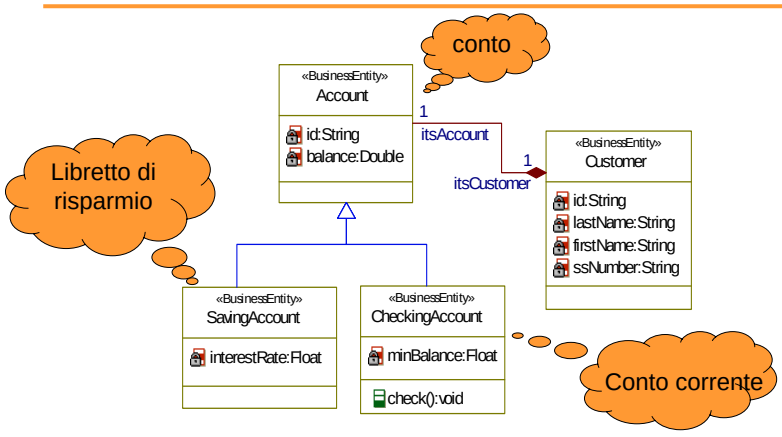
MDA in sintesi – Trasformazioni automatiche

- **Linguaggio di specifica** per
  - azioni (dinamica, transizioni di stato, corpo metodi)
  - vincoli (asserzioni, precondizioni, postcondizioni, invarianti)
- **Regole di traduzione**
  - **mapping** tra modelli (PIM-PSM-Codice)
- Diversi tipi di trasformazione
  - Da modello a modello
    - PIM->PIM (design pattern, DBC,…)
    - PIM->PSM
    - PSM->PSM
  - Da modello a codice
    - PIM->Codice
    - PSM->Codice

Esempio di trasformazione PIM->PIM: Replicazione automatica di design pattern 1/3

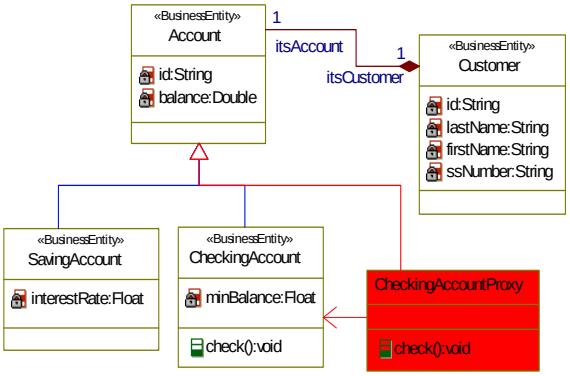
- Un generatore MDA può incapsulare conoscenza sulla struttura e l'applicazione dei design pattern
- Design pattern visti come **componenti**
- La gran parte del codice di un pattern è "meccanico" perché:
  - Implementa una **struttura costante** e nota a priori
  - Realizza **interazioni stereotipate** (c'è struttura anche nella dinamica)
- Questa parte di modello può essere generata automaticamente (almeno in parte)…

Esempio di trasformazione PIM->PIM: Replicazione automatica di design pattern 2/3



