The Transformation Algorithm for Query Mechanisms of Hibernate Framework

Tisinee Surapunt, and Chartchai Doungsa-ard

***Abstract*—**The collaboration work is an important key for working in software engineering field. In a team should have developers whom have difference abilities. To achieve goal on time and budget, they should select an appropriate tool which conform to their experience. Then the time consumption for learning and understanding will be decreased. However, some projects are depended on the legacy development. The project manager cannot determine the favorite tools and techniques. So, in the study propose a transformation algorithm which can help developers spend less time to get used to with the new technique. The experiment focused on the query mechanisms of Hibernate Framework. There are variety of query mechanisms which are Criteria API, HQL and Native SQL. The experiment is concerned only Criteria API and HQL which present the ability of conversion syntax. After that, to measure the efficiency and validate of function, the back to back testing is required. The retrieved data between both methods have to compare. The results showed the equivalence queried data. Thus, the transformation algorithm can covert those statements correctly.

***Keywords*—**Transformation Algorithm, Hibernate Framework, Criteria API, HQL

I. Introduction

Many software project managers emphasize that the communication and collaboration in a team are important. Because of the difference ability(s) of each person can take responsibility(s) on their own work. Then the projects can be impelled successfully. However, not only the conversation and team work can drive the projects finish, but other factors are also relevant. In fact, more than 70% of the projects might be faced with failures, over budget and delivered late [1]. So, software project management is needed for control and predict the possible risks which occur and affect to the project outcomes [2].

Hibernate Framework is one example which shows team working in software development. The framework provides 3 types of query mechanisms; HQL(Hibernate Query Language), Criteria API and native SQL. Each mechanism represents the difference expression pattern and retrieving performance. When the data is retrieved, the performance is measured by the execution time. Thus, each query method is appropriate and works properly in the distinct projects' environment. There are many elements which affect to the performance; WHERE condition, amount of data in the database, amount of query records set [3]. Ideally, the developers should understand all query patterns for selecting an appropriate method with the projects. But they are not perfect. So, the study would like to tune up the working style gets along well with the application performance by presenting the transformation algorithm.

The study, an algorithm to transform the expression from Criteria API to HQL is focused. The Criteria API works with the persistence objects and query as objects. Thus, most of the developers can develop and understand the structure. For the function of HQL which works similar as SQL and query records set as entity. The functionality of a transformation algorithm will be helped the developers to convert the expression easily. So, the developers do not waste time to understand and learn the structure of the syntax in another query technique. In section II, the literature reviews and related works are discussed. The transformation algorithm is presented in section III. The preliminary results and case studies are described in section IV. Finally, section V is the conclusion of an experiment and future work.

II. Literature Reviews

A. Hibernate Framework

Hibernate Framework is an ORM technique which can apply easily with database application. The framework is related with persistence objects which can map to the relational entities [4]. Most of developers develop applications using object oriented technique. Some of them retrieve the data direct through JDBC. They aware to face with problems when they try to map persistence objects to relational entities. The problem is called impedance mismatch. There is a possibility which persistence objects map with difference relational entities [5]. The architecture of Hibernate Framework is divided into 3 layers as shown in Fig. 1 which are Presentation Tier, Business Logic Tier and Database Access Tier. The Presentation Tier is the interface to the application. The entity objects relevant to the application are sent to the application. The Business Logic Tier expresses the mapping between object in the presentation tier and data recorded in the database. Finally, the Database Access Tier is manipulate the data for collecting in the database [6].

Tisinee Surapunt is with the Software Engineering, College of Arts Media and Technology, Chiang Mai University, Chiang Mai 50200 Thailand (corresponding author's phone: 084222532; e-mail: tsurapunt@gmail.com).

Chartchai Doungsa-ard, is with the Software Engineering, College of Arts Media and Technology, Chiang Mai University, Chiang Mai 50200 Thailand (e-mail: c.doungsaard@gmail.com).

