Source Code Pattern Recognition using Machine Learning

1. Foreword

Being the most popular public cloud service, Amazon AWS has become a de facto standard for many enterprises to migrate their business applications. However, migrating systems manually incurs huge amount of labor cost, project risk and uncertainties. More works have been done on migrating an entire system instance to AWS, but very few efforts are taken on the source code level transformation. We propose an automated solution to migrate system to AWS with source code transformation. The solution is flexible, language and platform independent. In this solution, a code analyzer (e.g. PMD / FindBugs) is used with pre-defined source code pattern to locate the code segments that need to be transformed. The code segments will be annotated accordingly with association to specific targeted output code patterns. Based on those code templates setting, transformation will take place automatically with options to replace, insert or remove source codes using a domain specific programming language.

2. The Challenges of Automated Code Transformation

How to locate the right source code segment A good code analyzer is a pre-requisite to locate the appropriate code segments that violate the AWS rules. Invalid identification may lead to wrong subsequent transformation. Code pattern matching can bring precise results, but may not cover all the related codes. Using key words search may cover all the related codes, but it is difficult to extract code info like variables, constants etc. which may be needed in transformation.

Different implementation methods for the same function For the same business logic, different developers may bring in different design and implementation styles. Programming against enterprise code standards will help standardization and make code pattern recognition easier. Further enhancement to enable repetitively code pattern recognition will require machine learning algorithms.

Manual effort is still needed It is nearly impossible to implement an entire system

transformation automatically. The degree of automation is largely depended on the code quality and recognized patterns. It will be less economical to transform badly written codes, which do not follow best practices or standards.

3. Transformation Solution

We propose an automatic transformation solution based on pattern matching, as shown in Fig.1. The transformation is an iterative process. In each iterative step, one or more patterns are used to drive the transformation. Firstly, the source code is analyzed using a code analyzer or manually. Annotations are added into the source code to describe the functions or structures. Then, the users can design exact patterns to match different code styles and extract essential information from the code itself. The extracted information would be used in the following transformation process. Since different programmers may have different preferences to implement the same function, designing one single pattern to cover all the scenarios seems impossible. Thus, several patterns can be designed and keep expanding based on manual experiences, which can then be saved into a pattern repository for future project use. Sometimes, it is too difficult to define a pattern and throw an exception for all the scenarios. Manual transformation will become an alternative.

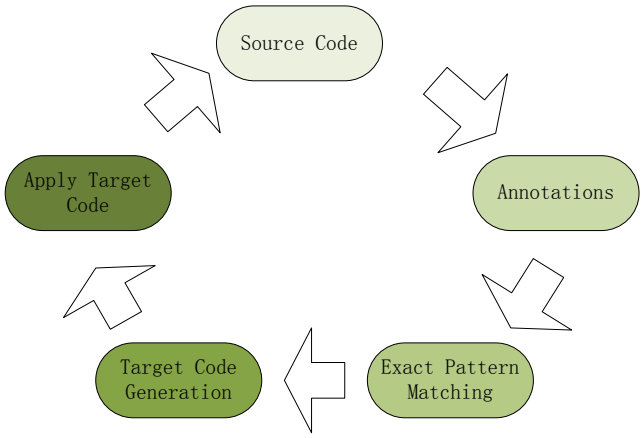

Fig .1 Transformation work flow

The target code generation in Fig.1 transforms the matched source code to the target code using code templates defined from AWS. Code template consists of text, variables and control characters. Variables with a prefix $ could be the same as the parameters in transformation model. During the transformation, the variables would be replaced by the values from transformation model. The conditional and loop logic can be defined using the character “#”. Following are two examples:

For control:

#i *f (condition)*
... ...
#else
... ...
#end

For loop:

#forEach (collection)
... ...
#end

Finally, the generated target code would be applied to update the system code. The target code can be used to insert into the source code, replace the source code or update part of the source code. After the target code applied, the source code is ready for next transformation iteration.

4. Transformation Cases

Here we introduce two transformation cases. Each case will be described from problem description, source code, pattern for transformation, transformation actions, template and target code.