- Generazione del pattern Proxy per la classe CheckingAccount …

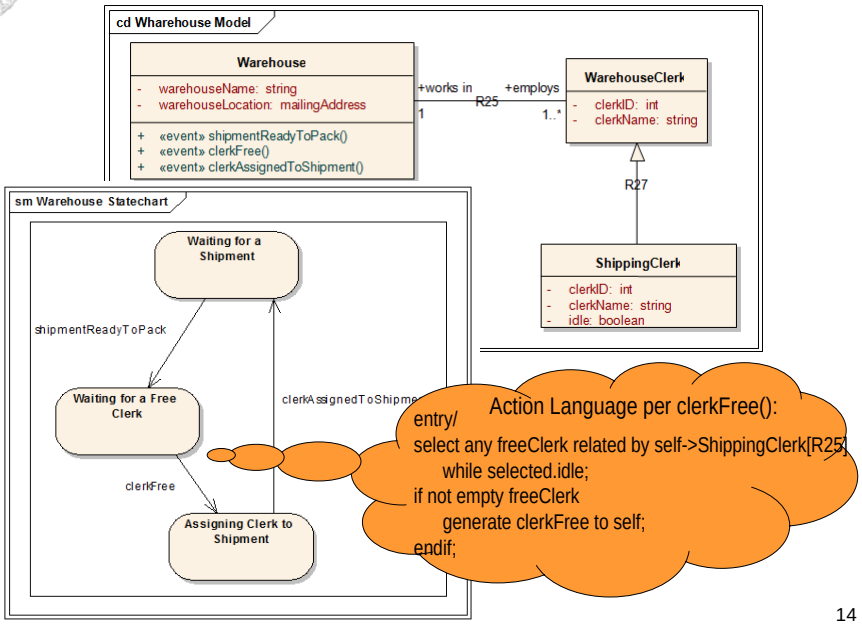
Esempio di trasformazione PIM->PIM: Replicazione automatica di design pattern 3/3



- Libera il programmatore dai dettagli meccanici della replicazione
- Trasformazione successiva: generare il codice (PIM->Codice)

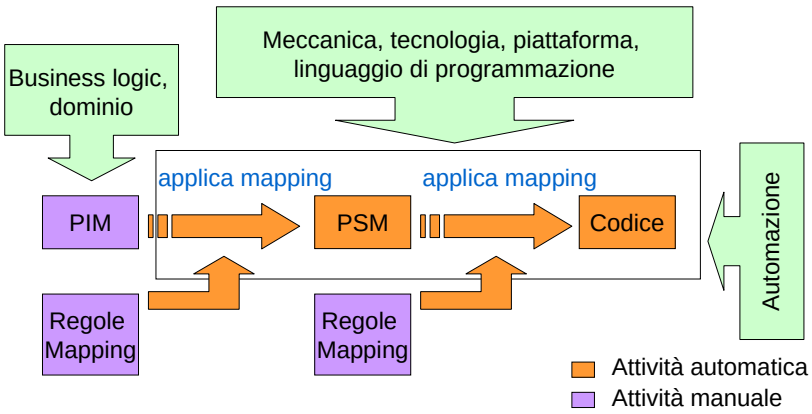
13

Esempio di PIM pronto ad essere tradotto in Codice

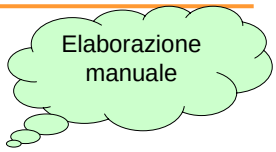


14

Automazione in MDA

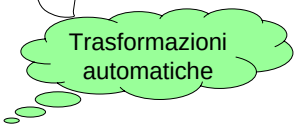


Diversi linguaggi per diversi aspetti del sistema



- Modello UML (diagrammi grafici)
- Azioni, dinamica (action language)
- Asserzioni, precondizioni, postcondizioni, invarianti (Eiffel, oppure linguaggi ad hoc come OCL)

- Meccanica DBC, eccezioni, pattern
- Codice applicazione (Java, C++, C#, VB, …)
- Rappresentazione schemi di dati (XML)
- Mapping con un database relazionale (Jdbc)



