Designing ETL Processes Using Semantic Web Technologies

Dimitrios Skoutas Alkis Simitsis

{dskoutas,asimi}@dblab.ece.ntua.gr

National Technical University of Athens

Dept. of Electrical and Computer Engineering

http://www.dblab.ece.ntua.gr

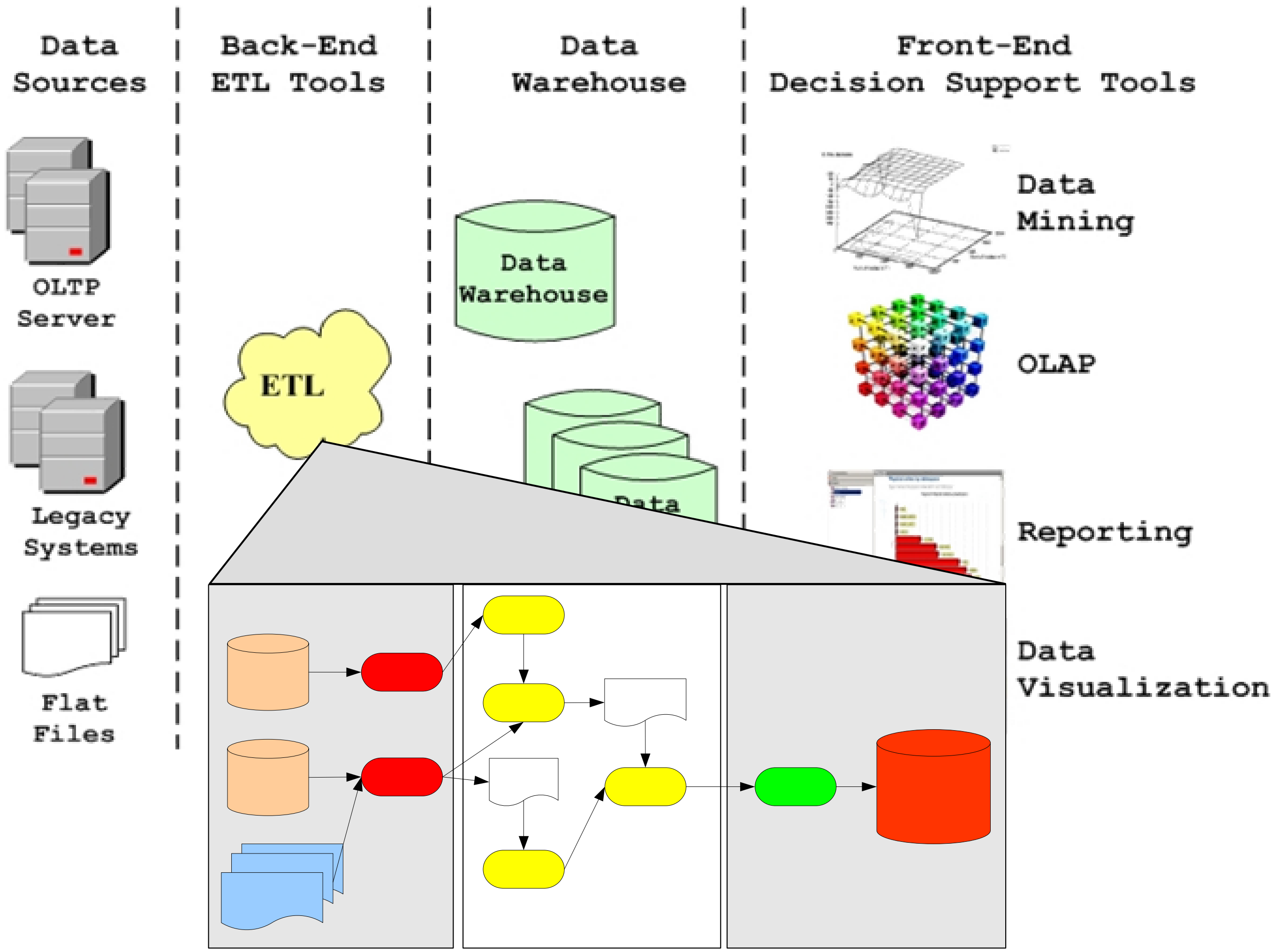
Outline

- Introduction
- Datastore Annotation
- Ontology Generation
- ETL Transformations
- Conclusions

