INF FAKULTÄT FÜR INFORMATIK

Challenges in Refactoring Multi-Language Software Applications

59. CREST Open Workshop

Hagen Schink

Institute of Technical and Business Information Systems
Otto-von-Guericke-University of Magdeburg, Germany

2018-03-26

Introduction

Introduction I

- Polyglot Programming [Ford, 2008]
- Refactoring [Opdyke, 1992, Fowler, 1999]

Introduction II

Listing 1: SQL query in Java code.

```
1  String stmt = "SELECT name FROM departments "
2              + "WHERE name LIKE ?";
3
4  PreparedStatement query;
5  query = con.prepareStatement(stmt);
6
7  query.setString(1, "M%");
8  ResultSet result = query.executeQuery();
```

Introduction III

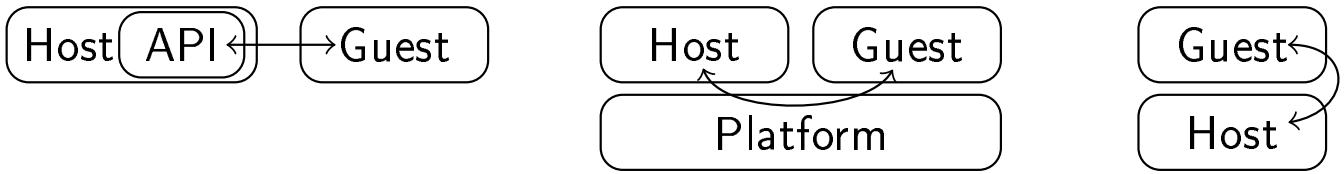
- Refactoring an MLSA can break the application [Schink et al., 2011]
- Automated *Multi-Language Refactorings* (MLR):
  - exist only for specific language interactions
  - are difficult to implement on a general basis [Chen and Johnson, 2008]
- Test coverage is required to check language interaction

Questions

1. *Why are MLRs hard to implement on a general basis?*
2. *How can we provide general refactoring support, though?*
3. *How could refactoring in Multi-Language Software Applications (MLSA) look like in future?*

Language Interaction in MLSAs

Types of Language Interaction



No relation *Platform* *Language*

Figure: Relations between host and guest language.