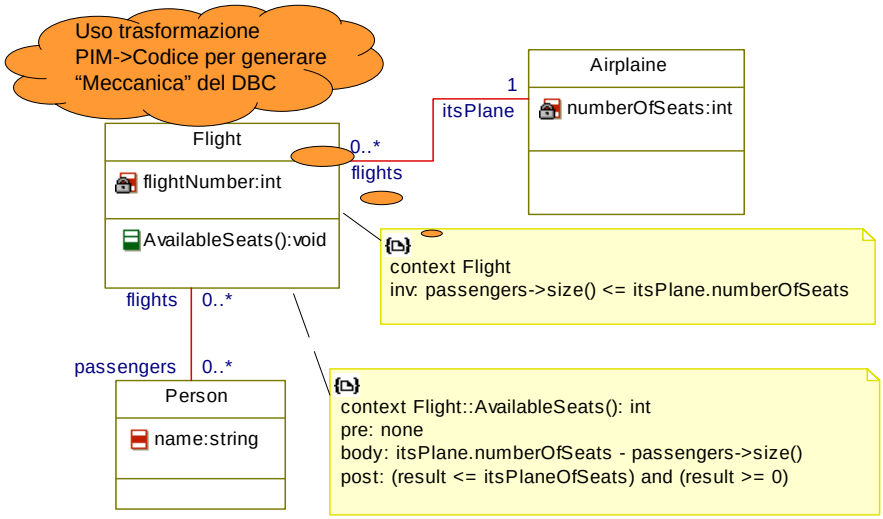
Agenda

- Model-Driven Development (MDD)
- Model-Driven Architecture (MDA)
- **Verifica di modelli UML**
- Modelli eseguibili
- Argomenti di tesi

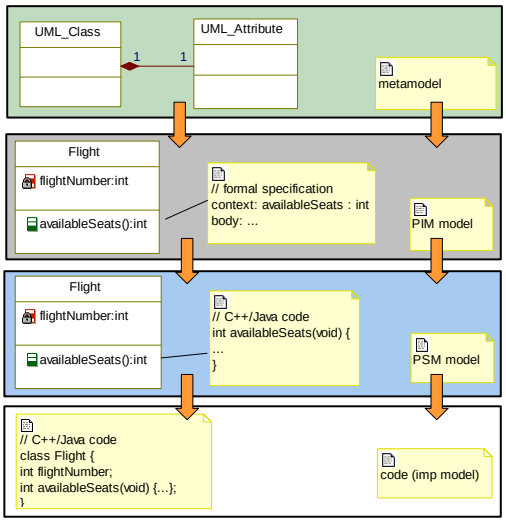
Verifica di modelli UML

- UML + OCL: **specifiche formali** nei modelli
- Un generatore MDA può generare automaticamente il codice per:
  - Verifica degli **invarianti**
  - Verifica delle **precondizioni** e delle **postcondizioni**
  - Verifica delle **asserzioni**
  - Gestione delle **eccezioni**
- Trasformazioni MDA per generare il codice di test

Modello UML e specifiche formali



MDA: una vista d'insieme



- Tool di analisi di modello
- Correttezza sintattica modello

- Progettista – programmatore
- Definisce modello concettuale
- Specifica azioni indipendente dalla piattaforma

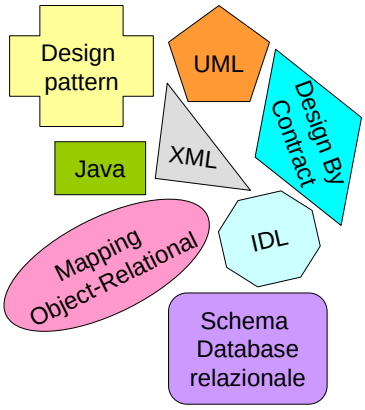
- Tool di trasformazione MDA
- Intervento sporadico programmatore

- Tool di trasformazione MDA
- Intervento sporadico programmatore

20

MDA: integrazione automatica di tecnologie

- Diversi linguaggi per descrivere diversi aspetti di un componente (EJB, modello ad oggetti-modello relazionale,…)
- Il problema vero è la mancanza di un'**architettura generale** per integrare le diverse specifiche
- MDA suggerisce un framework per l'**integrazione automatica** di tecnologie diverse …
- … basta disporre della giusta trasformazione! Serve supporto tool!



