SEEK: Scalable Extraction of Enterprise Knowledge

Joachim Hammer

Dept. of CISE
University of Florida

20-June-2002
Outline of Talk

- Motivation
- SEEK Information Architecture
- Knowledge Extraction
  - Schema Extraction
  - Code Analysis
- Extraction Scenario
- Evaluation
- Status and Future Work
Database Research & Development Center at the University of Florida

- Affiliated with Computer Science Department
  - Funded entirely through research grants brought in by participating faculty

- Members
  - Faculty: Stanley Su (Founder & Director), Chris Jermaine, Joachim Hammer, Markus Schneider, Herman Lam (EE Dept.)
  - Several Visiting Scholars
  - Students: ~10 Ph.D., ~30 M.S. & undergraduates
  - Staff: Secretary and system administrator
Areas of Expertise in Center

- Knowledge management
  - Object-oriented knowledge base management
  - Knowledge extraction from legacy systems
- Workflow and information brokering
- Active database management systems
- Data transformation and integration
- Data warehousing
- Spatial and spatio-temporal databases incl. visual specification languages
- Database systems for semi-structured data and XML
SEEK Project

Faculty
Joachim Hammer
Mark Schmalz
William O’Brien
Ray Issa
Joe Geunes
Sherman Bai

Students
Sangeetha Shekar
Nikhil Haldevnekar
Jae-Hyun Choi
Rodrigo Castro
Huanqing Lu
Bibo Yang
Bo Lu

- Sponsored by NSF
- Year 1 of initial 3-year effort
Motivation

- Need for decision/negotiation support to improve performance and customization across extended business networks
- Hard ... participating firms have unique and often incompatible information systems, varying levels of sophistication
  - Current approaches require manual coding of connection software - Not scalable
- Development of a tool kit to facilitate integration of heterogeneous legacy data and knowledge
SEEK Environment & Context

extended enterprise

supplier

sub/supplier

coordinator/lead

Secure Hosting Infrastructure

analysis (e.g., E-ERP)

SEEK

SEEK

SEEK
SEEK Information Architecture

Connection Toolkit

Legacy Source

Application

Secure, value-added extraction of firm knowledge

Seek Components

Analysis Module

Knowledge Extraction Module

Wrapper

AM: query analysis, knowledge composition (mediation)

W: source connection and translation

KEM: configuration of W and AM at build-time
Run-Time: Querying and Analysis

- Different information contexts: Application, Analysis Module, Source
  - Translator needed to convert between information contexts
  - Assume existence of translator between AM and application contexts
- Analysis module provides robust (value-added) mediation
  - Solution strategy based on information available in source
  - Capable of composing final answer out of multiple source results
- SEEK wrapper responsible for syntactic and semantic conversions
  - Formulates source queries based on capabilities of source
  - Restructures source results to conform to information context of AM
Run-Time Interactions

- **Q/R^C**: query/result expressed in information context \( C \)
  - \( H \) stands for hub (e.g., e-marketplace, supply chain coordinator)
  - Note that single query from hub may generate one or more queries from AM
  - Subscript \( i \) denotes specific query/result within set
Build-Time: Knowledge Extraction

- Extract information about legacy source to facilitate development of wrapper and configuration of AM
  - Produces “description” of accessible knowledge in source
- *Schema extraction* from data source
- *Analysis of application code* to augment schema with semantics and extract business rules
- *Schema Matching* to infer mappings between information context of AM with that of legacy source
- Quality and accuracy of extracted knowledge (and hence the wrapper and AM) improves over time and with human input
Overview

Data Reverse Engineering (DRE)

- Schema Extractor (SE)
- Semantic Analyzer (SA)

Legacy Source
- Legacy DB
- Legacy Application Code

Reports
- Schema Information
- Embedded Queries

Domain Model
- Domain Ontology

Schema Matching (SM)
- Schema, semantics
- business rules
- Mapping rules

train, validate
to wrapper generator

revise, validate
Schema Extraction

- Based on data reverse engineering algorithms, e.g., Chiang 94/95, Petit et al. 96
  - Reduced dependency on human input
  - Eliminated limitations (e.g., consistent naming, legacy schema in 3-NF)
- Use database catalog to directly extract concepts and simple constraints
- Use database instances to infer relationships and constraints
- Interact with code analysis to augment schema with semantics
- Produces E/R-like representation of the entities, relationships, and constraints
Semantic Analysis