Case 1: DAO Transformation to MyBatis

- **Problem Description**

Use Object Relational Mapping options like Hibernate or Mybatis as data access best practice, rather than using java.sql.DriverManager. I

- **Source Code Sample**

```
public UserTEO getUserById(Integer id)
        Connection con = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        UserTEO user = null;
        String sql="SELECT ID, NAME FROM USERINFO WHERE ID = ? ";
        try {
            con = ConnDB.getConn();
            con.setAutoCommit(false);
            ps=con.prepareStatement(sql);
            ps.setString(1, id);
            rs=ps.executeQuery();
            con.commit();
            if(rs.next()){
                user = new UserTEO();
```

```
                user.setId(rs.getString("id"));
                user.setName(rs.getString("name"));
            }
        } catch (SQLException e) {
            try {
                con.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
            e.printStackTrace();
        }finally{
            rs.close();
            ps.close();
            con.close();
        }
        return user;

    }
```

- **Target Code Sample**

```
public UserTEO getUserById(Integer id, SqlMapClient sqlmapClient) {
       try
       {
           UserTEO user =
(UserTEO)sqlmapClient.queryForObject("user.getUserById", id);
           return user;
       }
       catch(Exception e)
       {
           e.printStackTrace();
       }
       return null;
   }
```

```
<sqlMap namespace="user">
    <typeAlias alias="UserTEO" type="com.howtodoinjava.ibatis.demo.dto.UserTEO"
/>

    <resultMap id="userResultMap" class=" UserTEO">
        <result property="id" column="ID" />
        <result property="name" column="NAME" />
    </resultMap>

    <select id="getUserById" parameterClass="java.lang.Integer"
resultMap="userResultMap">
         SELECT ID, NAME FROM USERINFO WHERE ID = #id#
```

```
    </select>
</sqlMap>
```

- **Pattern for Transformation**

```
Pattern Name: dao.pat
daoMatch=(#
$entity $methodName($type $para) {
    $*
    $sqlStr(# SELECT $* FROM $table WHERE $id #)
    $*
    (# $objName.$/set.*/$($.$/get.*/$("$var~")); #)+
    $*
    (# return $objName; #)
}

#);
```

- **Code Templates**

```
Template Name: MyBatis_DataAccess.ftl
${entity} ${methodName}(${type} ${para}, SqlMapClient sqlmapClient) {
    try {
        ${entity} ${objName} =
(${entity})sqlmapClient.queryForObject("${objName}.${methodName}", ${para});
        return ${objName};
    } catch(Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

```
Template Name: Mybatis_Conf.ftl
<sqlMap namespace = "${objName}">
    <typeAlias alias = "${entity} type = "${GetFullQualifiedName(filePath,entity)}"
/>

    <resultMap id = "${objName}ResultMap" class = "${entity}">
     <#list var as being>
         <result property = "${being}" column = "${being?upper_case}" />
        </#list>
    </resultMap>

     <select id = "${methodName}" parameterClass =
"${GetFullQualifiedName(filePath,type)}" reslutMap = "${objName}ResultMap">
         ${sqlStr}= #${id?lower_case}#
     </select>
</sqlMap>
```

- **Transformation Actions**

```
import "dao.pat"
match "#daoMatch"

replace "MATCH" do
   use_template "MyBatis_DataAccess.ftl"
end

create do
   filename "UserTEO.xml"
   use_template " Mybatis_Conf.ftl "
end
```

Case 2: DAO Transformation to SimpleDB

- **Problem Description**

Use SimpleDB in AWS instead of local databases

- **Source Code Sample**

Service.java:

```
public class Service {
     void save(){
      MyItemDAO dao = MyItemDAO.getInstance();
      MyItem item = new MyItem();
      item.setName("name");
      dao.insert(item);
     }
}
```

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){
        this.getHibernateTemplate().delete(item);
     }
}
```

```
MyItem.java:
     import javax.persistence.Column;
     import javax.persistence.Entity;
     import javax.persistence.GeneratedValue;
     import javax.persistence.Id;
     import javax.persistence.Table;
     import org.hibernate.annotations.GenericGenerator;

     @Entity
     @Table(name="SimpleTable")
     public class MyItem {
      private Integer id;
      private String name;

      @Id
      @GeneratedValue(generator = "system-uuid")
      @GenericGenerator(name = "system-uuid", strategy = "uuid")
      public Integer getId() { return id; }
      public void setId(Integer id) { this.id = id; }

      @Column(name = "NAME")
       public String getName() {return name;}
       public void setName(String name) {this.name = name; }
     }
```

