ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/308812065

A Pattern-Based Code Transformation Approach for Cloud Application Migration

Conference Paper · June 2015

DOI: 10.1109/CLOUD.2015.15

CITATIONS
11

READS
137

6 authors, including:

Zhengong Cai
Zhejiang University
7 PUBLICATIONS **67** CITATIONS
SEE PROFILE

Liping Zhao
The University of Manchester
150 PUBLICATIONS **1,443** CITATIONS
SEE PROFILE

Xinyu Wang
Anhui Medical University
204 PUBLICATIONS **3,866** CITATIONS
SEE PROFILE

Xiaohu Yang
Zhejiang University
107 PUBLICATIONS **1,578** CITATIONS
SEE PROFILE

All content following this page was uploaded by Liping Zhao on 14 December 2021.

The user has requested enhancement of the downloaded file.

A Pattern-Based Code Transformation Approach for Cloud Application Migration

Zhengong Cai[1], Liping Zhao[2], Xinyu Wang[3], Xiaohu Yang[3], Juntao Qin[1], Keting Yin[1]

[1] Software College, Zhejiang University, Ningbo China {caizg, 21451063, yinkt}@cst.zju.edu.cn

[2] School of Computer Science, The University of Manchester, Manchester UK liping.zhao@machester.ac.uk

[3]College of Computer Science, Zhejiang University, Hangzhou China {wangxinyu, yangxh}@zju.edu.cn

Abstract—**To support the migration of software applications to the cloud environment, cloud venders have proposed different migration methodologies and guidelines. Yet, most of them require human intervention, involving manually performing repetitive tasks. This paper proposes a pattern-based transformation approach for cloud application migration. The approach automatically modifies the source code of an application before the migration, to make it cloud-ready, and then transforms the source code to the target code in the cloud environment. The approach is supported by three key elements (patterns, rules and templates) and a process that systematically applies these elements. First, a pattern matching engine based on a regular expression processing technique is used to identify the parts of the source code that require modification and to extract the essential tokens from the source code for code transformation. Next, transformation rules are invoked to change the source code into the target code using a template, designed according to the target cloud environment. The proposed approach has been demonstrated on 19 open-source projects, by migrating them to Amazon Web Services.**

Keywords- Cloud computing; cloud migration; pattern-based code transformation; transformation rule; AWS

I. INTRODUCTION

Cloud computing has become a dominating force in the IT industry, due to its higher scalability, greater reliability and lower cost. In recent years, businesses, including both startup companies and traditional industries, have begun to move their in-house data centers and software applications to the cloud. Cloud migration, as this trend has been called, has become an important IT strategy of most companies.

Yet, one critical issue facing cloud migration is that cloud storage (database and file) access and security services are different from those of traditional systems. Consequently, applications migrated to the cloud platform cannot directly use these basic cloud services. Furthermore, due to incompatible storage and file structures, migrated applications cannot take the full advantage of cloud computing features, such as dynamic scalability. To solve these problems, cloud computing venders have suggested some techniques for transforming source code [10]. So far, however, most of these techniques are manual operations and repetitive, making cloud migration both time-consuming and error-prone.

To support code transformation and architecture transformation, technologies and methodologies have also begun to emerge [10], [11], [14], such as transformation decision making processes [13], transformation methodologies [5], transformation methods [8], and supporting toolkits [15]. Yet, most of these developments aim to help manual transformation. While open source transformation tools like stratego/xt [18] are available for automated code transformation, they are not reliable. There are also commercial solutions and tools [19], but they offer customized solutions, rather than general transformation methods or tools.

This paper proposes a pattern-based code transformation approach for cloud migration to perform the repetitive transformation tasks of storage access and security services. This approach consists of a process that iteratively transforms the source code into the target cloud code. The process uses a set of code patterns to identify the database access, file access and security services in the source code and then applies a set of templates to change these services into their corresponding cloud services. The transformation process is driven by a set of transformation rules. The proposed approach has been used to migrate 19 open source projects to the Amazon Web Services (AWS) cloud.

The remaining paper is organized as follows. Section II introduces the proposed approach and describes its detailed processing steps. Section III demonstrates this approach through 19 open-source projects. Then Section IV discusses related work on transformation solutions and tools, and finally, Section V concludes the paper.

II. THE PROPOSED APPROACH

A. Overall Process

As described above, the proposed approach takes the source code of an application as input and applies an iterative process to transform the code into the cloud compatible code. Each process iteration consists of four main steps: scan code, locate source code to transform, design templates according to the target environment, and execute transformation rules. The scanning step can be executed manually or by existing scanning tools. In the second step, a pattern matching method is used to locate the right source code to transform, and to extract some essential tokens that should not be changed during the transformation. Then the templates for the target code are designed according to the target platforms. Finally, the transformation rules are designed and executed

Authorized licensed use limited to: University of Manchester. Downloaded on December 14,2021 at 16:43:20 UTC from IEEE Xplore. Restrictions apply.