- Identify semantic descriptions for schema items in database in application code
  - E.g., trace database schema names back to output statements
- Using *code slicing* to reduce application code to only those statements that are of interest to the analyzer (Horwitz, Reps 92)
- Apply *pattern matcher*
  - discover associations among variables
  - identify patterns that encode business information
    - E.g., business rules encoded in IF-THEN-ELSE statements
Architecture

Queries

1. AST Generation

2. Dictionary Extraction

3. Code Analysis

4. Inclusion Dependency Mining

5. Relation Classification

6. Attribute Classification

7. Entity Identification

8. Relationship Classification

Legacy Source

DB Interface Module

Data

Application Code

Metadata Repository

AST

Schema

XML DTD

Knowledge Encoder

XML DOC

To Schema Matcher

configuration

1. AST Generation

2. Dictionary Extraction

3. Code Analysis

4. Inclusion Dependency Mining

5. Relation Classification

6. Attribute Classification

7. Entity Identification

8. Relationship Classification

Legacy Source

DB Interface Module

Data

Application Code

Metadata Repository

AST

Schema

XML DTD

Knowledge Encoder

XML DOC

To Schema Matcher
Extraction Scenario

- Project scheduling in building construction domain
- Legacy source using relational database system and scheduling application
  - Application written in C
  - Oracle 8i RDBMS
  - Using Oracle Pro*C as gateway
- Scenario based on actual subcontractor participating in construction of new School of Architecture at UF
## Legacy Database Schema

<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Proj</strong></td>
<td><code>[P_ID, P_NAME, DES_S, DES_F, A_S, A_F, ...]</code></td>
</tr>
<tr>
<td><strong>Avail</strong></td>
<td><code>[PROJ_ID, AVAIL_UID, RES_ID, AVAIL_FROM, AVAIL_TO, UNITS]</code></td>
</tr>
<tr>
<td><strong>Res</strong></td>
<td><code>[PROJ_ID, RES_UID, RES_NAME, R_ACWP, R_BCWP, R_BCWS, ...]</code></td>
</tr>
<tr>
<td><strong>T</strong></td>
<td><code>[PROJ_ID, T_UID, T_ID, T_NAME, T_DUR, T_ST_D, T_FIN_D, ...]</code></td>
</tr>
<tr>
<td><strong>Assn</strong></td>
<td><code>[PROJ_ID, ASSN_UID, T_UID, R_ID, ASSN_BASE_C, ASSN_ACT_W, ...]</code></td>
</tr>
</tbody>
</table>

Sample Application Code

/* sample program to perform project scheduling */
char *aValue;
char *cValue;
int bValue = 0;

/* more code ... */
EXEC SQL SELECT T_ST_D, T_FIN_D INTO :aValue, :cValue FROM T
WHERE T_PRITY = :bValue;

/* more code ... */
int flag = 0;
IF (cValue <= aValue)
{
    flag = 1; /* exception handling */
}

/* more code ... */
printf ("Task Start Date %d", aValue);
printf ("Task Finish Date %d", cValue);

/* more code ... */
Step 1: Generation of AST

1. dcIns
2. embSQL
3. if
4. print
5. print

Program

embSQL

beginSQL

SQLselectone

columnlist

hostvariablelist

SQLAssignment

<id>

T_ST_D

<id>

T_FIN_D

<id>

aValue

<id>

cValue

<id>

T_PRITY

<id>

bValue
Step 2*: Dictionary Extraction

- **Proj**: \([\text{P\_ID, P\_NAME, DES\_S, DES\_F, A\_S, A\_F, ...}]\)
- **Avail**: \([\text{PROJ\_ID, AVAIL\_UID, Res\_ID, AVAIL\_FROM, AVAIL\_TO, ...}]\)
- **Res**: \([\text{PROJ\_ID, RES\_UID, RES\_NAME, R\_ACWP, R\_BCWP, R\_BCWS, ...}]\)
- **T**: \([\text{PROJ\_ID, T\_UID, T\_ID, T\_NAME, T\_DUR, T\_ST\_D, T\_FIN\_D, ...}]\)
- **Assn**: \([\text{PROJ\_ID, ASSN\_UID, T\_UID, R\_ID, ASSN\_BASE\_C, ...}]\)

- If key info not available, use embedded SQL to reduce set of possible key attributes
- Note, if raw schema was all we are after, done!
- Remaining steps 3-8 will produce schema that has richer semantics, including entities, relationships, etc.
Elimination Patterns