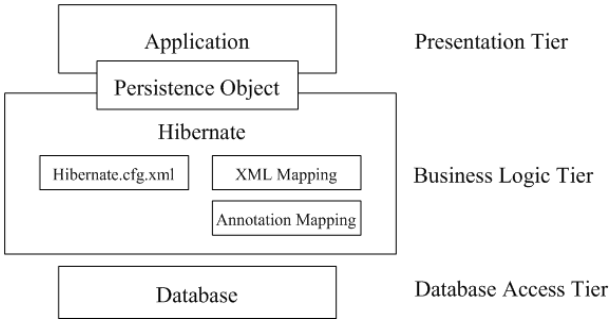

Fig. 1 The Architecture of Hibernate Framework

The Hibernate Framework reduces programmers' work load to manage the database. Moreover, the framework provides the powerful efficiency to manipulate with the data. After the persistence mapped with relation models successfully, the data entities can also create, retrieve, update and delete by objects easily.

B. The performance comparison between Criteria API and HQL query

The difference query mechanisms of Hibernate Framework are shown a difference performance. Each method can work properly on difference environment. The factors which impact the query performance are number of data in the database, number of retrieved data, condition of an expression. Thus, Tisinee Surapunt and Chartchai Doungsa-ard studied about the performance of each query mechanism when the data are retrieved [3]. In the experiment, the same conditions are defined to test both mechanisms which are number of data in the database and number of queried data in each execution time. The measurement is execution time which record while retrieving data. During the querying data process, there are 2 groups is observed. The first is small query group which means the data between 100 - 1,000 records set are queried. The second is large group which means the data between 1,000 - 30,000 records set are queried. Then, the result showed that both sizes of query group are affected to query performance of each method. The small group work properly with both mechanisms. The percentage of difference is nearly to zero. However, the large group work smoothly with the Criteria API method. It is suitable to query data with large number of data. Moreover the percentage of difference is significant number. Thus, the developers can choose the appropriate query method to work with project correctly. It can be helped to reduce the developing time to run their application.

C. The Algorithm design for transformation

The transformation algorithm is well-known in the database application field. Many developers have to responsible in many projects. Sometime they would like to maintain applications while spend less time. So, There are algorithms represent their function which can convert the syntax of an expresstion. Hao Jiang, Liwei Ju and Zhuoming Xu [7] studied about the relational database design and Hibernate Framework for working with semantic websites. Normally, the semantic websites use query command which name is SPARQL. The command is understood by RDF graph which is base on XML language. Thus, the researchers demonstrated the transform algorithm which can change SPARQL expression to HQL. After that, the websites can work together with Hibernate Framework. The researcher's team developed the steps to convert SPARQL to RDF OWL format. Then, the transformation is considered to work with Hibernate Framework. The fundamental function of framework can map entities to be objects. They changed SPARQL model to be java classes under OWL_DL rule. OWL_DL is similar to XML language which separate to be child node. So, the Faked RDF View Tree (FRVT) algorithm is created which shown in Fig. 2

```
IN: root of the FRVT tree
OUT: HQL
FRVT (root)
   BEGIN:
      From += "root.val"
    For each child c of root {
        If c is a leaf {
          If c in select vars {
             Select += c.value;
             Where += "and c.value is not null"
          }else if c in filter vars{
           Where += "and c.filter"
          }else{
           Where += "and c.value is not null"
        }else {
         FRVT(c);
        }
    }
   END
```

Fig. 2 Faked RDF View Tree Algorithm (FRVT)

The efficiency of the transformation algorithm between SPARQL to SQL is studied by Jiseong Son, Dongwon Jeong and Doo-Kwon Baik [8]. The research represented the query performance in difference data storage of RDF structure which are Triples, Jena and Sesame. After that, the result is compared the cost of modification and verification of each storage. The query syntaxes are in SPARQL expression forms. The expressions need to convert to be SQL statement before retrieving from the database. However, in fact each storage use difference algorithm for converting expressions. In the experiment, the researchers selected Chebotko's Algorithm for all data storage. Then, the result shown that Jena and Sesame have to add Triple structure for working with Chebotko's Algorithm. The number of query records set represented the same amount of all storage. So, the data storages do not need to work with specific transformation algorithm because developers can modify the structure of storage. It can be helped to reduce cost of modification and verification.

