Service Identification in Legacy Code Using Structured and Unstructured Analysis

Inbal Ronen, Netta Aizenbud, Ksenya Kveler

May 2007
Outline

- Motivation
- Structured and Unstructured Analysis
- Our Method
- Case Study
- Future Directions
- Conclusions
SOA Transformation

- Moving to SOA has become a strategic goal of many companies
  - Flexible, distributed architecture
  - Better adaptation to a rapidly changing business environment
  - Better alignment of business processes and underlying applications with business goals

➤ There is a need for tools that assist in the transformation process
Reuse of Legacy Systems

- Lots of resources and time have been spent on existing legacy systems
- Need to retain as much as possible of previous investments
- Requires identification of where a service or part of it is already implemented and can be reused
- Manual identification of candidate code sections is tedious and requires domain experts

⇒ A (semi) automatic tool is needed to assist in this process
Transformation Approaches

- **Top Down**
  - Define *to-be* model and implement it
  - No consideration of existing system

- **Bottom Up**
  - Start from given legacy *as-is* system and expose as SOA
  - Harder to adapt to new business models

- **Meet in the middle**
  - Combine Top Down and Bottom Up approaches
  - Start from target to be model
  - Map into existing system
  - Exploit reuse and implement if needed
Structured and Unstructured Source Code Analysis

- **Unstructured Analysis**
  - Information Retrieval (IR) techniques, e.g. tokenization, usage of thesauri and stemming of source code and comments
  - No consideration of code structure and semantics

- **Structured Code Analysis**
  - Classic static analysis, e.g. control flow, data flow
  - No comment analysis
  - No identification of non-exact matches

- **Combination of techniques facilitates effective and precise service implementation search**
SOMA – Service Oriented Modeling and Architecture

An IBM end-to-end SOA method for the identification, specification, realization and implementation of services, components and flows

- **Service Identification** – combines top-down, bottom-up, and meet-in-the-middle techniques for the identification of services to be implemented in the new SOA environment

- **Service Specification** – further designs the subsystems that were found in the previous step and specifies the coordination between them. Details the components that implement the services

- **Service Realization** - defines the software that realizes a given service

⇒ Our method fits into the **Service Identification** phase
Service Identification Method

- Service definition includes a *service title* – short functionality description (e.g. “Add a new customer account”)

- Our method
  - receives a service title as input
  - searches for potential implementations in the code
  - ranks the results by relevance to the service title
Example 1

- **Identify P0030-PROC-CREATE-ACCT procedure**
  - The name indicates that it implements the desired functionality
  - Procedure name does not include the exact terminology
    - Contains **CREATE** - a synonym of “add”
    - Contains **ACCT** - an abbreviation of “account”.

```assembly
1 000100 ID DIVISION.
2 000200
3 000300 PROGRAM-ID. PROG1.
4 ... 
5 214000 PROCEDURE DIVISION
6 214100**************************************
7 214400* HANDLES RECORD ADDITION
8 214410* REQUEST. AFTER OPERATION IS
9 214500* COMPLETED, EVENT REPORT IS
10 214510* CREATED. ACCOUNT MANDATORY
11 214600* FIELDS ARE SET IN THIS
12 214700* PROCEDURE
13 214800**************************************
14 215000 P0030-PROC-CREATE-ACCT.
15 215100
16 215200 MOVE +0 TO TRAN-001-RECORD
```
Example 2

- **Identify P0040-PROC1 procedure**
  - The preceding comment resembles the desired service title
  - The comment adheres to the company convention:
    
    MATCH inside the convention strengthens the impression that the procedure is a good candidate
Example 3

- **Identify P0050-PROC-REP procedure**
  - Neither procedure name nor its comment is relevant to the service title
  - References `AA-REQ-COUNTER` variable
  - The variable holds the “number of add acct requests” as stated by the variable definition comment

- **Low ranking**
Method Overview

- **Stage 1: source code processing**
  - Analyze code structure
  - Identify components of interest in code and comments
  - Insert code and processing results into repository

- **Stage 2: search and ranking**
  - Search for service title matches in the artifacts that have been processed
  - Rank match relevance, taking into account structural and semantic context

* Stage 2 can be repeated for multiple service titles over the same processed code artifacts
Source Code Processing Stage

- **Identify programming constructs** (e.g., variable declarations, procedure names, comments)
  - Perform shallow analysis based mostly on a composition of regular expressions
  - Perform deep static code analysis (control flow and data flow detection)
  - Analyze comments, exploit conventions
  - Mark constructs using annotations

- **Tokenize - enable matching of substrings**
  - Consider special characters (spaces, commas, underscores) and code naming practices (Hungarian notation, CamelCase)

- **Insert tokens and annotations into repository**
  - Ignore tokens with low semantic value (e.g., “and”, “the”)
  - Our method uses a search engine as the repository (provides indexing and querying capabilities)
Search and Ranking Stage

- **Tokenize service title**
  - Apply the techniques used during the source code processing stage

- **Construct and execute search query**
  - Include the tokens from the service title
  - Exploit unstructured analysis capabilities of the search engine (e.g. stemming, thesauri and abbreviation usage) to search for inexact matches
    - Provide common language and domain-specific thesauri and abbreviation dictionaries to the search engine