- If primary key info cannot be directly retrieved from catalog or schema, use elimination patterns
  - Rule out possibility that $a_1, \ldots, a_n$ can form a primary key

```
SELECT DISTINCT <selection>
FROM   Relation
WHERE  a1=<scalar_expression1> AND \ldots
       AND an=<scalar_expressionn>;

SELECT <selection>
FROM   Relation
WHERE  a1=<scalar_expression1> AND \ldots
       AND an=<scalar_expressionn>
GROUP BY \ldots ;
```
Step 3*: Code Analysis

- **Pre-slicing**
- Traversing AST in pre-order, identify all nodes corresponding to input, output and embedded SQL statements
- Maintain array containing the node number, statement name, and list of identifiers
  - Identifiers make up set of slicing variables

<table>
<thead>
<tr>
<th>Node number</th>
<th>Statement</th>
<th>Identifiers</th>
<th>Direction of Slicing</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>embSQL (Embedded SQL statement node)</td>
<td>aValue, cValue</td>
<td>Backwards (since both identifiers appear in output statements)</td>
</tr>
</tbody>
</table>
Code Slicing

- Produce *reduced AST* by retaining only those nodes that contain an occurrence of the slicing variable(s) in sub-tree
Program semantics are determined from pre-order traversal of reduced AST as follows:

- *dcln* node: data type of the identifier
- *embSQL*: mapping of identifier name to corresponding column name and table name
- *printf/scanf* nodes: extract ‘meaning’ of identifier from output text string
Business Rule Extraction

- Identify rules by matching templates against code fragments in AST
- So far, templates for business rules encoded in:
  - loop structures
  - conditional statements
  - mathematical formulae encoded in loop structures and/or assignment statements
- Note, occurrence of an assignment statement itself does not necessarily indicate the presence of a mathematical formula
  - likelihood increases significantly if statement contains slicing variable
## Result of Code Analysis

<table>
<thead>
<tr>
<th>Identifier Name</th>
<th>Meaning</th>
<th>Possible Business Rule</th>
<th>Data Type</th>
<th>Corresponding Column Name in Database</th>
<th>Corresponding Table Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>aValue</td>
<td>Task Start Date</td>
<td>if (cValue &lt; aValue)</td>
<td>Char * =&gt; string</td>
<td>T_ST_D</td>
<td>T</td>
</tr>
<tr>
<td>cValue</td>
<td>Task Finish Date</td>
<td>if (cValue &lt; aValue)</td>
<td>Char * =&gt; string</td>
<td>T_FIN_D</td>
<td>T</td>
</tr>
</tbody>
</table>
Extracted Business Rules

- Variables have been replaced by their extracted meaning (to the extent that they are known)

```c
Rule1: if(Task Beginning Date < early start date of task) 
{ 
    Task Beginning Date = early start date of task ; 
} 

Rule2: Project Cost = Project Cost + Task Cost ; 

Rule3: if(Resource category == "brick") 
{ 
    Resource Cost = 2500; 
} 
```
Step 4*: Discovering Inclusion Dependencies

- Identify constraints to help classify the extracted relations
  - Inclusion dependencies indicate existence of inter-relational constraints incl. class/subclass relationships
- Using foreign key info and equi-join queries (AST) to directly identify existing inclusion dependencies
- For remaining pairs of relations, identify IND by matching data types and names of attributes
  - Use *subset test* on database instances to verify existence of inclusion dependency
- Group INDS into POSSIBLE and FINAL
  - Solicit user input for INDs in POSSIBLE set
Extracted INDs

\[ \text{Assn}[T_{UID},\text{Proj}_{ID}] \ll T \ [T_{UID},\text{Proj}_{ID}] \]

\[ \text{Assn} \ [\text{Res}_{uid},\text{Proj}_{ID}] \ll \text{Res} \ [\text{Res}_{uid},\text{Proj}_{ID}] \]

\[ \text{Avail} \ [\text{Res}_{UID},\text{Proj}_{ID}] \ll \text{Res} \ [\text{Res}_{UID},\text{Proj}_{ID}] \]

\[ \text{Res} \ [\text{Proj}_{ID}] \ll \text{Proj} \ [P_{ID}] \]

\[ T \ [\text{Proj}_{ID}] \ll \text{Proj} \ [P_{ID}] \]

\[ \text{Assn} \ [\text{Proj}_{ID}] \ll \text{Proj} \ [P_{ID}] \]

\[ \text{Avail} \ [\text{Proj}_{ID}] \ll \text{Proj} \ [P_{ID}] \]
Step 5*: Classification of Relations

