**Università degli studi di Salerno**

**Facoltà di Informatica**



**Dipartimento di Informatica**

**Corso di laurea magistrale in Informatica**

**Data science e machine learning**

**Studio di usabilità degli operatori iconici di CoDIT**

**Docente Candidati**

Francesca Tassatone 0522500568

Roberta Gesumaria 0522500569

Daniele Lupo

0522500579

Prof. Giuseppe Polese

Dott.ssa Loredana Caruccio

**Anno accademico 2018-2019**

Sommario

[Introduzione 3](#_Toc18674879)

[IBench 4](#_Toc18674880)

[Installazione 4](#_Toc18674881)

[Utilizzo 4](#_Toc18674882)

[CoDIT 5](#_Toc18674883)

[Struttura XMI riconosciuta da CoDIT 6](#_Toc18674884)

[Sezione iniziale 6](#_Toc18674885)

[Sezione entità e attributi 6](#_Toc18674886)

[Sezione relazioni 7](#_Toc18674887)

[Sezione finale 7](#_Toc18674888)

[Coordinate elementi 8](#_Toc18674889)

[Parser 9](#_Toc18674890)

[Sezione iniziale 9](#_Toc18674891)

[Sezione entità e attributi 9](#_Toc18674892)

[Sezione relazioni 10](#_Toc18674893)

[Sezione finale 11](#_Toc18674894)

[Studio di usabilità 12](#_Toc18674895)

# Introduzione

Il progetto svolto riguardava lo studio di usabilità sull’espressività e la complessità percepita degli operatori iconici di CoDIT utilizzando scenari che coinvolgano schemi di grosse dimensioni. Per generare tali schemi ci è stato indicato di utilizzare il tool iBench.

Vediamo nel dettaglio il lavoro da svolgere.

CoDIT è un tool per l’integrazione di schemi. Gli schemi su cui lavora sono schemi concettuali sui quali è possibile applicare operatori iconici per l’integrazione. I formati richiesti da CoDIT sono XMI o SVG.

Per lo studio richiesto, abbiamo però bisogno di schemi più grandi di quelli che si hanno a disposizione, per ottenere ciò è stato utilizzato IBench.

IBench genera schemi logici in formato XML basandosi su un file di configurazione.

Le prime differenze da notare sono le tipologie di file e di schemi differenti che si ottengono da un tool e che vengono elaborati dall’altro. Inoltre, i file XMI riconosciuti da CoDIT hanno una struttura non generica, quindi è stato necessario convertire lo schema, da logico a concettuale, e ottenere una strutturara adeguata in maniera da rendere il file comprensibile al tool.

Successivamente, è stato effettuato uno studio di usabilità sull’espressività e la complessità percepita degli operatori iconici di CoDIT, sia dai membri del gruppo che da altri utenti, sempre con competenze informatiche.

# IBench

iBench è un generatore di metadati per la creazione di mappature, schemi e vincoli di schemi arbitrariamente grandi e complessi. Può essere utilizzato con un generatore di dati per generare efficacemente scenari realistici di integrazione dei dati con diversi gradi di dimensione e complessità. Può essere utilizzato per creare benchmark per diverse attività di integrazione dei dati (virtuali), integrazione dei dati, scambio di dati, evoluzione degli schemi, mappatura degli operatori come composizione e inversione e corrispondenza degli schemi.

L'utente fornisce un file di configurazione che specifica quali tipi di metadati e dati generare (es. schemi, dati, vincoli, mappature, ...) e proprietà dello scenario generato. iBench produce un file XML che memorizza i metadati generati in base alla configurazione dell'utente. Se richiesto, genera anche i dati di istanza per gli schemi generati.

## Installazione

IBench è scritto in Java usando ant per il building e ivy per scaricare le dipendenze.

La prima operazione svolta è stata quella di scaricare e installare ant.

Eseguendo il comando ant nella cartella principale, è stato scaricato *ivy* ed utilizzato automaticamente per scaricare il jar delle dipendenze. Successivamente, verrà generata una cartella build in cui sono presenti vari script di shell (.sh per Linux e Mac e .bat per Windows).

## Utilizzo

# CoDIT

CoDIT (Conceptual Data Integration Tool), supporta il processo di integrazione dei dati a livello concettuale, permette la costruzione di uno schema globale correlando sottoschemi delle sorgenti, il DBA potrà specificare in maniera visuale come combinare le sorgenti di dati. Lo schema globale potrà essere specificato allineando i sottoschemi delle sorgenti oppure specificando nuovi costrutti e mappando i sottoschemi delle sorgenti ad essi.

A livello metodologico CoDIT si basa su un linguaggio visuale, CoDIL (Conceptual Data Integration Language).

CoDIL permette di gestire l’eterogeneità degli schemi, per far fronte alla necessità di risolvere i conflitti dovuti all’utilizzo di diversi costrutti e/o nomi per modellare la stessa realtà.

Attraverso CoDIL il DBA potrà selezionare costrutti degli schemi sorgenti (uno per ogni schema) in corrispondenza ed associare un operatore iconico per specificare come risolvere il conflitto.

Il linguaggio CoDIL fornisce un insieme di operatori iconici che possono essere applicati a coppie di sottoschemi concettuali. Attraverso l’utilizzo di tali operatori è possibile specificare in che modo i sottoschemi selezionati sono correlati e scegliere come questi devono essere mappati nello schema riconciliato.

Le funzionalità offerte da CoDIT sono:

1. Disegna schema;
2. Caricamento dinamico;
3. Create target scheme.



Figura : Home tool CoDIT.

L’operazione sulla quale ci siamo focalizzati è il caricamento dinamico. I formati riconosciuti da CoDIT sono XMI e SVG. Il file XMI, però, ha una struttura specifica che riporteremo di seguito.