- **Analyze query results**
  - Search engine returns the location of each query token occurrence (or its synonym)
  - The method assembles valuable occurrence combinations such that
    - There is exactly one match for every token (or one of its synonyms)
    - Token match locations are close to each other

- **Rank match relevancy**
  - Evaluate textual similarity: 100% for exact match of all service title tokens
  - Aggregate results to procedure level
  - Apply supplemental ranking heuristics based on semantic context
Ranking Heuristics

- High score for match in procedure declaration
- Higher score for match in procedure declaration and in the adjacent comment
- Match in variable declaration or close to it
  - Use data flow analysis results to find variable referencing code
  - Identify the reference location as a match with low score
  - The rationale: variable definition comment might include the service title, no additional comment in the variable usage code
- Separate matches for service title subject (noun) and action (verb)
  - Look for noun matches in or close to a variable declaration
  - Look for verb matches in or close to the variable referencing code

```
075340 01 ACNT-NUMBER PIC S9...
...
215000 P0040-PROC1.
215150*** INITIALIZE NUMBER
215160*** BEFORE ADDING
215200 MOVE +0 TO ACNT-NUMBER
```

Noun match
Verb match
Case Study Implementation

- Implemented a plug-in to an internal IBM tool that supports the IBM SOMA methodology
- Leveraged the Unstructured Information Management Architecture Open Source framework (UIMA)
- Used Juru as the underlying search engine
- Enables the user to select a service definition from the to-be model
- Returns a list of ranked implementation candidates
Case Study

- **Used a customer application (a large bank in North America)**
  - Consists of a set of 30 COBOL programs and 48 copybooks, with a total size of 60K lines

- **Searched for six service titles that are common for banking applications**

- **Search in two levels**
  - The program level – identify the program that is more likely to contain the requested functionality, by calculating the total match rate in proportion to its size
  - The procedure level – search for specific procedures that implement the requested functionality. The rank given to each procedure was calculated using the heuristics described before
Case Study Results

- Manual inspection of the procedures pointed to by our method shows that the results are valuable with 80% success rate
  - Users are provided with valuable candidates in the legacy code for service realization
  - Greatly assists in the transition to a SOA enabled architecture

- For service title “Reject Transaction” the results were inconsistent
  - The service title is too general and not sufficiently focused
    - A transaction is a widely used concept in COBOL programs
    - The rejection operation can appear everywhere a transaction occurs
Case Study Results cont.

- Some programs appear in the result list of the program level search only
  - A match rate that depends only on the textual fitness of the searched service title to the code text is not accurate enough
  - The additional heuristics focus the results on more appropriate areas in the code

- Identified a main program (A) that routes the execution to different programs according to the business function
  - Guides the user to the entry point of most of the business functions, from there the execution path of a specific function can be followed

<table>
<thead>
<tr>
<th>Service Title</th>
<th>Program ranked first</th>
<th>Program ranked second</th>
<th>Program ranked third</th>
<th>Program ranked forth</th>
<th>Program ranked fifth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminate Payment</td>
<td>D</td>
<td>B</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject Transaction</td>
<td>I</td>
<td>J</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modify Status</td>
<td>F</td>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modify Rating</td>
<td>G</td>
<td>H</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Close Account</td>
<td>F</td>
<td>A</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Account</td>
<td>B</td>
<td>F</td>
<td>M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Future Directions

- **Use additional Natural Language processing (NLP) techniques**
  - Consider sentence breaks in comments, e.g. “… account. Open…”
  - Identify main noun and verb in service title, e.g. for title “Open a new user account” identify “account” and “open”
    - Give higher ranking to these in query and ranking
    - In OO code look for class names that include the noun and method names that include the verb
    - Identify matches that include a verb in procedure declaration and a noun as one of the parameter names or variable names in the body
  - Consider whether a token in title is a verb or noun during query expansion, e.g. for “Record Status”, “record” is a verb and not a noun
Future Directions cont.

- **CRUDL (Create, Read, Update, Delete or List) Analysis**
  - Identify language constructs that perform these tasks, e.g. for the notion of “creation” in the service title look for `INSERT` in SQL or `new` and `malloc` statements in code.

- **Consider additional information on a service in the model, e.g.**
  - Service descriptions
  - Service interfaces

- **Consider feedback from previously performed mappings**
Summary

- Presented a method for the identification of services in legacy source code in the context of SOA transformation
- The technology uses a combination of structured and unstructured analysis techniques over source code and its comments
  - Considers information found in comments
  - Elaborates in-exact matches
  - Takes into account structural and semantic context of a match during ranking
- Compared to manual inspection of the code, which is the prevalent practice nowadays, our method significantly reduces the required effort
Contacts

- **Authors**
  - Netta Aizenbud neta@il.ibm.com
  - Ksenya Kveler ksenya@il.ibm.com
  - Inbal Ronen inbal@il.ibm.com

- **Software Asset Management Group at HRL**
  - Jonathan Bnayahu bnayahu@il.ibm.com