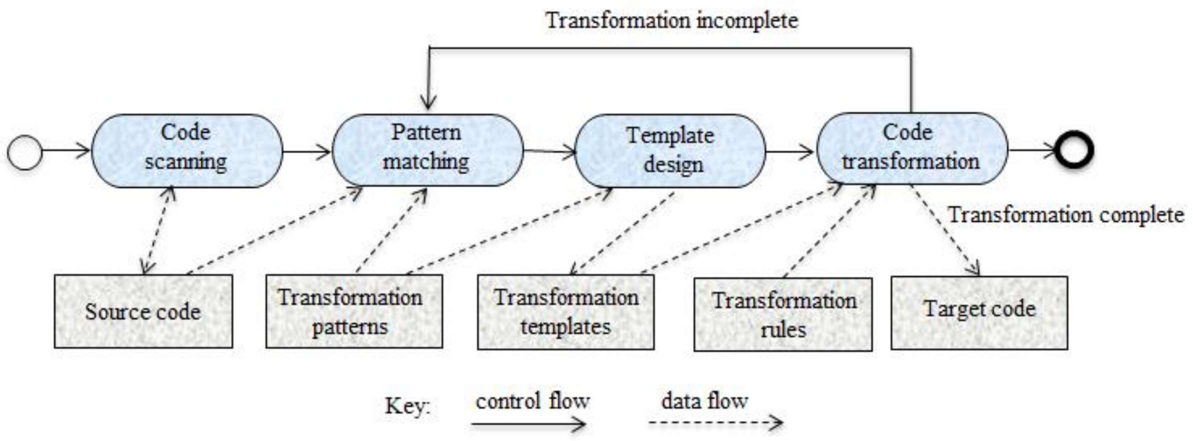
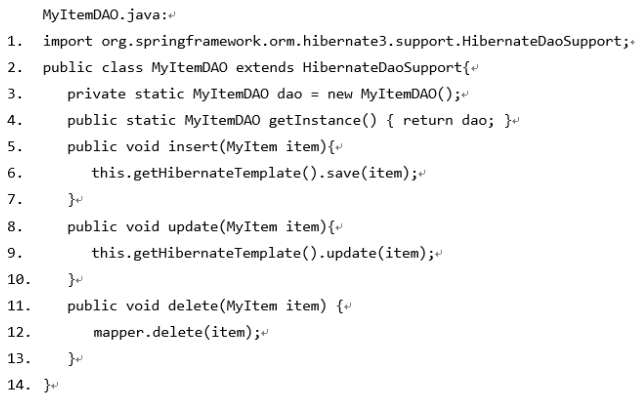


Figure 1. The proposed pattern-based transformation approach

to transform the located source code with designed templates. Figure 1 illustrates this overall process whereas the detailed process steps are presented in the following subsections.

This is an iterative process from step 2 to step 4. For each iterative process, a transformation pattern is applied, and the following steps are executed with relevant templates and rules. The transformation is complete when all the transformation patterns are applied.

To illustrate this process, a running source code example is given, as Figure 2 shows. This is a Java source code for database access using *hibernate,* which is an object-relation mapping framework in java. In the source code, *MyItem* is a class defining a business object. The database operations are insert, update and delete. Migrating this source code to the cloud means that the cloud storage should be used. In this running example, the AWS *DynamoDB* is used as the target storage services.



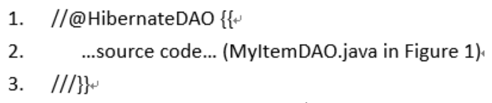
```
MyItemDAO.java:
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
public class MyItemDAO extends HibernateDaoSupport{
   private static MyItemDAO dao = new MyItemDAO();
   public static MyItemDAO getInstance() { return dao; }
   public void insert(MyItem item){
      this.getHibernateTemplate().save(item);
   }
   public void update(MyItem item){
      this.getHibernateTemplate().update(item);
   }
   public void delete(MyItem item) {
      mapper.delete(item);
   }
}
```

Figure 2. A running example: Migrating the database access source code to the cloud environment.

B. Step 1: Code Scanning

This step selects and annotates the specific blocks of code from the source code for transformation. It can be performed manually or by using third-party tools. The code scanning is performed only once to identify what the code is for. Annotation begins with "//" and will therefore not affect the compilation of the source code, as Figure 3 shows. The annotation can be used as an index for locating source code to transform or for further semantic analysis.

The input to the scanning steps are source code and a lot of code features describing what code to locate. The features can be key identifiers, token sequences or a syntax tree. The code satisfying these features would be located and annotated.



```
//@HibernateDAO {{
      ...source code... (MyItemDAO.java in Figure 1)
///}}
```

Figure 3. Annotated source code

C. Step 2: Pattern Matching

This step matches the annotated code with predefined patterns. Based on these patterns, some code information can be extracted and transformed into the target code. A pattern in our approach is a recurring code segment that needs to be transformed for cloud migration.

An extended regular expression is used to express our patterns. The extended regular expression has similar syntax with the regular expression, but it differs from the regular expression in two ways. First, the extended regular expression is token-based. Thus the granularity of pattern matching is a token rather than a character. The second difference is the semantic context. The extended regular expression has a special syntax to represent the context, e.g. "{}" as beginning and end of a code block in Java. In the matching results, the "(" and ")" should appear in pair. It avoids the incomplete code block. In Table I, the syntax of proposed patterns are listed. "$x" is used to extract a token from the matched code, and "$x(#..#)" is used to assign the matched code block to the variable *x*.

To do the pattern matching, the engineers can define a pattern using the pattern syntax. The pattern can be general,

Authorized licensed use limited to: University of Manchester. Downloaded on December 14,2021 at 16:43:20 UTC from IEEE Xplore. Restrictions apply.

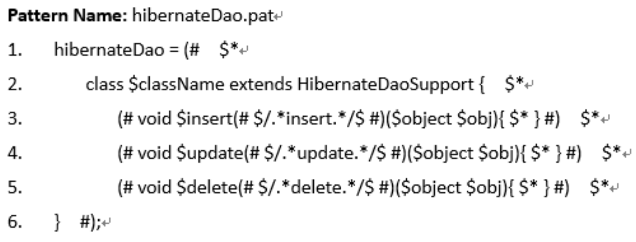
or domain-specific if it is too difficult to design general pattern. A pattern should be started with "（#" and ended with "#）". If the engineers want to extract some tokens for further processing, "*$x*" can be used to assign the matched code to the variable *x*.