Outline

- Introduction
- Datastore Annotation
- Ontology Generation
- ETL Transformations
- Conclusions

Extract-Transform-Load (ETL)



Problem Description

- Conceptual design of ETL processes
  - is a critical task performed at the early stages of a DW project
  - describe the integration of data from heterogeneous sources into the Data Warehouse
- Two main goals
  - specify inter-schema mappings
  - identify appropriate transformations

Motivation

- The problem of heterogeneity in data sources
  - structural heterogeneity
    - data stored under different schemata
  - semantic heterogeneity
    - different naming conventions
      - e.g., homonyms, synonyms
    - different representation formats
      - e.g., units of measurement, currencies, encodings
    - different ranges of values

Motivation

- Lack of precise metadata/documentation
  - often available in natural language format
- Current approaches are not sufficient
  - commercial tools
    - focus on the graphical representation and design of the ETL transformations
  - research works
    - focus on the representation and formal description of the ETL transformations
- The identification of the necessary transformations and the inter-schema mappings
  - needs to be done manually
  - is time consuming and error prone

Overview of our approach

- Key idea
  - an ontology-based approach to facilitate the conceptual design of an ETL scenario
- An ontology
  - is a “formal, explicit specification of a shared conceptualization”
  - describes the knowledge in a domain in terms of classes, properties, and relationships between them
  - machine processable
  - formal semantics
  - reasoning mechanisms
- The Web Ontology Language (OWL) is used as the language for the ontology
  - W3C recommendation
  - based on Description Logics

Overview of our approach

- Method
  - Construct an appropriate application vocabulary
  - Annotate the data sources
  - Generate the application ontology
  - Apply reasoning techniques to
    - select relevant sources
    - to identify required transformations

A motivating example

Data Stores	Schemata
DS1	products(id,name,amount,price,guarantee,type,*sid*) stores(sid,name,location)
DS2	software(id,name,*sid*,quantity,price) hardware(id,name,*sid*,quantity,price) stores(sid,name,city,street,number)
DW	products(id,name,*sid*,quantity,price) stores(sid,name,city,street)

Paris/Rome/Athens

software/hardware

Above 200 Euros

software

prices in Dollars

Paris/Rome/Athens

hardware

only software products

Rome/Athens

From 500 to 1500 Euros

Outline

- Introduction
- Datastore Annotation
- Ontology Generation
- ETL Transformations
- Conclusions

Datastore Annotation

- Creating the application vocabulary

  - a set of common terms to describe the sources and the application requirements

  - $V = (V_C, V_P, V_F, V_T, f_P, f_F, f_T)$

    V_C: concepts of the domain

    V_P: concept features $\quad f_P : V_P \rightarrow V_C$

    V_F: representation formats $\quad f_F : V_F \rightarrow V_P$

    V_T: feature values $\quad f_T : V_T \rightarrow V_F \cup V_P$

Datastore Annotation

- A relational datastore DS comprises
  - a set of relations R_{DS}
  - a set of attributes A_{DS}
- Relation mappings
  - $f_{RM} : R_{DS} \rightarrow V_C$
    - maps each relation to a primitive concept
- Attribute mappings
  - $M_{AM} \subseteq A_{DS} \times V_P$
    - maps attributes to features