III. TRANSFORMATION ALGORITHM

Because software engineers are focused on working as a team, so it is possible that the developers are familiar with difference techniques. If the existing project use unfamiliar

methods with their abilities, the time consumption can be affected to the developing time. Some developers take lots of time to learn and practice the new technique even if they have fundamental knowledge. Therefore, the transformation algorithm is designed for reducing jobs' of database application developers. The algorithm's functional can convert an expression from Criteria API to HQL. When both mechanisms work, the execution times are difference significantly. There are many experiments to test query performance of each query method [3, 9, 10]. Many researchers tested the performance of query mechanisms. The environment is defined under conditions which are number of data in the database, number of query records set, WHERE condition of the expressions. Then, the results showed that each method can work properly in difference environment. In the study, the expression syntax of Criteria API is broken down and transformed into a graph which is called Control Flow Graph (CFG). The Control Flow Graph is one representation technique of data flow analysis. The graph is useful for compiler optimization to software engineering. The information is described clearly through the picture. Every nodes and edges might be traversed during the execution [11].

Criteria API syntax + java grammar → Control Flow Graph

Fig. 3 The step to convert Criteria API syntax to control flow graph

As fig. 3 is shown, the Criteria API syntax is needed to transform to be a control flow graph. The developers can understand the structure of syntax clearly and easy to manipulate with their values. Because the structure of the expression is similar to java source code, the developers use a java grammar to establish the graph. The transformation algorithm supported base cases which are from the Beginning Hibernate textbook [12]. The expressions contain 8 cases which are shown in table.1; query without WHERE condition, query with restriction, paging, unique result, ordering result, row counting, projection and aggregation.

TABLE I
EXPRESSIONS PATTERN OF CRITERIA API AND HQL

Name	**Criteria API**	**HQL**
Query Simple	Criteria crit = session.createCriteria(Product.class);	Query query = session.createQuery("from Product");

Restriction	Criteria crit = session.createCriteria(Product.class);
	crit.add(Restrictions.gt("price", new Double(500.00)));
	crit.add(Restrictions.like("name", "Home%"));
		Query query = session.createQuery
		("from Product where price > 500.00 and name like 'Home%'");

Paging	Criteria crit = session.createCriteria(Product.class);
	crit.setFirstResult(1);
	crit.setMaxResults(2);
		Query query = session.createQuery("from Product");
		query.setFirstResult(1);
		query.setMaxResults(2);

Unique	Criteria crit = session.createCriteria(Product.class); crit.setMaxResults(1); Product product = (Product) crit.uniqueResult();	Query query = session.createQuery("from Product"); query.setMaxResults(1); Product product = (Product) query.uniqueResult();

Order	Criteria crit = session.createCriteria(Product.class);
	crit.add(Restrictions.gt("price", new Double(500.00)));
	crit.addOrder(Order.desc("price"));
		Query query = session.createQuery
		("from Product p where p.price>500.00 order by p.price desc");

Row count	Criteria crit = session.createCriteria(Product.class); crit.setProjection(Projections.rowCount());	Query query = session.createQuery ("select count(*) from Product product");

Projection	Criteria crit = session.createCriteria(Product.class);
	ProjectionList projList = Projections.projectionList();
	projList.add(Projections.property("name"));
	crit.setProjection(projList);
		Query query = session.createQuery
		("select product.name from Product product");

Aggregate	Criteria crit = session.createCriteria(Product.class);
	ProjectionList projList = Projections.projectionList();
	projList.add(Projections.min("price"));
	crit.setProjection(projList);
		Query query = session.createQuery
		("select min(price) from Product");

However, the base cases did not include last 2 patterns; projection and aggregation. Both concepts are focused on Database Management System (DBMS) which concern only attributes not objects. So, the algorithm supports only 6 cases. There are the examples of Criteria syntax as shown in Fig. 4. and Fig. 6. The Criteria API syntax is similar to Object Oriented Programming technique. Most of developers can understand and work with this kind of syntax easily. The query syntax structure uses object as an instance. The methods are called for using every functions. Then the expected result is the HQL expression syntax. In Fig. 4 is a Criteria API expression without condition. The database application retrieves all data records from Product entity.

```
Criteria crit = session.createCriteria(Product.class);
```

Fig. 4 The example input of Criteria API without condition expression

Moreover, in Fig. 5 is the control flow graph which is generated from Criteria API expression with java grammar.

The graph can be helped the developers understand the structure of syntax easily.