TABLE I. PATTERN SYNTAX LIST

Pattern Syntax	Description
(#	the start of pattern matching
#)	the end of pattern matching
$	match any token
?	match zero or one of the preceding token
*	match zero or more of the preceding token
+	match one or more of the preceding token
|||	inclusive OR relationship
&&&	inclusive AND relationship
$x	extract a matched token and assign it to variable x
$x(# … #)	extract a token sequence matched the pattern in "(#...#)" and assign it to x

A pattern example is given in Figure 4. The pattern, which is called "*hibernateDao",* is used to match the part of the source code that performs database access with hibernate. In Figure 4, Line 2 matches the class definition of a data access object. Lines 3, 4 and 5 match the operations insert, update and delete methods. "$*" matches any tokens. "$*className*", "$*object*" and "$*insert*" / "$*update*" / "$*delete*" are to extract the source elements that would be filled into target templates.

For the same function, different programmers may write different source code. It is nearly impossible to use a general pattern to match all the source code. Thus, it is not a good idea to design a general pattern for all cases. Instead, we have designed some domain-specific patterns that can match a subset of the cases. The patterns have been applied to transform over ten open source java projects.



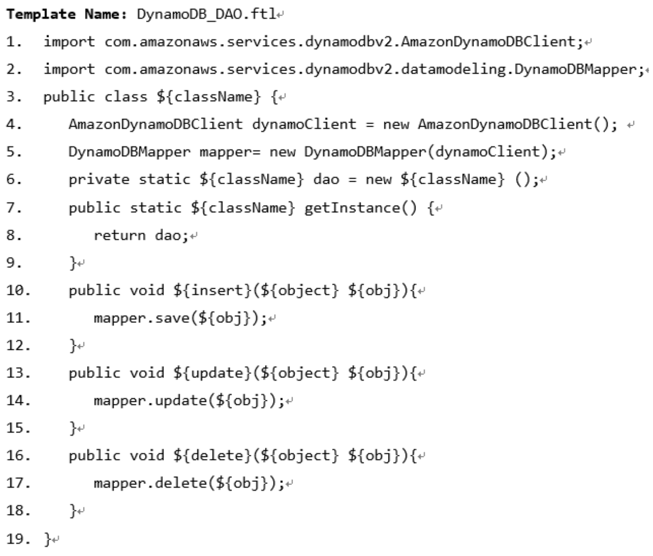
```
Pattern Name: hibernateDao.pat
1.    hibernateDao = (#   $*
2.        class $className extends HibernateDaoSupport {   $*
3.            (# void $insert(# $/.*insert.*/$ #)($object $obj){ $* } #)   $*
4.            (# void $update(# $/.*update.*/$ #)($object $obj){ $* } #)   $*
5.            (# void $delete(# $/.*delete.*/$ #)($object $obj){ $* } #)   $*
6.    }   #);
```

Figure 4. The pattern "hibernateDao" for matching database access oprations.

D. *Step 3: Template Design*

After locating the source code, a template-based method is proposed to generate the target code from the source code. For each target environment, the programming languages, design styles and programming interfaces are different. A template can help the engineers to control the styles of target source code.

The template is composed of variable and invariable elements. The invariable elements are the code text that will appear in the target code. The variable elements will be replaced by the inputs of the templates. As in Figure 5, this is a target code template for database access of AWS *DynamoDB*. In the example, the import statements are used as invariable elements, since the target code introduces new types that are not in source code. The "${..}" is for the variable elements which will be filled with the extracted elements from source code during the pattern matching phase. Besides the single elements, a collection of variables or the user-defined methods can also be used in the template.



```
Template Name: DynamoDB_DAO.ftl
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
public class ${className} {
   AmazonDynamoDBClient dynamoClient = new AmazonDynamoDBClient();
   DynamoDBMapper mapper= new DynamoDBMapper(dynamoClient);
   private static ${className} dao = new ${className} ();
   public static ${className} getInstance() {
      return dao;
   }
   public void ${insert}(${object} ${obj}){
      mapper.save(${obj});
   }
   public void ${update}(${object} ${obj}){
      mapper.update(${obj});
   }
   public void ${delete}(${object} ${obj}){
      mapper.delete(${obj});
   }
}
```

Figure 5. The DynamoDB example for data access

E. *Step 4: Code Transformation*

Step 4.1 Design Transformation Rules

Transformation rules are designed by migration engineers to design a user-defined transformation strategy. The transformation rule is composed of a set of code operation commands, including "match", "replace", "insert", "delete", "copy", etc. The syntax of the commands is listed in Table III. These commands are the minimum unit of code transformation logic. In each transformation rule, a match command is invoked to locate the source code to modify. Then, another operation command is invoked to transform the located source code to target code by referring template. The details of the commands are:

- *Match*: the command is to locate the source code to modify, where *"PATTERN*" is the pattern name to refer. The parameter *on "ANNOTATION"* means only match the code with given "*ANNOTATION*". The "*ANNOTATION*" is added at code scanning step.
- *Replace:* the parameter *"MATCH|VAR"* means the whole matched code or only the *VAR* part (e.g. $x) in the pattern will be replaced by the target code. The

Authorized licensed use limited to: University of Manchester. Downloaded on December 14,2021 at 16:43:20 UTC from IEEE Xplore. Restrictions apply.