Datastore Annotation

- Attribute annotation
  - $I_{R,p} = (\varphi, min, max, T, n, R', \Gamma_f, \Gamma_\alpha)$
    - φ: the representation format
    - min, max, T: range of values
    - n: cardinality
    - R': referenced relation (foreign keys)
    - Γ_f, Γ_α: function and attributes used for aggregation

Reference example (cont'd)

- Application vocabulary

Vc = {product, store}
$V_{Pproduct}$ = {pid, pName, quantity, price, type, storage}
V_{Pstore} = {sid, sName, city, street}
V_{Fpid} = {source_pid, dw_pid}
V_{Fsid} = {source_sid, dw_sid}
V_{Fprice} = {dollars, euros}
V_{Ttype} = {software, hardware}
V_{Tcity} = {paris, rome, athens}

- Datastore mappings

DS1		
products → product	id → pid name → pName amount → quantity	price → price type → type sid → storage
stores → store	sid → sid name → sName	location → city location → street

- Datastore annotation

DS1		φ	min	max	T	n	R´	Γ_f	Γ_α
products	I_{pid}	source_pid	-	-	-	1	-	-	-
	I_{pName}	-	-	-	-	1	-	-	-
	$I_{quantity}$	-	-	-	-	-	-	-	-
	I_{price}	euros	200	-	-	1	-	-	-
	I_{type}	-	-	-	{software, hardware}	1	-	-	-
	$I_{storage}$	-	-	-	-	1	store	-	-
stores	I_{sid}	source_sid	-	-	-	1	-	-	-
	I_{sName}	-	-	-	-	1	-	-	-
	I_{city}	-	-	-	{paris, rome, athens}	1	-	-	-
	I_{street}	-	-	-	-	1	-	-	-

Outline

- Introduction
- Datastore Annotation
- Ontology Generation
- ETL Transformations
- Conclusions

Ontology Generation

- An OWL ontology is generated based on the application vocabulary and the annotations

- The ontology consists of
  - a set of primitive classes corresponding to the specified concepts, representation formats and ranges or sets of values
  - a set of properties corresponding to the specified features of the concepts of the domain
  - a set of defined classes representing the datastores, based on the datastore mappings and annotation

- A reasoner is used to infer the class hierarchy

Reference example (cont'd)

- The class hierarchy

- owl:Thing
  - Above_200
    - From_500_To_1500
  - City_Values
  - Dollars_Price
  - DW_Pid
  - DW_Sid
  - Euros_Price
  - Product
    - DS1_Products
    - DS2_Hardware
    - DS2_Software
    - DW_Products
  - Source_Pid
  - Source_Sid
  - Store
    - [DS1_Stores, DS2_Stores]
    - DW_Stores
  - Type_Values

- Definition for class DS1_Products

OWL-Class: DS1_Products

Intersection of:
(∀pid . Source_Pid)
Product
(= 1 storage)
(∀storage . DS1_Stores)
(= 1 pid)
(= 1 pName)
(= 1 price)
(∀type . ((∃hasValue . {hardware})
∪ (∃hasValue . {software})))
(∀price . (Euros_Price ⊓
Above_200))
(= 1 type)

Outline

Generating ETL transformations

- Common types of ETL operations

Symbol	Name	Functionality
$\sigma(P, R)$	select	Filters the values of property P, excluding those values that do not belong to the set specified by R.
$f(P, F_1, F_2)$	convert	Converts the values of property P from the representation format F_1 to the representation format F_2.
$add(\alpha, v)$	add	Adds attribute α to the current schema, setting its values to v, where v is either a default value or null.
$nn(p)$	not null	Deletes the data records having a null value for property p.
$dd(P)$	detect duplicates	Detects, and appropriately removes, records having the same value for property P.
$\pi(\alpha_1, \ldots, \alpha_n)$	project	Projects the current schema preserving only the attributes denoted by $\alpha_1, \ldots, \alpha_n$.
$c(\alpha_1, \ldots, \alpha_n, P)$	concatenate	Concatenates the values of attributes $\alpha_1, \ldots, \alpha_n$ to set the value of property P.
$s(\alpha, P_1, \ldots, P_n)$	split	Splits the value of attribute α to set the values of properties $P_1, \ldots, P_n$.
$\gamma(f, P, P_\gamma)$	aggregate	Aggregates the values of property(-ies) P by grouping them by those of property(-ies) P_γ and applying the function(-s) f.
U	union	As the operator "union" in SQL.
J	join	As the operator "join" in SQL.
D	distribute	The "inverse" operation of join: if the data records contain attributes from multiple relations, as a result from a previous join operation, then this operation distributes data to the involved relations accordingly.