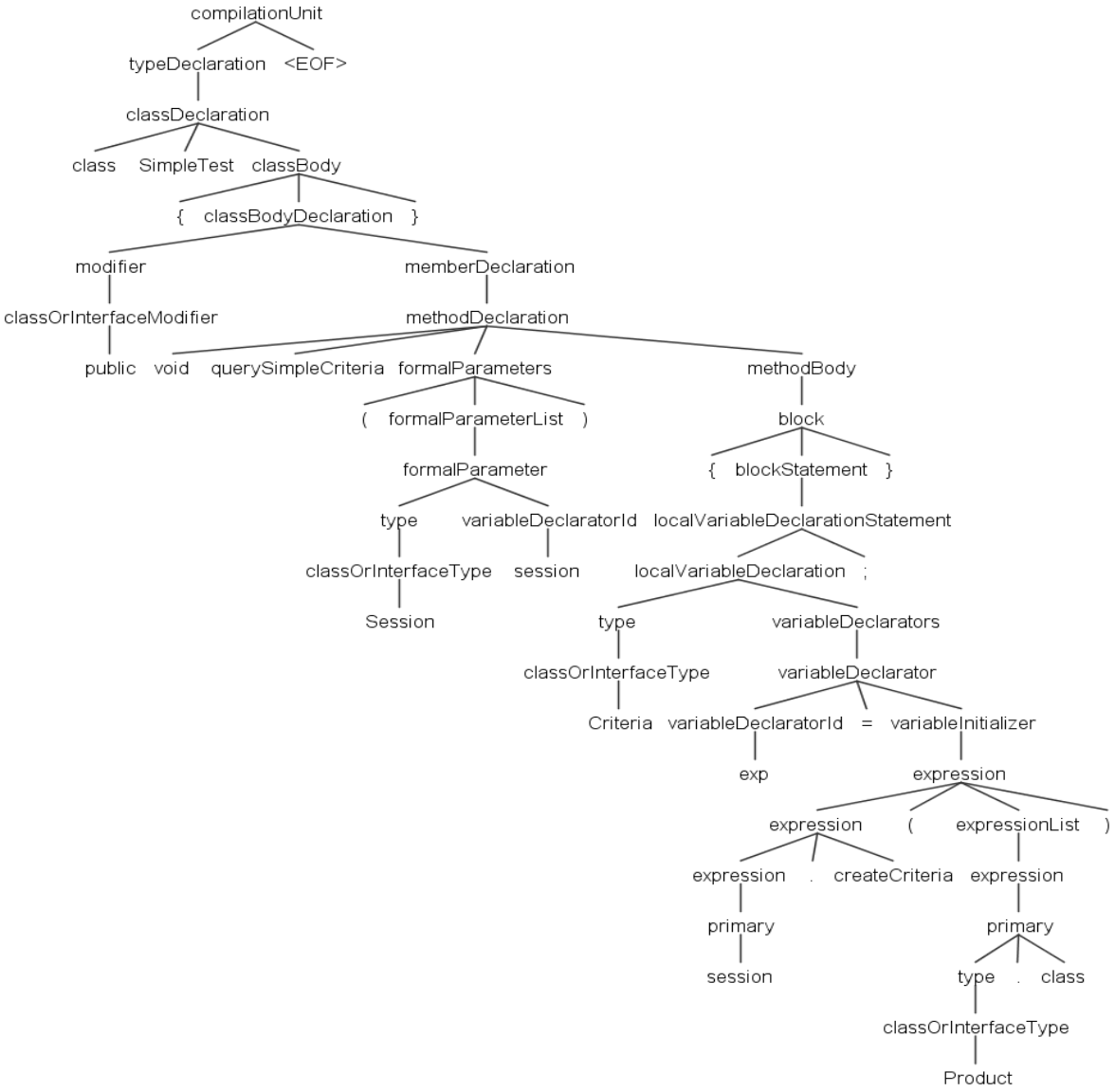

Fig. 5 The CFG from Criteria API expression without condition

After the developers got the control flow graph, the HQL syntax is the result of the conversion. The converted syntax is shown in Fig. 6. The developers notice the changed value from the structure of the control flow graph.

```
Query query = session.createQuery("from Product");
```

Fig. 6 The example output after converted Criteria API to HQL expression

Moreover, the expression with condition can show the retrieved data as the following defined restrictions in Fig. 7.

```
Criteria crit = session.createCriteria(Product.class);
crit.add(Restrictions.gt("price", new Double(500.00)));
crit.add(Restrictions.like("name", "Home%"));
```

Fig. 7 The example output after converted Criteria API to HQL expression

In order to make more understand with conversion, the control flow graph is generated as shown in Fig. 8. Both CFGs look like a tree graph which can walk and find the node's value by depth first search method.