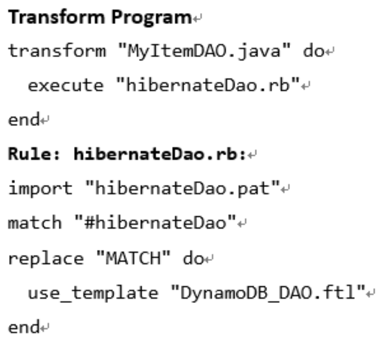
use_template means generating the target code with the following template.

- *Insert:* insert the target code into the given position. The parameter "into" is for the target file to insert. "*before* | *after*" is to give the exact location to insert. "*once* | *each"* means the command is executed once or for each matched code.
- *Delete*: delete the matched code. The parameter is similar as that of insert command.
- *Extract*: extract the selected code as a new method. The method name is provided with *"method_name*". The referred but not defined variables will be as the method parameters, and the modified variables will be packaged as the return object.
- *Move*: move the matched code to a new position, as absolute value or indirect value. The indirect value is determined by another pattern matching.
- *Copy*: similar as *Move*, except for keeping the original source code.
- *Create*: create a new file with "filename" and put into a given directory using "*to_dir*". The file can follow a template using "*use_template*" with a set of parameters through "*key=>value*".
- *Import*: import a pattern before referring it.

TABLE II. TRANSFORMATION COMMAND SYNTAX

Command	Description
Match	match "PATTERN" [do on "ANNOTATION" end]
Replace	replace "MATCH|VAR " do use_template "TEMPLATE" [{"KEY"=>"VALUE"}] end
Insert	insert [once|each] do into "SOUCECODE" before|after [first|last|each], "PATTERN" ,"VAR" use_template "TEMPLATE" end
Delete	delete "MATCH|VAR"
Extract	extract_method "MATCH|VAR" [do method_name "METHODNAME" end]
Move	move "MATCH|VAR" [once|each] do before|after [first|last|each] , "PATTERN" , "VAR" end
Copy	copy "MATCH|VAR" [once|each] do before|after [first|last|each] , "PATTERN" , "VAR" end
Create	create do filename "FILENAME" to_dir "DIR" use_template "TEMPLATE"[,{"KEY"=>"VALUE"}] parameters ({ {"KEY"=>"VALUE"} }) End
Import	import "PATTERN"

Figure 6 is an example of the transformation rule for hibernate DAO to Amazon DynamoDB, using the template in Figure 5. The *import* statement is to load the predefined pattern. The token "*MATCH*" means all the matched code should be modified. "*use_template*" means the operation will use a template to generate target code, and "replace" means the matched code will be replaced by the generated target code. The rule is invoked in the transformation program by "*execute rulename*".



```
Transform Program
transform "MyItemDAO.java" do
  execute "hibernateDao.rb"
end
Rule: hibernateDao.rb:
import "hibernateDao.pat"
match "#hibernateDao"
replace "MATCH" do
  use_template "DynamoDB_DAO.ftl"
end
```

Figure 6. Transformation rule for hibernateDAO to DynamoDB

Step 4.2 Execute Transformation

The transformation rules can be executed in single or batch mode. A domain-specific language (DSL) is designed using a programming language named Ruby to run the transformation rules. In a DSL file, one or more rules are invoked to execute transformation. Also, a DSL script file can import another script file to integrate more transformation rules. In this way, the transformation rules and script can be reused. The advantages of DSL-based scripts include:

- **Programmable**. With internal DSL design, the DSL script can be programmed flexible. No external parser is needed, neither too much learning efforts.
- **Reusable**. The transformation rules and scripts can be reused for changing similar code appearing in different projects or modules.
- **Hierarchical layers**. The transformation strategy is in three layers: command, rule and script. The higher layer is composed of lower layer.

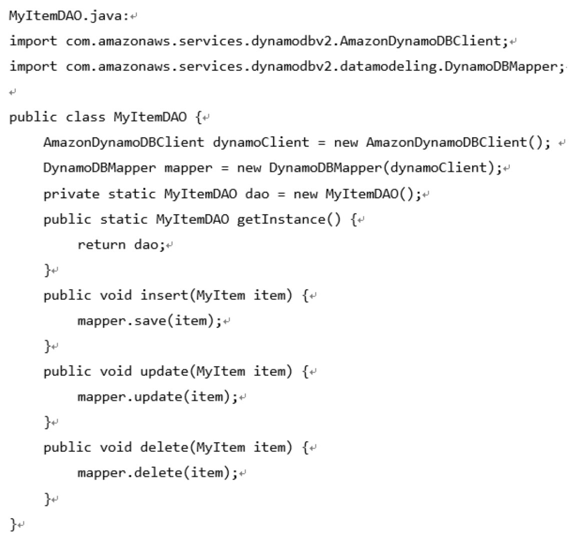
After executing the transformation rules, the results of Figure 1 is shown in Figure 7.

If the transformation results is accepted by the engineers, the transformed code will be committed to source code repository. Usually, manual edit is inevitable to modify the transformed code for syntax or semantics. After each transformation circle, the new pattern is applied for locating code to transform. When all the predefined patterns are applied, the transformation with our approach completes. For other code that needs to transform, manual transformation should be done or new patterns should be proposed to restart the transformation process.

F. *Target Code Validation*

The target code should be validated after transformation. It is very difficult to check the equivalence of two pieces of source code using formal methods. Back-to-back testing [21] is introduced here to make sure the target code works as the original system. Currently, there are many existing testing frameworks, so we do not discuss on the validation in detail here.

