UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II
Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

tesi di laurea

Metodologie per la generazione automatica di codice: eXtreme Non Programming

Anno Accademico 2005/2006

relatore
Ch.mo prof. Porfirio Tramontana

candidato
Paolo Pellecchia
Matr. 831/119

Outline – Uno sguardo d'insieme

- Esplorare una nuova prospettiva di sviluppo software: XNP
- Definire i passi essenziali del nuovo processo
- Focalizzare l'attenzione sulla trasformazione dei modelli pensati dall'uomo in quelli comprensibili dalla macchina
- Proporre una soluzione basata su XML
- Fornire un'applicazione pratica delle metodologie proposte

Generazione automatica di codice

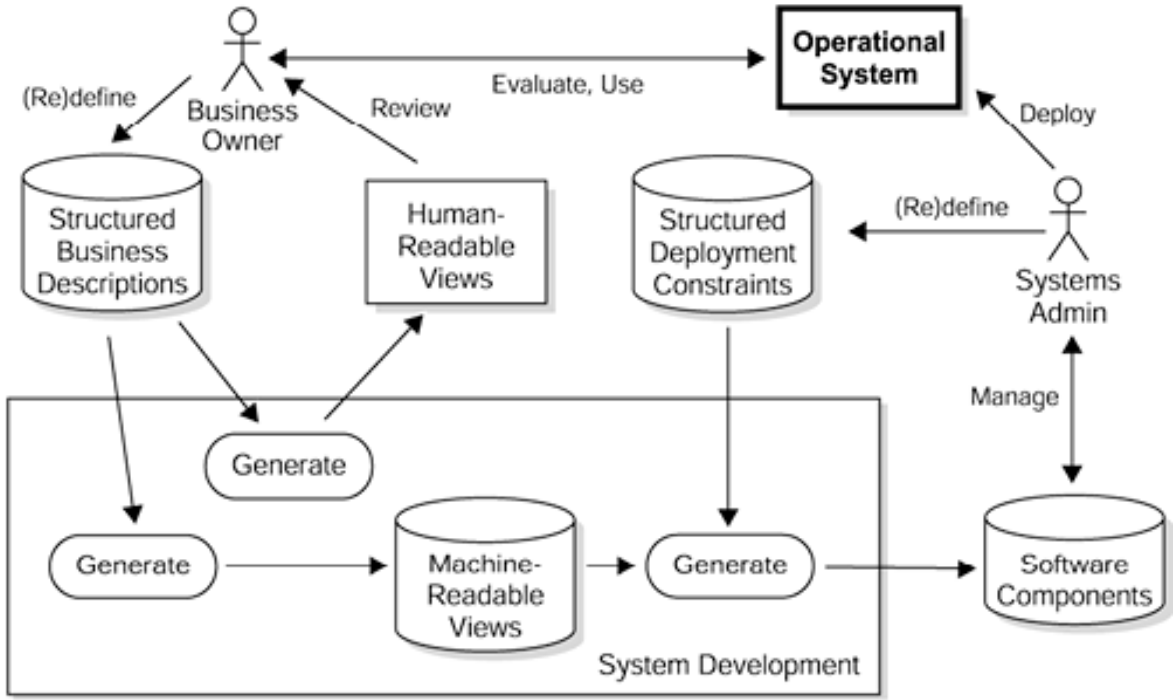
- **Motivi**
  - Ottenere un prodotto software qualitativamente elevato
  - Adeguare i tempi di sviluppo ai continui cambiamenti del mercato
  - Ridurre i costi e gli errori umani dovuti alla programmazione
- **Proposte**
  - Spostare l'attenzione sulla modellazione
  - Instaurare un rapporto 1:1 tra modello e codice
  - realizzare un *Compilatore di Modello* che effettui un mapping tra un *modello concettuale* e la sua rappresentazione software
- **Soluzioni**
  - eXtreme Non Programming (XNP)
  - Conceptual Schema – Centric Development (CSCD)
  - Model-Driven Engineering (MDE)
  - Generative Programming

eXtreme Non Programming – XNP

- Descrivere completamente il sistema in maniera strutturata
  - Business Rules
  - Rule Engine
- Generare una vista riassuntiva del modello
  - Human-Readable views
  - Contratto con l'utente
- Garantire l'equivalenza del modello pensato dall'uomo e quello inteso dalla macchina
  - Machine-Readable views
- Separare il modello dalla piattaforma
  - Parser di modello specifico per ciascun linguaggio/piattaforma
  - Semplifica il passaggio a nuove tecnologie