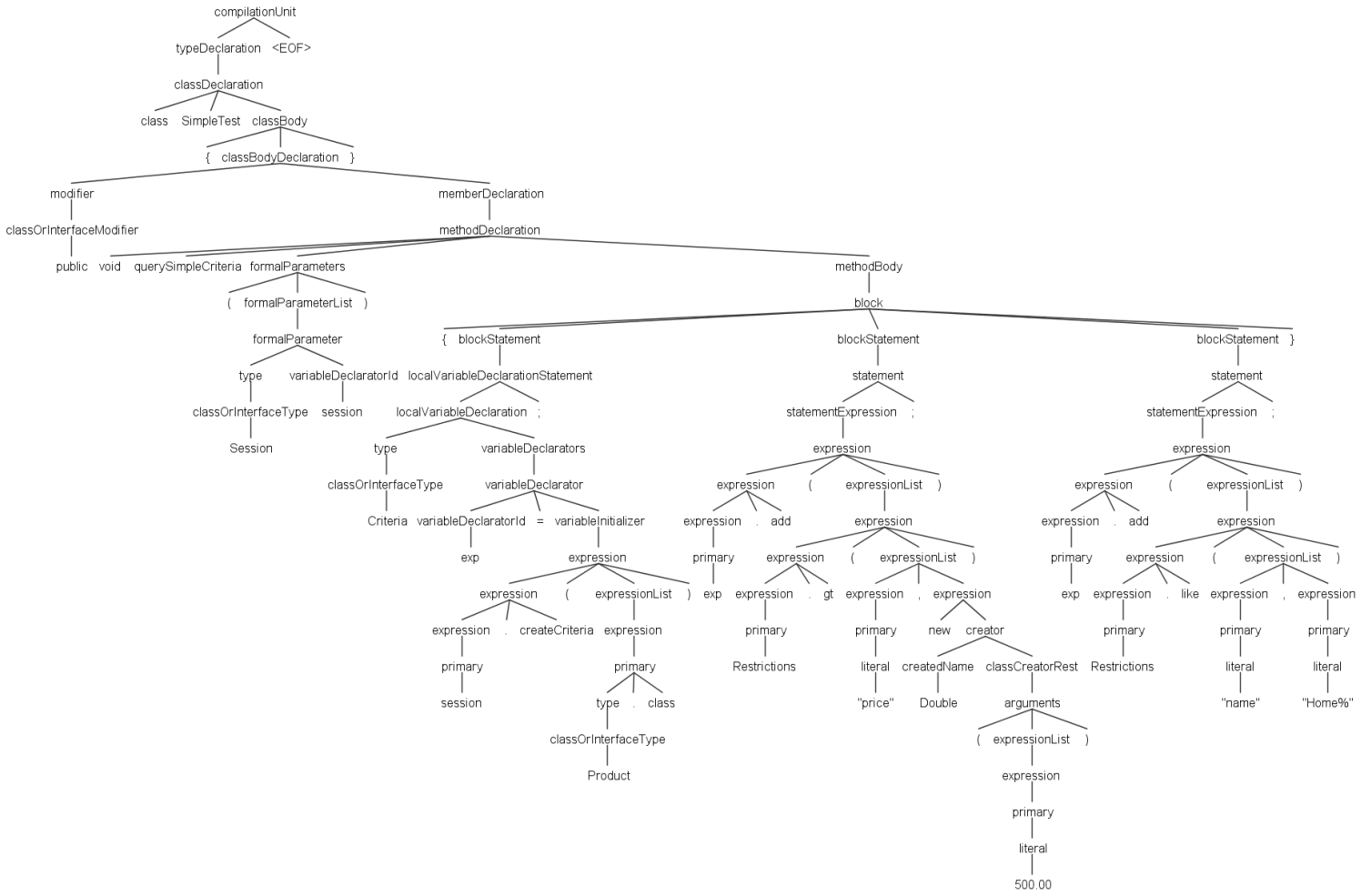

Fig. 8 The CFG from Criteria API expression with condition

After that the result showed output of expression with condition of HQL query syntax in Fig. 9.

```
Query query = session.createQuery
("from Product where price > 500.00 and name like
'Home%'");
```