Authorized licensed use limited to: University of Manchester. Downloaded on December 14,2021 at 16:43:20 UTC from IEEE Xplore. Restrictions apply.



```
MyItemDAO.java:
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;

public class MyItemDAO {
    AmazonDynamoDBClient dynamoClient = new AmazonDynamoDBClient();
    DynamoDBMapper mapper = new DynamoDBMapper(dynamoClient);
    private static MyItemDAO dao = new MyItemDAO();
    public static MyItemDAO getInstance() {
        return dao;
    }
    public void insert(MyItem item) {
        mapper.save(item);
    }
    public void update(MyItem item) {
        mapper.update(item);
    }
    public void delete(MyItem item) {
        mapper.delete(item);
    }
}
```

Figure 7. The transformed DynamoDB code

G. *Tool Support*

We developed a transformation tool for validating the proposed approach. The system is composed of three critical modules, including analysis module, transformation module and interaction module. The analysis module is to locate the source code to transform, the transformation module is to do the transformation and the interaction module is to process the engineers' inputs.

The source code in change is managed with a code repository, where the code version is recorded after each iterative process. If the engineers want to cancel some modification, the configuration management can help to rollback to previous version. The input code features, patterns, transformation rules & scripts and code templates are adjusted and extended during the transformation process. Some machine learning algorithms may be applied to improve the input.

III. Experiments and Evaluation

A. *Experiment Setting*

We have collected 19 open source Java projects from the Internet as examples to demonstrate our approach, as in Table III. All these projects contain the code with storage or security services. All these projects have to be transformed when migrating to cloud according to existing analysis tools like [20]. For our experiments, we have collected the relevant data that record the migration of these projects onto the cloud: (1) the number of changes made to the database and file access operations, (2) the number of changes made to the security operations and (3) the number of hours required to make these changes manually. The projects cover a diverse range, from Java web projects to plugins to tools, etc. Although these projects serve different purposes, they all have storage and security services. Figure 8 summarizes these 19 projects.

TABLE III. The 19 Java projects used to demonstrate our Approach

Java Project	**Location**	**Access Date**
appassembler-maven-plugin	http://mojo.codehaus.org/appassembler/appassembler-maven-plugin/	May 20, 2014
Cbe	http://sourceforge.net/projects/cbe/	June 6, 2014
Freecs	http://freecs.sourceforge.net/	May 10, 2014
Freemarker	http://freemarker.org/	June 10, 2014
hd1-financeocr	https://bitbucket.org/hd1/financeocr	June 15, 2014
Jactiverecord	http://git.oschina.net/redraiment/jactiverecord	June 17, 2014
Jcvsweb	http://www.jcvs.org/	June 20, 2014
jetbrick-templat	http://git.oschina.net/sub/jetbrick-template	June 22, 2014
jfinal_91zcm	http://git.oschina.net/jianggege/jfinal_91zcm	June 29, 2014
JForum	http://jforum.net/	June 5, 2014
Jobo	http://www.matuschek.net/jobo/	July 10, 2014
Jportlet	http://jportlet.sourceforge.net/download.html	July 1, 2014
Jspider	http://j-spider.sourceforge.net/quick/download.html	July 1, 2014
maven-dotnet-plugin	https://bitbucket.org/grozeille/maven-dotnet/src/fdbf13460f71f1a63feb9ef6291f1f1062de0e12/maven/?at=default	May 20, 2014
netty-4.0.19	http://netty.io/downloads.html	July 12, 2014
Pooka	http://www.suberic.net/pooka/	July 20, 2014
QuickServer	https://github.com/QuickServerLab/QuickServer-Main/releases/tag/v2.0.0	July 15, 2014
Toughradius	http://git.oschina.net/jamiesun/toughradius	July 15, 2014
webit-script	http://git.oschina.net/zqq90/webit-script	July 25, 2014

We then applied our approach to transform each of these projects into a cloud environment. Amazon cloud has been chosen as our target cloud environment due to its maturity. *PaaSLane* [20] was used to perform the code scanning step. It detects the JDBC (Java DataBase Connection) related code, file access using java.File, and security related scenarios including session management, network protocol, encryption, etc.

B. *Experiment Results Evaluation*

All the detected warnings were modified manually by the students in our research group. And then we applied our approaches to execute the transformation. The transformation efforts comparison is given in Figure 9. The efforts of using our approach includes developing patterns, rules and templates. Compared with manual transformation, our approach reduces 60-90% cost. The efforts can be reduced further, if there are already some transformation patterns, rules and templates that can be reused.

C. *Analysis and Discussion*

Authorized licensed use limited to: University of Manchester. Downloaded on December 14,2021 at 16:43:20 UTC from IEEE Xplore. Restrictions apply.