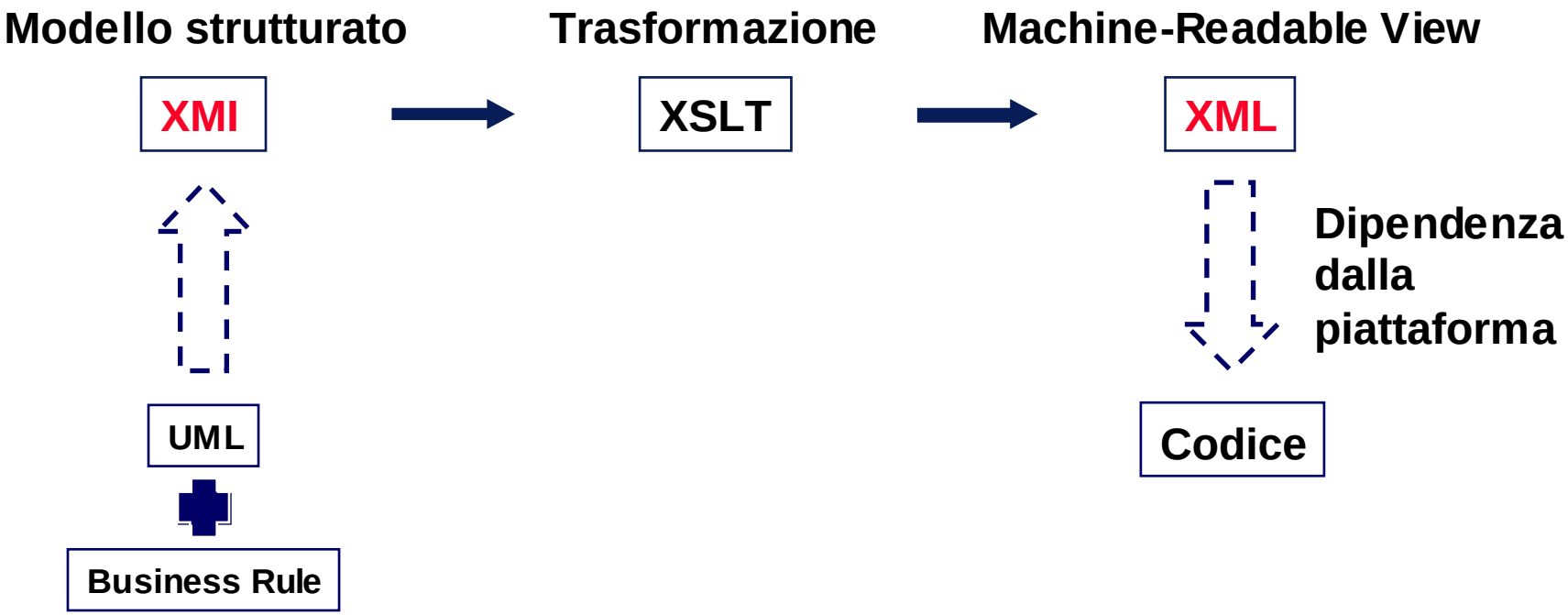
Processo di generazione del codice

- **Tre passi fondamentali**
  - **Descrizione Strutturata del Business**
  - **Generazione delle viste intermedie**
  - **Produzione del codice**



Trasformazione di modelli

- Definire un formato per rappresentare la descrizione strutturata del Business
- Definire un formato che rappresenti la Machine-Readable view
- Trasformare il primo formato nel secondo



Linguaggi e strumenti

- **XML**
  - Interoperabilità
  - Polivalenza
  - Portabilità
  - Struttura gerarchica

```
<persona>
    <nome>
        <primoNome>Mario</primoNome>
        <secondoNome>Francesco</secondoNome>
        <cognome>Rossi</cognome>
    </nome>
<persona>
```