Fig. 9 The example output after converted Criteria API to HQL expression

As the outputs showed, the HQL query expressions were from a designed algorithm in Fig. 10. The main mechanism is changing some Criteria API's values to HQL's value. The procedure of changing is observing the node's value by visit all nodes. Then, the compiler knew that which nodes are needed to change. After that, the node's name is collected as the id and retrieve the node's value for changing. Some node's values have to change because of the structure of HQL expression. Moreover, the difference entity or class name and difference methods are relevant. After complete changed values, the tree is traversed and show the result of HQL syntax. Then, the algorithm can reduce the time consumption when the applications are developed and maintained. Therefore, the algorithm id Fig. 10 can extract more methods which shown in Fig. 11 and Fig. 12 when the expression and primary node are called.

```
Start
    n.type is a type of node element
    n.id is a value of each node
    numChild is amount of child's node
    int count ← 0
    traverse all nodes
       dfs(node current)
          n.root ← top
             if n.type is classOrInterfaceType
                replace the word "Query" when n.id ← "Criteria"
             else if n.type is expression && numChild = 3
                Changed.id-Expression(node n)
              else if n.type is primary
                 Changed.id-Primary(node n)
             mark node as visited
     End traverse
End
```

Fig. 10 Criteria API to HQL Transformation Algorithm

```
Changed.id-Expression(node n)
check n.id of n[2]
  if n.id = "createCriteria"
    n.id ← "createQuery"
  if n.id = "rowCount"
    n.id ← "Count"
  if n.id = "gt"
    n.id ← ">"
  if n.id = "lt"
    n.id ← "<"
  if n.id = "eq"
    n.id ← "="
```

Fig. 11 Changen.id-Expression Method

```
Changed.id-Primary(node n)
if numChild = 3
  n.id[1] ← " "
  n.id[2] ← "from"
  n.id ← n.id[2] + n.id[1] + n.id[0]
  remove child
else if n.id = "Restrictions"
  count++
    if count = 1
      add the word "where" in the
condition
    else if count > 1
      add the word "and" in the condition
    else if n.id = "Order"
      n.id ← "order by"
```

Fig. 12 Changed.id-Primary Method

IV. PRELIMINARY RESULT AND CASE STUDY

After the algorithm designed, the performance testing is needed. We have to check the function of an algorithm which can convert the expressions correctly. The researchers tested by using other syntax from Hibernate text book and Hibernate official websites. [13-16] There are 2 forms of the expression; short form and full form. The designed algorithm can work with both forms. Short and full forms are collected the result in session variable but, the structure of full form has to declare a persistence object(s). The first time of retrieving the data is from the database directly. For the next if the same data are required, the Hibernate Framework can get directly from session variable. The execution time will be decreased. For the short form syntax is not necessary to do object declaration. The developers are allowed to retrieve data by using session variable. If the existing project use one of both styles which they cannot deal with, the algorithm can convert the expression syntax automatically. Moreover, the conversion of syntax is successful.

The performance of an algorithm is measure. In the study, the back to back testing is used for measurement which can show the equivalence of results. The back to back testing is one type of testing technique. The outputs are expected to compare and analyze [17]. The tests which come from an algorithm for testing are test cases. The test cases are usually designed before starting the test. The test cases will help testers to know which abilities of an algorithm have to test. After that, all cases have to execute with tool which develop from an algorithm. When the tests are success, the results are needed to compare. So, the retrieved data from Criteria API method is an actual result and the retrieved data from HQL method is expected result. Finally, the actual result and the expected result have to be the same. To compare the results, the testing is implemented the containsAll method for checking the equivalence query results of both query mechanisms. The testing result was pass which means that all patterns can

query the same data for Criteria API and HQL. Thus, the transformation algorithm can work correctly and efficiently. The syntax after converted is still retrieve the same data with the original commands.

V. Conclusion

In the study, a transformation algorithm is presented. The experiment tested the conversion of syntaxes between Criteria API to HQL expression. The difference structure of both expressions are concerned; full and short syntaxes. The principle of short and full syntaxes is shown difference declaration. However, the results are retrieved the same data under the same conditions. Not only to pick up the syntaxes for testing from the Hibernate textbook in the experiment, but the other examples from official websites and real world projects are also used. Thus, the transformation algorithm can be proved that it can work appropriately with real world situations. For getting more efficient of the algorithm, the queried results have to measure. The function of the transformation algorithm is confident to apply with any situations. The researchers decided to use back to back testing for evaluating the equivalence. The back to back testing check the equality of queried data between both mechanisms. It means that the retrieved data from Criteria API and HQL have to be the same. When the input of the algorithm is Criteria API expression, the result get the HQL expression. After the HQL expression result, the queried data should be compared to the queried data from input. The data from Criteria API should contain in the data from HQL and vice versa. So, The function of a transformation algorithm is acceptable. In the future work, the research will plan to do the transformation algorithm to convert the syntax back from HQL to Criteria API. So, the developers who familiar with difference structure will select the appropriate algorithm which relate with their works.