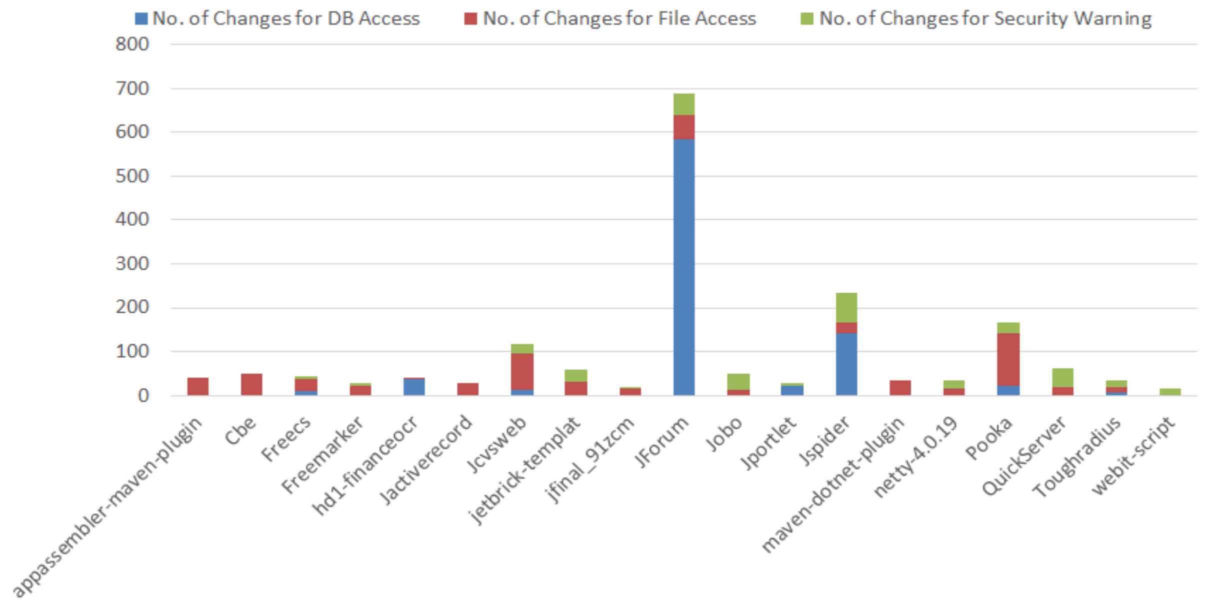


Figure 8. The no. of change warnings for DB, file and security

The experiment results show that our proposed approach can greatly improve the transformation efforts for migrating software applications on cloud. Our approach has three advantages:

- The repetitive transformation tasks can be processed using a general pattern and related rules.
- This is an iterative process. Only one or one type of transformation task is executed. Since it is nearly impossible to automatically transform a complete application, some manual review or edits are inevitable after executing each transformation.
- The experiments focused on the database access, file access and security warning. But the approach can be applied to variety of source code including code for business logic, if the patterns can be defined.

However, the experiments also meet the following potential threats that may block the experiment results.

- One general pattern for all cases or several patterns where each for some cases. Our approach supports any pattern following our pattern syntax. However, the more general, the more efforts should be taken to define. For example, it is too difficult to design one pattern matching all kinds of source code for database access, but it is more feasible to design a specific code pattern for each framework, like hibernate.
- The transformation cases. The selected projects may not cover all the kinds of code implementation. There may exist some scenarios that is not easy to design general pattern. For generality, all the projects were downloaded from internet.
- How to evaluate the manual efforts and do the comparison. For the experiment cases, the students in our group are separately arranged to analyze and modify the cases. The students doing manual modifications are different from those developing the transformation rules. In this way, both of them need comprehension time, which is more similar as the actual transformation environment.

The above analysis shows that our approach can not only be used to analyze the claimed three kinds of warnings, but also some other scenarios. It can be easily used and evaluated with iterative process. The transformation efforts are shifted from modifying the source code manually to defining patterns, transformation rules and templates. The developing efforts can be refined by introducing the machine learning for pattern recovery.

IV. Related Work

This section discusses closely related work on the cloud migration process, methods and supporting tools.

A. Migration Process

Many researchers have proposed their processes to migrate software systems to cloud. *Lv* et al [1] proposed a rapid cloud migration solution (RCMS) to improve the migration process and quality. *Rabetski* et al [2] migrated a document comparison system to the cloud. Based on the *MODAClouds* (MOdel-Driven Approach for design and execution of applications on multiple Clouds) EU project, *Gunka* et al [3] introduced an evolutionary approach for cloud migration. *Rowe* et al [4] introduced a migration process for the text-mining application. They proposed the benchmark for execution time that could be used to determine whether migration is meaningful. *Zhao* et al [10] divided the existing migration methods into three strategies, and compared the similarities and divergences among them. *Cretella* et al [11] proposed an overview of the migration approaches based on cloud patterns. However, all the processes aimed to migrate the whole applications onto cloud at once or focused on some transformation phases.

Authorized licensed use limited to: University of Manchester. Downloaded on December 14,2021 at 16:43:20 UTC from IEEE Xplore. Restrictions apply.