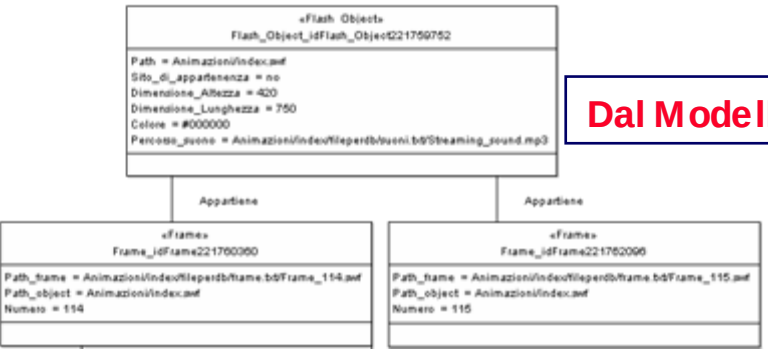
- **XMI**
  - Formato di interscambio diffuso
  - Basato su XML
  - Indipendente dal linguaggio di modellazione adottato

```
<UML:Class xmi.id="id" name="nomeClasse"
visibility="public" isRoot='' isLeaf='' isAbstract=''>
    <UML:Classifier.feature>
        <UML:Attribute xmi.id="" name="" visibility=""/>
        ...
```

- **XSLT**
  - Dedicato alle trasformazioni tra documenti
  - Basato su XML
  - XSLT engine
  - Costrutti simili a quelli dei linguaggi procedurali

```
<xsl:template match="/*">
    <dataroot>
        <xsl:for-each select="*">
            <xsl:call-template name="primaAssociazione">
                <xsl:with-param name="i" select="@id"/>
            </xsl:call-template>
        </xsl:for-each>
```

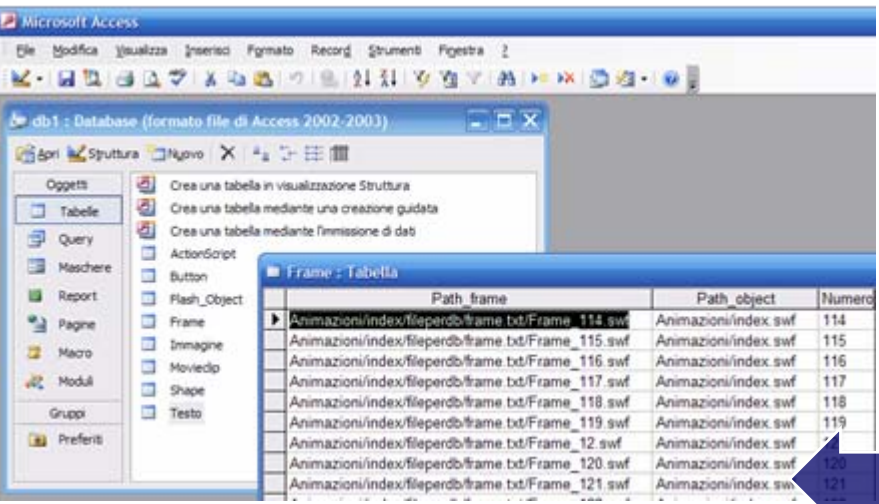
Un esempio concreto: Flash Application



Dal Modello all'XMI

```
<UML:Class xmlns:UML="" xmi.id="idActionScript221761088" name=
"ActionScript_idActionScript221761088" visibility="public"
isSpecification="false" namespace="M.1" isRoot="true" isLeaf="true"
isAbstract="false" isActive="false">
    <UML:ModelElement.stereotype>
        <UML:Stereotype xmi.idref=
        "ActionScript_Stereotipo_idFlash_Object221759752"/>
        </UML:ModelElement.stereotype>
    <UML:Classifier.feature>
        <UML:Attribute xmi.id="idPath_Action221761144" name=
        "Path_Action" visibility="private" isSpecification="false"
        ownerScope="instance">
            <UML:Attribute.initialValue>
                <UML:Expression xmi.id=
                "value_idPath_Action221761144" body=
            "Animazioni/index/fileperdb/script.txt/Sprite_114.as"/>
            </UML:Attribute.initialValue>
        </UML:Attribute>
```