References

[1] Johnson, J., The Ghost of Christmas Future, in Chaos in the New Millennium. 2000, The Standish Group: West Yarmouth, MA.

[2] Linda, W. and K. Mark, Software project risks and their effect on outcomes. Commun. ACM, 2004. 47(4): p. 68-73. http://dx.doi.org/10.1145/975817.975819

[3] Tisinee, S. and D.-A. Chartchai, The Hibernate Framework Query Mechanisms Comparison. Lecture Notes on Software Engineering, 2014. 2: p. 268-274. http://dx.doi.org/10.7763/LNSE.2014.V2.135

[4] RedHat, J.C. Hibernate ORM Idiomatic persistence for Java and relational databases. [cited; Available from: http://hibernate.org/orm/.

[5] Patrick, P. and H. Nick, Hibernate Quickly. 2006, Greenwich: Manning Publications Co.

[6] Chuanlong, X., Y. Guangcan, and T. Meng, Efficient implement of ORM (Object/Relational Mapping) use in J2EE framework: Hibernate, in International Conference on Computational Intelligence and Software Engineering, 2009. 2009. p. 1-3.

[7] Hao, J., J. Liwei, and X. Zhuoming, Upgrading the relational database to the Semantic Web with Hibernate, in International Conference on Web Information Systems and Mining. 2009, IEEE Computer Society. p. 227-230.

[8] Jiseong, S., J. Dongwon, and B. Doo-Kwon, Practical Approach: Independently Using SPARQL-to-SQL Translation Algorithms on Storage, in Fourth International Conference on Networked Computing and Advanced Information Management. 2008, IEEE Computer Society. p. 598-603.

[9] Wang, J. and F. Rui, The Research of Hibernate Cache Technique and Application of EhCache Component, in IEEE 3rd International Conference on Communication Software and Network (ICCSN). 2011. p. 160-162.

[10] Qinglin, W., H. Yanzhong, and W. Yan, Research on Data Persistence Layer Based on Hibernate Framework, in 2nd International Workshop on Intelligent Systems and Application (ISA), 2010. 2010. p. 1-4

[11] Uday P. , K., S. Amitabha , and K. Bageshri Data Flow Analysis: Theory and Practice. 2009, CRC Press (Taylor and Francis Group).

[12] Jeff, L. and M. Dave, Beginning Hibernate. Vol. 2. 2010, New York, United States of America: SPRINGER.

[13] christian Bauer, G.K., Hibernate in action. 2005: Manning Publication Co.

[14] Patrick Peak, N.H., Hibernate Quickly. 2006, Greenwich: Manning Publications Co.

[15] java2s.com. Java Tutorial » Hibernate » Restrictions. 2009 [cited 2014 2/4]; Available from: http://www.java2s.com/Tutorial/Java/0350__Hibernate/RestrictionLIKE.htm.

[16] java2s.com. Java Tutorial » Hibernate » Criteria. 2009 [cited 2014 2/4]; Available from: http://www.java2s.com/Tutorial/Java/0350__Hibernate/ReturnUniqueResult.htm.

[17] Reactive Systems, I. (2012) Achieving ISO 26262 Compliance with Reactis. Volume, 28

Tisinee Surapunt The place of birth was born in Chiang Mai Thailand, April 30th 1988. The background education is Bachelor of Science, software engineering major (International Program) of College of Arts Media and Technology Chiang Mai University, Chiang Mai Thailand in 2009.

Presently, she is a graduate student in software engineering major of College of Arts Media and Technology Chiang Mai University, Chiang Mai Thailand. Moreover, she works as a TEACHING ASSISTANCE in software engineering program of bachelor degree, College of Arts Media and Technology, Chiang Mai University Thailand. The research interest is about testing field. She is also advertent performance testing.

Ms. Surapunt Awarded an Erasmus Mundus Scholarship to study at Corvinus University of Budapest, Budapest Hungary in October 2009.