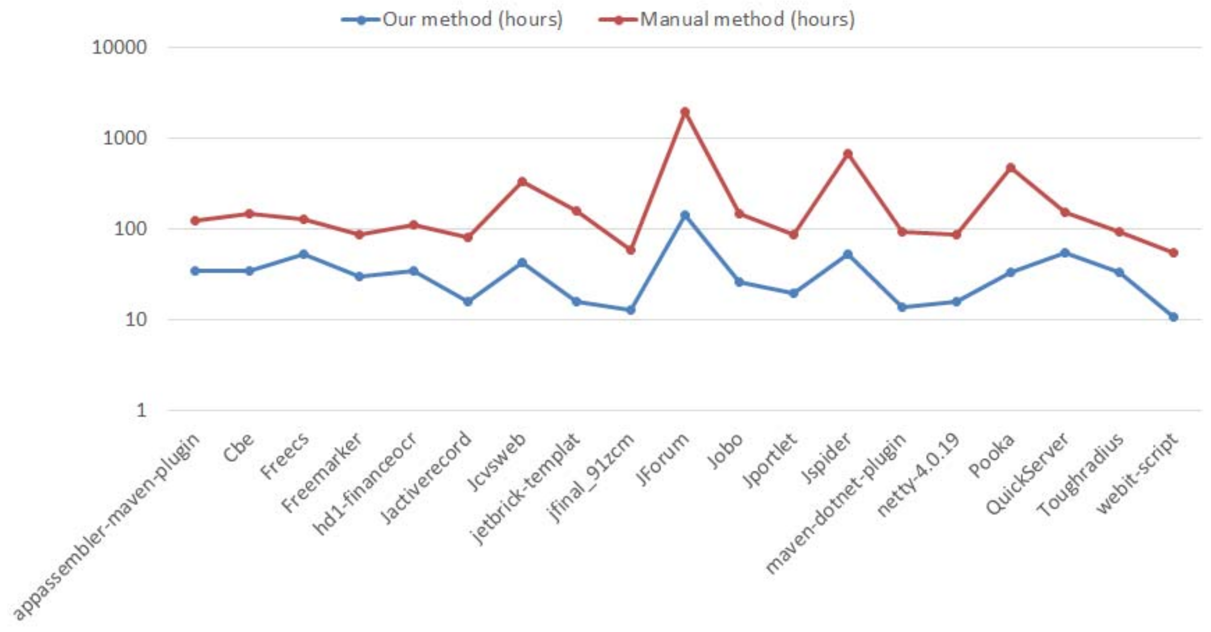


Figure 9. The efforts comparison of manual and our approach for the 19 projects

B. *Cloud Migration Methods*

Peddigari et al [5] proposed a unified cloud migration framework to drive the application development on cloud. The researchers claimed that the framework and factory based approaches can be treated as a systematic approach to migrate applications to cloud platform in a sustainable manner. The factory approach is composed of variety of teams to do and evaluate the migration. However, it also needs plenty of efforts for processing repetitive tasks.

Cardoso et al [6] presented and summarized several frameworks of real migration cases. They claimed a six steps framework – workload definition, workload suitability analysis, definition of alternatives, TCO (Total Cost Ownership) calculation, definition of criteria, and multi-criteria method application. *Wang* et al [7] claimed a paradigm based on BPR (Business Process Reengineering) and gBPR (Green BPR) to make the enterprises can be transformed successfully. Additionally, *Nicolas* et al [8] proposed a novel framework to help medium-sized enterprises (SMEs) to master migration related impediments. But all these frameworks or methods have little consideration on the efforts of the repetitive tasks.

Venugopal et al [9] presented a methodology to smoothly migrate a web service based enterprise application to the multicore cloud. It is to improve the performance of applications due to the context switch of multiple threads. This method is put forward to encourage the mass migrations of enterprise applications to the cloud. The approach is for specific target platforms and performance goals.

Ward et al [12] presented a framework Darwin that could be used to accomplish workload migration. Their framework was developed to accelerate heterogeneous source/target migrations to standard virtualized environments such as Linux/Linux or Windows/Linux. The framework is exciting, but it considers little on the fine-granularity transformation to utilize cloud characteristics.

C. *Cloud Migration Tools*

Khajeh-Hosseini et al [17] proposed two tools to support decision making during the Cloud migration. One is a modeling tool that can estimate the cost of using public *IaaS* (Infrastructure as a Service) clouds. The other is a spreadsheet that outlines the benefits and risks of using *IaaS* clouds. The tools have been applied in the migration of a digital library, a searching engine and a R&D division of a media corporation. They claimed the tools can help decision-makers to model their computational resource usage patterns, as well as the benefits and risks of the cloud migration. *Andrikopolos* et al [13] looked into the different prices of different cloud service providers. They focused on designing and developing a migration decision support system that can address the issues of vender selection and cost calculation. Their system can match the most qualified cloud service provider according to user-provided requirements. There tools focus on helping engineers to do the migration decision, not for the transformation.

Khajeh-Hosseini et al [15] investigated and discussed the benefits and risks in the enterprises use cloud computing by involving a fieldwork at an IT solutions company who was migrating their systems to cloud. They point out that cloud computing is a significantly cheaper compared to purchasing system infrastructure. They also develop a collection of tools called the Cloud Adoption Toolkit to help migrating their application to cloud. But the toolkit is also for decision making for adoption of cloud migration.

Meng et al from the NEC Laboratories China proposed a new Application Migration Solution (AMS) to migrate legacy applications to web applications efficiently [16]. The

Authorized licensed use limited to: University of Manchester. Downloaded on December 14,2021 at 16:43:20 UTC from IEEE Xplore. Restrictions apply.

core technology of AMS is GUI recognition and reconstruction technology. AMS can recognize and extract the GUI information to generate HTML templates. Then AMS fills HTML templates with user action description data to generate the final webpage. AMS has been validated with several business level legacy applications and it has delightful results. However, it focused on the GUI reconstruction and considered little on the service layers.

Stratego/XT [18] provides a rule-based transformation strategy and tool to transform source code for refactoring and translation. The tool uses ast-like (Abstract Syntax Tree) pattern matching iteratively to locate source code. The solution has been implemented as Eclipse plugins. When doing transformation, the rules and strategies should be designed as well as the code patterns. Also, it considers little of code context, so it's difficult to execute transformation with code semantics. Besides, some commercial venders like Semantic Design [19] claims on user-specific transformation solutions. They claimed many successful cases on code translation. However, for the code transformation, they only provide customized services without general solutions.