Generating ETL transformations

- Two main steps
  - select relevant sources to populate each DW element
  - identify required data transformations between the sources and the DW

Generating ETL transformations

- Select relevant sources
  - a source relation R_S, represented by class C_S
  - a target relation R_T, represented by class C_T
  - R_S is provider for R_T, if
    - C_S and C_T have a common superclass
      - ensures that the integrated data records have the same semantics
    - C_S and C_T are not disjoint
      - prevents data integration between datastores with conflicting constraints

Generating ETL transformations

- Identifying transformations (I)
  - use **`project`** operations to exclude attributes not mapped to the ontology
  - use **`concatenate/split`** operations for n:1/1:n mappings
  - use **`join`** operations for foreign key attributes

Generating ETL transformations

- Identifying transformations (II)
  - if $C_S \equiv C_T$ or $C_S \sqsubset C_T$, no transformations are required
  - if $C_T \sqsubset C_S$, for each additional constraint of C_T the corresponding **`select`**, **`aggregate`** or **`not null`** transformation is added
  - else, as previous plus **`convert`** operations

Generating ETL transformations

- Identifying transformations (III)
  - use **`add`** operations for missing attributes
  - use **`union`** operations for data coming from multiple sources
  - use **`distribute`** operations to load data to more than one relations in the DW
  - use **`detect duplicates`** operations for target attributes having a *not null* constraint

Reference example (cont'd)

- Reasoning on the mappings

c(street, number, street)

DS1		
products → product	id → pid name → pName amount → quantity	price → price type → type sid → storage
stores → store	sid → sid name → sName	location → city location → street

DS2		
software → product	id → pid name → pName sid → storage	quantity → quantity price → price
hardware → product	id → pid name → pName sid → storage	quantity → quantity price → price
stores → store	sid → sid name → sName	city → city street → street number → street

Reference example (cont'd)

- Reasoning on the definitions

σ(type, {software})

OWL-Class: DS1_Products

Intersection of:
(∀pid . Source_Pid)
Product
(= 1 storage)
(∀storage . DS1_Stores)
(= 1 pid)
(= 1 pName)
(= 1 price)
(∀type . ((∃hasValue . {hardware})
∪ (∃hasValue . {software})))
(∀price . (Euros_Price ⊓
Above_200))
(= 1 type)

OWL-Class: DW_Products

Intersection of:
(= 1 quantity)
(∀pid . DW_Pid)
Product
(∀price . (Euros_Price ⊓
From_500_To_1500))
(= 1 storage)
(= 1 pid)
(= 1 pName)
(= 1 price)
(= 1 type)
(∀type . (∃hasValue . {software}))
(∀storage . DW_Stores)

Outline

- Introduction
- Datastore Annotation
- Ontology Generation
- ETL Transformations
- Conclusions

Conclusions

- An application ontology to
  - model the domain
  - formally specify the semantics of the datastore schemata
- Automated reasoning techniques to
  - identify relevant sources for populating the DW
  - infer required conceptual transformations for propagating data from the sources to the DW

Current and Future Work

- Extend the approach to non-relational sources
- Evaluation on real-world ETL scenarios
- Maintenance/adaptation of the ETL workflow
- Automated design of the conceptual DW schema

Thank You

Questions