- Relations classified into
  - strong: strong real-world entity
  - regular: relationship between entities
  - weak: weak real-world entity
  - specific: relationship for which not all entities are currently present in schema

- Based on Chiang’s classification scheme

- Use primary key information (Step 2) and inclusion dependencies (Step 4)
  - User input required
Classification

- strong relationship relation
  - Proj
  - dependent of
  - Assn
  - dependent of
  - T

- weak relationship relation
  - Res
  - dependent of
  - Avail
  - weak relationship relation
Step 6: Classification of Attributes

- **Primary Key (PK):** Identifier for relation
- **Foreign Key (FK):** References primary key in related relation
- **Dangling (DK):** Attributes belonging to PK of weak entity-relation or specific relation that do not appear as PK of other relations
- **Non-Key (NK):** rest
## Attribute Classification

<table>
<thead>
<tr>
<th></th>
<th>PK</th>
<th>DK</th>
<th>FK</th>
<th>NK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proj</td>
<td>P_ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Res</td>
<td>Proj_ID + Res_UID</td>
<td>Res_UID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Proj_ID + T_UID</td>
<td>T_UID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avail</td>
<td>Proj_ID + Avail_UID</td>
<td>Avail_UID</td>
<td>Res_ID, Proj_ID</td>
<td></td>
</tr>
<tr>
<td>Assn</td>
<td>Proj_ID + Assn_UID</td>
<td>Assn_UID</td>
<td>R_ID+ Proj_ID, T_UID+ Proj_ID</td>
<td></td>
</tr>
</tbody>
</table>

All Remaining Attributes
Step 7 : Identify Entity Types
Step 8: Identify Relationship Types

- Identify relationships present as relations in the legacy schema
  - Relation types (regular and specific) obtained from Step 5 are converted into relationships (card N-M)

- Identify relationships among the entity types (strong and weak) that were not present as relations
  - *IS-A relationships* (card 1-1)
  - *Dependent relationship* (card 1-N, owner to weak)
  - *Other binary relationships* (card 1-N or 1-1)
Extracted E/R Schema

- **Proj**
  - P_ID
  - P_Name
  - Des_S

- **Res**
  - Proj_ID
  - Res_UID
  - Res_Name

- **T**
  - Proj_ID
  - T_UID
  - T_ID

- **Avail**
  - Proj_ID
  - Avail_UID

- **Assn**
  - has
  - N
  - M

Relations:
- Proj has Res
- Proj has T
- T has Assn
- Res has Res
- Res has Res
- Res has Avail
- T has Proj
- T has T
- T has Avail

Cardinalities:
- 1
- N
- M

Key Attributes:
- P_ID
- Proj_ID
- Res_ID
- T_ID
- T_UID
- Avail_UID
Evaluation

- Tested implemented SE prototype using 15 database projects from graduate database class
  - Wide range of applications, medium-sized projects (20-40 tables)
- Test 1: Extraction of schema
- Compared generated output (no user intervention) to design document produced by students, measured errors
  - Missing/undetected components
  - Phantom components
- Total number of errors grows with complexity of project
  - Extraction accurate, average 1 error for every 5 tables
  - Number of phantoms exceeds missing components
- Next: Similar evaluation for code analysis
Current Status & Future Plans

**Current**

- Implemented interactive knowledge extraction prototype consisting of SE and SA
  - Extracted knowledge stored in XML document
- Developing suitable model for representing extracted knowledge
- Setting up more extensive testbed
  - Data collection in cooperation with partners from construction industry (manufacturing examples planned)

**Future**

- Researching approaches to schema matching
- Integration with wrapper development toolkit
- Enhance DRE with ability to improve with time and usage cases
- Conduct detailed cost/benefit analysis of approach
Summary and Conclusion

- SEEK is a structured approach to integrating domain-specific legacy sources
- Modular architecture provides several important capabilities
- (Semi)automatic knowledge extraction
  - DRE, semantic analysis, schema matching
- Important contributions to theory of knowledge capture and integration
- Requirement for building scalable sharing architectures
- Enabling technology for (semi)automatic ontology creation
  - Enabler for Semantic Web?
More Info

Publications:

- “SEEKing knowledge in legacy information systems to support interoperability.” *ECAI-02 Workshop on Ontologies and Semantic Interoperability*, Lyon, France, July 21-26, 2002, accepted for publication.


Web Site:

http://www.dbcenter.cise.ufl.edu/seek/index.htm