Agenda

- Model-Driven Development (MDD)
- Model-Driven Architecture (MDA)
- Verifica di modelli UML
- **Modelli eseguibili**
- Argomenti di tesi

Modelli eseguibili

- Le tecniche di **manipolazione automatica** di modelli UML richiedono:

- **semantica operazionale UML**
  - ***Precisa*** (sufficientemente dettagliata da poter essere elaborata dai tool per generare codice)
  - ***Non ambigua*** (differenti tool devono generare lo stesso codice in corrispondenza delle stesse primitive)

- Definizione di **profili UML**
  - Sottoinsieme stabile di primitive e diagrammi, standardizzati alla versione UML 2.0, con una semantica ben definita.

- Uno di questi profili è: **eXecutable UML (xUML)**

23

Il profilo xUML in sintesi

- **Trasformazione automatica modello-codice**

- **Struttura dei modelli semplificata che comprende:**
  - *Diagrammi di classe* per definire la struttura del sistema
  - *Diagrammi di stato* + specifica formale *azioni* per la dinamica

- Altri diagrammi come supporto alla documentazione o al test (sequence diagram), ma non ancora direttamente eseguibili

- **Semantica precisa per le azioni**

- **Linguaggio di specifica per le azioni** (action language)

xUML = UML - Semantically Weak Elements + Precise defined Action semantics

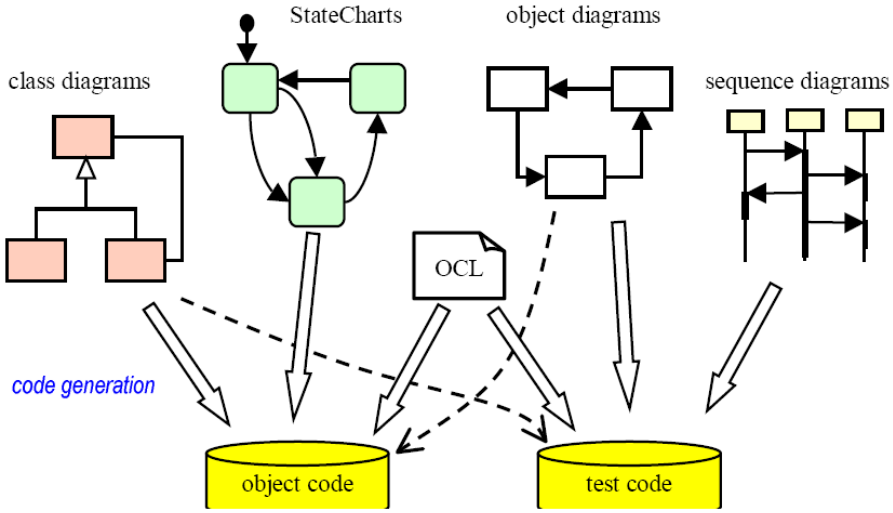
24

Input e output in Executable UML

- Executable UML
  - Input:
    - modello astratto di un programma
      - (PIM, specifiche formali in OCL, action language)
    - compilatore di modello specifico per la piattaforma
  - Output:
    - Applicazione eseguibile
- Executable UML è una possibile implementazione del processo di sviluppo MDA
- Non molti tool per la modellazione oggi supportano completamente xUML

25

Executable UML in breve



26

Agenda

- Model-Driven Development (MDD)
- Model-Driven Architecture (MDA)
- Verifica di modelli UML
- Modelli eseguibili
- **Argomenti di tesi**

27

Argomenti di tesi

- Metodi e strumenti di validazione e verifica assistita di modelli UML (debugging/test)
- Metodi e strumenti per la certificazione dei modelli UML (verifica formale statica, proof engine, model checker,…)
- Sviluppo di metriche per l'analisi della qualità dei modelli UML
- Definizione di profili UML
- Tecniche automatiche di layout dei modelli UML
- Metodi e strumenti software per la trasformazione di modello (basati su UMT-QVT, Eclipse,…)
  - Ad esempio, il mapping tra action language e Java/C++

28

Grazie per l’attenzione!

29