- **Target Code Sample**

```
Service.java:
public class Service {
     void save(){
      MyItemDAO dao = MyItemDAO.getInstance();
      MyItem item = new MyItem();
      item.setName("name");
      dao.insert(item);
     }
```

```
}
```

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);
        }
}
```

MyItem.java:

```
    import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
    import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
    import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
    @DynamoDBTable(tableName="SimpleTable")
    public class MyItem {
        private Integer id;
        private String name;

     @DynamoDBHashKey(attributeName="Id")
        public Integer getId() { return id; }
        public void setId(Integer id) { this.id = id; }

     @DynamoDBAttribute(attributeName="Name")
     public String getName() {return name;}
     public void setName(String name) {this.name = name; }
    }
```

- **Pattern for Transformation**

Pattern Name: hibernateDao.pat

```
hibernateDao = (#
$*
class $className extends HibernateDaoSupport {
      $*
    (# void $insert(# $/.*insert.*/$ #)($object $obj){ $* } #)
      $*
    (# void $update(# $/.*update.*/$ #)($object $obj){ $* } #)
      $*
    (# void $delete(# $/.*delete.*/$ #)($object $obj){ $* } #)
      $*
}
#);
```

Pattern Name: hibernateEntity.pat

```
hibernateEntity = (#
      $*
      @Entity
      @Table(name="$table")
      public class $className{
            $*
            @Id
            $*
            (# public $idType $/get.*/$ () {
                  return $id;
               }
            #)
            $*
            (# @Column(name = "$name")
                  public $attrType~ $/get.*/$(){
                        return $attr~;
                  }
                  public void $/set.*/$($ $attrName){
                  this.$ = $attrName;
                  }
            #)+
            $*
      }
#);
```

- **Code Templates**

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});
 }
}
```

Template Name: DynamoDB_Entity.ftl

```
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAttribute;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;

@DynamoDBTable(tableName="${table}")
public class ${className} {
    private ${idType} ${id};
    <#list attrType as being>
    private ${being} ${attr[being_index]};
    </#list>

    @DynamoDBAttribute(attributeName="${id?cap_first}")
    public ${idType} get${id?cap_first}() {return ${id};}
    public void set${id?cap_first}(${idType} ${id}) {
        this.${id} = ${id};
    }

    <#list attrType as being>
    @DynamoDBAttribute(attributeName="${attr[being_index]?cap_first}")
     public ${being} get${attr[being_index]?cap_first}() {return
      ${attr[being_index]};}
     public void set${attr[being_index]?cap_first}(${being}
       ${attr[being_index]}) {
        this.${attr[being_index]} = ${attr[being_index]};
    }
    </#list>
```

```
}
```

- **Transformation Actions**

Script Action:

```
transform "MyItemDAO.java" do
  execute "hibernateDao.rb"
end

transform "MyItem.java" do
  execute "hibernateEntity.rb"
end
```

Rule Action:

hibernateDao.rb:

```
import "hibernateDao.pat"

match "#hibernateDao"

replace "MATCH" do
  use_template "DynamoDB_DAO.ftl"
end
```

hibernateEntity.rb:

```
import "hibernateEntity.pat"

match "#hibernateEntity"

replace "MATCH" do
  use_template "DynamoDB_Entity.ftl"
end
```

5. Highlights of the Solution

The transformation solution has the following highlights:

1. Language and platform independent. The transformation solution uses patterns to extract essential information from the source code and generate target code using the target code templates. The matching engine and transformation executor are designed with text mining technique and is highly extendable. Thus, it can be language and platform independent.
2. Flexible patterns. Instead of using general patterns, the solution provides

domain-specific patterns and pattern schema. The users can customize the patterns or redefine special pattern. This is a trade-off between the complexity of general patterns and the reusable of domain patterns. By repetitively scanning different source codes and update the pattern repository, the engine itself can learn and keep improving its efficiency and accuracy.

3. Directed target transformation process. The target architecture is represented in a set of code templates, which contains the structure and styles of the target code. For different platforms or systems, the users can either define different templates for customization or define the same code templates for consistence.
4. Exception handling for robustness. Complete automation is nearly impossible but a high degree of automation with thrown exceptions will definitely speed up the entire transformation process. It is recommended that an IDE (e.g. Eclipse plug-in) can be integrated with the above features to facilitate both automated and manual coding effort.
5. Greatly cut the transformation efforts. Our solution provides DSL-based strategy design for automating batch transformation tasks. This can greatly cut the efforts for transforming repetitive code.