## Struttura XMI riconosciuta da CoDIT

La struttura del file XMI può essere suddivisa in quattro parti:

* Una sezione iniziale;
* Una sezione in cui vengono indicate tutte le entità con i relativi attributi;
* Una sezione in cui vengono indicate tutte le relazioni;
* Una sezione finale.

### Sezione iniziale

La seguente sezione risulta essere identica in tutti i file forniti come esempi per un primo utilizzo di CoDIT.



Figura : Struttura XML CoDIT, sezione iniziale.

### Sezione entità e attributi

Per indicare l’entità viene utilizzato il tag "packagedElement", con i seguenti attributi:

* xmi:type = uml:Class, per indicare che si tratta di un’entità;
* xmi:id, per indicare l’id;
* name, per indicare il nome dell’entità;
* type, per indicare il tipo di entità. Non sempre viene specificato, di solito è usato per indicare le entità deboli (WeakEntity).



Figura : Struttura XML CoDIT, entità.

Per ogni entità, come figlio dell’elemento "packagedElement", viene indicato l’attributo “xmi:Extension” per specificare che si tratta di un’entità.



Figura : Struttura XML CoDIT, estensione identificativa dell'entità.

Per indicare gli attributi relativi all’entità viene utilizzato il tag "ownedAttribute” come figlio dell’elemento "packagedElement".

Per ogni attributo vengono definiti:

* xmi:type=”uml: Property, per indicare che è un attributo;
* xmi:id, per indicare l’id;
* name, per indicare il nome dell’attributo;
* visibility, per indicare la visibilità dell’attributo.



Figura : Struttura XML CoDIT, attributo.

Nel caso in cui l’attributo sia chiave primaria, viene utilizzato il tag “xmi:Extension”, come figlio di “ownedAttribute”, per indicare che si tratta di una chiave primaria.



Figura : Struttura XML CoDIT, attibuto chiave primaria.

### Sezione relazioni

In questa sezione vengono definite le relazioni tra le entità sopra definite.

Viene usato sempre il tag "packagedElement", questa volta però è utilizzato con lo scopo di definire le associazioni, ciò è specificato nell’attributo “xmi: type= uml:Association”.

La relazione tra le entità è indicata attraverso l’attributo “xmi:id” in cui è specificato il nome delle due entità relazionate, usando il seguente formato E1\_\_E2\_\_id.

Inoltre, se si deve definire una specializzazione, oltre ad indicare un id in cui ci sono i nomi delle entità, viene inserito anche l’attributo “type=ISA”.



Figura : Struttura XML CoDIT, sezione relazioni tra le entità.

### Sezione finale

La seguente sezione risulta essere identica in tutti i file forniti come esempi per un primo utilizzo di CoDIT.



Figura : Struttura XML CoDIT, sezione finale.

### Coordinate elementi

Un’ulteriore funzionalità offerta da CoDIT è quella di salvare la posizione degli elementi, memorizzando le relative coordinate all’interno del file XMI. Quindi, per ogni elemento identificativo di una entità, attributo o relazione, sono presenti gli attributi x e y.



Figura : Struttura XML CoDIT, entità e attributi con coordinate.

# Parser

Come specificato nel paragrafo introduttivo, iBench genera uno schema logico su file XML, mentre CoDIT lavora su schemi concettuali su file XMI con una struttura mostrata precedentemente. Per risolvere questo problema è stato creato un parser, che letto il file XML genera un nuovo file XMI con la struttura richiesta.

Per implementare il parser è stato utilizzato il linguaggio di programmazione Python, il quale offre una soluzione finalizzata proprio all’interfacciamento con i file XML. Infatti, grazie all’utilizzo del modulo xml.etree.ElementTree è stato possibile leggere il contenuto del file e creare una struttura per il nuovo file XMI da generare.

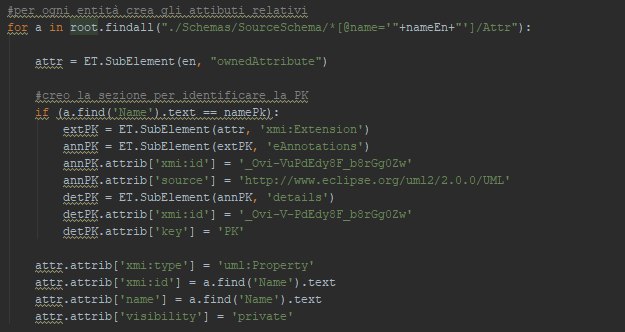
Di seguito mostreremo le varie parti del parser in relazione alla struttura del file XML mostrata nel capitolo “Struttura XMI riconosciuta da CoDIT”.

## Sezione iniziale

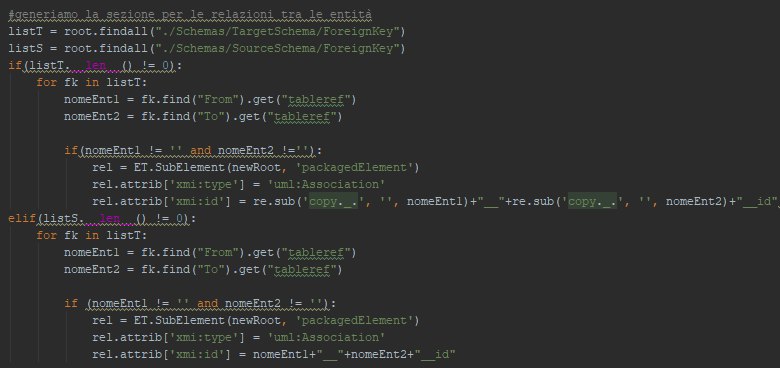


## Sezione entità e attributi





## Sezione relazioni



## Sezione finale



# Studio di usabilità