Da XMI alla vista intermedia in XML



Da XML al database in Access

```
<Flash_Object>
    <Path>Animazioni/prova.swf</Path>
    <Sito_di_appartenenza>no</Sito_di_appartenenza>
    <Dimensione_Altezza>440</Dimensione_Altezza>
    <Dimensione_Lunghezza>750</Dimensione_Lunghezza>
    <Colore>#ff9933</Colore>
    <Frame>
        <Path_frame>Animazioni/prova/fileperdb/frame.txt/Frame_1.swf
        <Path_object>Animazioni/prova.swf</Path_object>
        <Numero>1</Numero>
        <ActionScript>
            <Path_Action>Animazioni/prova/fileperdb/script.txt/Frame
```

Codice XSLT

```
<!-- se il secondo estremo corrente non contiene nell'xmi.idref l'xmi.id passato come parmetro e non è l'ultimo-->
<xsl:if test="not(/.../Association[number($posizione)]/.../AssociationEnd[2]/.../@xmi.idref=$i)
and (boolean(/.../Association[number($posizione+1)]/.../Class))">
    <!-- riapplica lo stesso template al secondo estremo dell'associazione successiva -->
    <xsl:call-template name="primaAssociazione">
        <xsl:with-param name="i" select="$i"/>
        <xsl:with-param name="posizione" select="$posizione+1"/>
    </xsl:call-template>
</xsl:if>
```

Problema 1

Capire da quale classe far iniziare l'albero XML

Soluzione:

Cercare le classi che non compaiono mai come secondo estremo nelle associazioni (Convenzione)

Problema 2:

Scegliere a quale tabella apparterrà la classe

Soluzione:

Trovare lo stereotipo della classe da inserire

```
<!--cerca e stampa la classe individuata da $i-->
<xsl:for-each select="/XMI/XMI.content/UML:Model/UML:Namespace.ownedElement/Class">
    <!--se la classe corrente ha lo stesso xmi.id contenuto in $i-->
    <xsl:if test="@xmi.id=$i">
        <!--Salva il valore dell'xmi.idref dello stereotipo a cui fa riferimnto la classe corrente in una variabile-->
        <xsl:variable name="riferimento" select="./ModelElement.stereotype/Stereotype/@xmi.idref"/>
        <!--controlla per ogni stereotipo quale possiede l'xmi.id salvato-->
        <xsl:for-each select="../Stereotype">
            <xsl:if test="$riferimento=@xmi.id">
                <!--Se lo trova crea un tag che ha per nome il nome stesso dello stereotipo-->
                <xsl:text disable-output-escaping="yes">&lt;</xsl:text><xsl:value-of select="@name"/>
        </xsl:text>
            </xsl:if>
        </xsl:for-each>
```

Problema 3:

Innestare correttamente le classi (relazioni DB)

Soluzione:

Riapplicare il template alle classi che compaiono come secondo estremo delle associazioni che hanno come primo estremo la classe corrente

```
<!--controlla per ogni associazione se il primo estremo è la classe $i -->
<xsl:for-each select="/.../AssociationEnd[1]/.../Class">
    <xsl:if test="@xmi.idref=$i">
        <!--cerca le associazioni che vedono la classe del secondo estremo come primo estremo in modo da innestarle nella prima-->
        <xsl:call-template name="cercaAssociazione">
            <xsl:with-param name="i" select="../../../AssociationEnd[2]/.../@xmi.idref"/>
        </xsl:call-template>
    </xsl:if>
</xsl:for-each>
```

Conclusioni e sviluppi futuri

- **Fattibilità**
  - Indipendenza dall'applicazione modellata
  - Indipendenza dai contenuti del database
  - Naturale trasformazione tra formati omogenei
- **Limiti**
  - Semplicità XSLT
  - Importazione XML
- **L'altro lato della medaglia**
  - Fino a che punto sono importanti i modelli?
  - Le piattaforme cambiano così frequentemente?
  - I clienti compreranno prodotti sviluppati in questo modo?
  - Occorre modificare radicalmente il proprio processo di sviluppo
  - Difficoltà nell'esprimere il comportamento in UML
- **Sviluppi futuri**
  - Raffinare l'algoritmo
  - Ripristinare relazioni in Access (VB Script)
  - Wizard per creare il modello
  - Definire delle viste intermedie valide per XNP