V. CONCLUSIONS

Cloud migration has become an important research topic, as more and more enterprise applications are migrating to cloud platforms and services. One common migration task is to modify the source code and most of the transformation solutions perform this tasks manually. This paper proposed a pattern-based transformation approach. In this approach, the scanning techniques are used to locate source code for transformation, pattern matching is used to locate the source code, templates are used to generate target code, and the transformation rules are designed for flexible programmable transformation. A software tool is developed to support this approach. According to the experimental results of migrating 19 Java projects to AWS, the proposed approach can save 60%-90% efforts for performing the repetitive tasks of modifying the storage and security related code. These tasks are commonly performed in most of software applications.

One limitation of our approach is that it needs to develop a large number of patterns and templates for different situations before performing the migration. Once enough patterns and templates have been designed, the existing patterns and templates can be reused. So the development efforts can be reduced further with these reusable patterns.

ACKNOWLEDGEMENT

The work in this paper was supported by the National Basic Research Program of China (973 Project No. 2011CB302603), National Key Technology R&D Program of the Ministry of Science and Technology of China (Project No. 2013BAH01B01), and Ningbo Natural Science Foundation (Grant No. 2014A610071).

REFERENCES

[1] H. Lv, and J. Liu, "RCMS:Rapid Cloud Migration Solution," Proc. Symp. on Computational Intelligence and Design (ISCID' 13), Vol. 1, Oct. 2013, pp. 426-429.

[2] P. Rabetski, and G. Schneider, (2013). "Migration of an On-Premise Application to the Cloud:Experience Report," Service-Oriented and Cloud Computing, Springer Berlin Heidelberg, 2013, pp. 227-241.

[3] A. Gunka, S. Seycek, and H. Kühn. "Moving an application to the cloud: an evolutionary approach," Proc. workshop on Multi-cloud applications and federated clouds, 2013, pp. 35-42.

[4] F. Rowe, J. Brinkley, and N. Tabrizi. "Migrating Existing Applications to the Cloud," Proc. In Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference, Dec. 2013, pp. 68-77.

[5] B. P. Peddigari. "Unified Cloud Migration Framework — Using factory based approach," Proc. Annual IEEE in India Conference (INDICON'11), Dec. 2011, pp. 1-5.

[6] A. Cardoso, F. Moreira, and P. Simões. "A Survey of Cloud Computing Migration Issues and Frameworks," In New Perspectives in Information Systems and Technologies, Volume 1, Springer International Publishing, 2014, pp. 161-170.

[7] H. I. Wang, and C. Hsu. "The paradigm framework of cloud migration based on BPR and gBPR," Proc. Conf. on Awareness Science and Technology and Ubi-Media Computing (iCAST-UMEDIA'13), Nov. 2013, pp. 460-465.

[8] N. Nussbaumer, and X. Liu, (2013, July). "Cloud Migration for SMEs in a Service Oriented Approach," Proc. Conf. on Computer Software and Applications Conference Workshops (COMPSACW'13), Jul. 2013, pp. 457-462.

[9] S. Venugopal, S. Desikan, and K. Ganesan. "Effective migration of enterprise applications in multicore cloud," Proc. Conf. on Utility and Cloud Computing (UCC'11), Dec. 2011, pp. 463-468.

[10] J. F. Zhao, and J. T. Zhou. "Strategies and methods for cloud migration," International Journal of Automation and Computing, vol. 11, no. 2, 2014, pp. 143-152.

[11] G. Cretella, and B. Di Martino. "An Overview of Approaches for the Migration of Applications to the Cloud," In Smart Organizations and Smart Artifacts, Springer International Publishing, 2014, pp.67-75.

[12] C. Ward, N. Aravamudan, K. Bhattacharya, K. Cheng, R. Filepp, R. Kearney, and C. C. Young. "Workload migration into clouds challenges, experiences, opportunities," Proc. Conf. on Cloud Computing (CLOUD'10), Jun. 2010, pp. 164-171.

[13] V. Andrikopoulos, Z. Song, and F. Leymann. "Supporting the migration of applications to the cloud through a decision support system," Proc. Conf. on Cloud Computing (CLOUD'13), Jun. 2013, pp. 565-572.

[14] M. A. Chauhan, and M. A. Babar, (2011, July). "Migrating service-oriented system to cloud computing: An experience report," Proc. Conf. on Cloud Computing (CLOUD'11), Jul. 2011, pp. 404-411.

[15] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville. "Cloud migration: A case study of migrating an enterprise IT system to IaaS," Proc. Conf. on Cloud Computing (CLOUD' 10), Jul. 2010, pp. 450-457.

[16] X. Meng, J. Shi, X. Liu, H. Liu, and L. Wang, (2011, July). "Legacy application migration to cloud," Prof. Conf. on Cloud Computing (CLOUD'11), Jul. 2011, pp. 750-751.

[17] A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, & P. Teregowda. "Decision support tools for cloud migration in the enterprise," Prof. Conf. on Cloud Computing (CLOUD'11), Jul. 2011, pp. 541-548.

[18] Stategoxt, http://strategoxt.org/Stratego/WebHome, avaiable at March 5, 2014

[19] SemanticsDesign, http://www.semanticdesigns.com/Products/DMS/DMSToolkit.html, available at July 2, 2014

[20] PaaSLane, http://www.paaslane.com/, available at April 20, 2014

[21] B. Berizier, "Software Test Techniques", Second Edition, Van Nostrand Reinhold, New York, 1990.

Authorized licensed use limited to: University of Manchester. Downloaded on December 14,2021 at 16:43:20 UTC from IEEE Xplore. Restrictions apply.

View publication stats