Types of Language Interaction

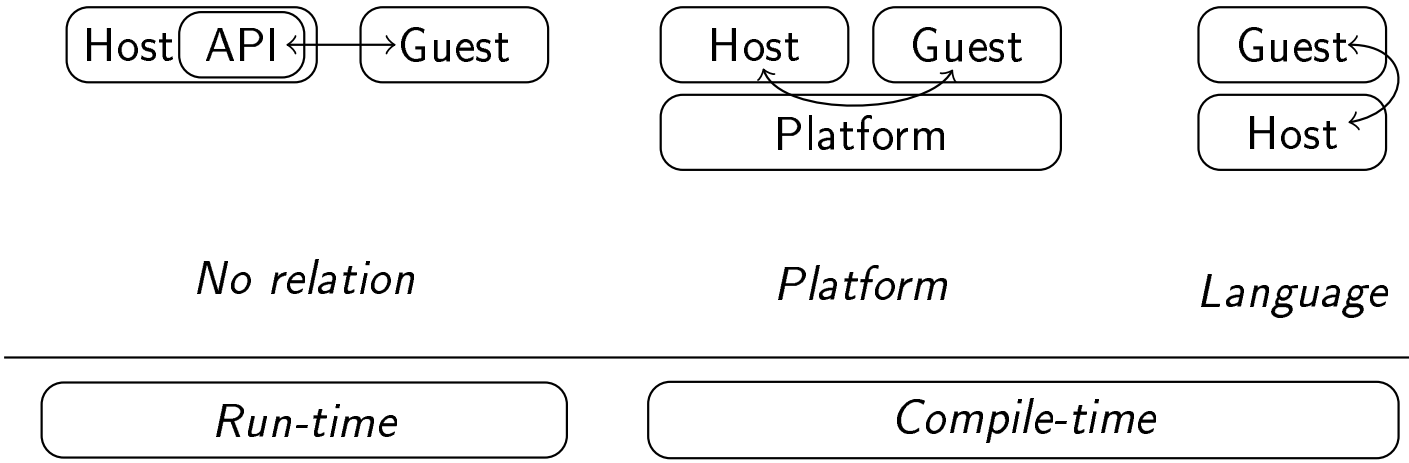


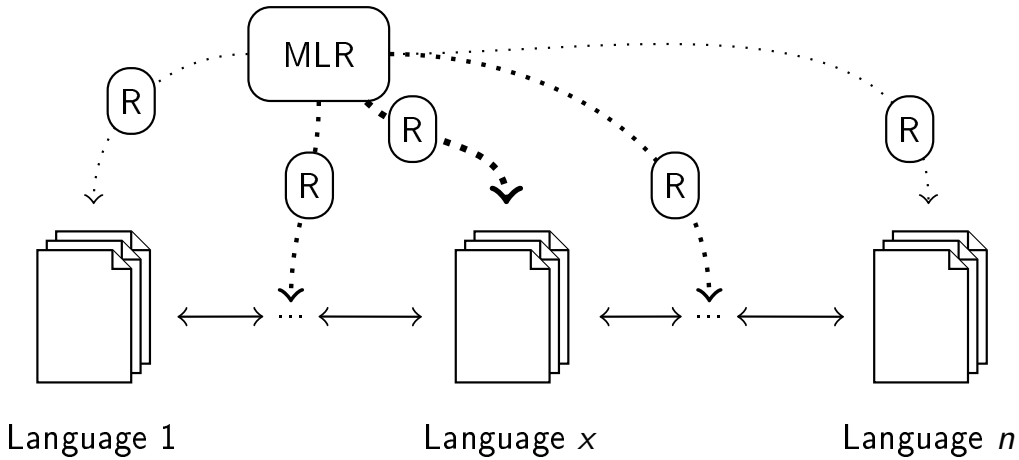
Figure: Relations between host and guest language.

Challenges for Automated Refactoring in MLSAs

"While finding a general solution for extending refactoring across multiple languages is hard, it is simple and possible to support automated refactorings for some common cases that programmers already encounter in their programs today."

[Chen and Johnson, 2008]

Challenges for Automated Refactoring in MLSAs



Language Combinations - Example

Listing 2: Definition of `sum` in Groovy in hello.groovy.

```
1  def sum(a, b) { a + b }
```

Listing 3: Invocation of `sum` with GroovyClassLoader.

```
1  GroovyClassLoader gcl = new GroovyClassLoader();
2  Class clazz = gcl.parseClass(new File("hello.groovy"));
3
4  Method sum = clazz.getMethod("sum",
5                               Integer.class,
6                               Integer.class);
7  Object instance = clazz.newInstance();
8
9  System.out.println(sum.invoke(instance, 1, 2));
```

Language Combinations - Example

Listing 2: Definition of sum in Groovy in hello.groovy.

```
1  def sum(a, b) { a + b }
```

Listing 4: Invocation of sum with ScriptEngine (JSR-223).

```
1  ScriptEngineManager factory = new ScriptEngineManager();
2  ScriptEngine engine = factory.getEngineByName("groovy");
3
4  engine.eval(new FileReader(new File("hello.groovy")));
5  Invocable inv = (Invocable) engine;
6  Object[] params = {1, 2};
7
8  System.out.println(inv.invokeFunction("sum", params));
```

Language Combinations - Other Examples

- Java - Clojure (similar to Groovy)
- Java - C (JNI, JNA)
- Java - relational Database (JDBC, JPA etc.)
- .Net - relational Database (DataReader, DataTable etc.)
- ...

Language Combinations - Summary

- Different approaches for interaction between two languages
- Approach may provide different options to establish interaction

Differing Language Concepts - Example

Listing 5: Invoice item in database.

```
1  CREATE TABLE invoice (
2     id              INT,
3     invoice_date    DATE,
4     amount          MONEY
5  );
```

Listing 6: Read data from table.

```
1  var query = "SELECT id, amount FROM invoice";
2  var cmd = new SqlCommand(query, cmd);
3  cmd.open();
4  var reader = cmd.ExecuteReader();
5
6  var id = reader.GetInt32(0);
7  var amount = reader.GetDecimal(1);
```

Differing Language Concepts - Summary

- Languages may not share all concepts
- Languages may differ in the implementation of a given concept
- Semantical changes may be necessary to adapt to a refactoring

Summary of Challenges

...or why a general approach to refactoring MLSAs is hard.

- Number of languages and frameworks
- Differing language concepts
- Scaling a solution to 1 and 2

Structure Graphs

Challenges and Measures

Challenges to approach

- A refactoring may not exist for all interacting languages
- Automatic refactoring may not available for all languages

Challenges and Measures

Challenges to approach

- A refactoring may not exist for all interacting languages
- Automatic refactoring may not available for all languages

Means to address the challenges

- Developers need to detect language interaction
- Developers need to know the mechanics of language interaction

Challenges and Measures

Challenges to approach

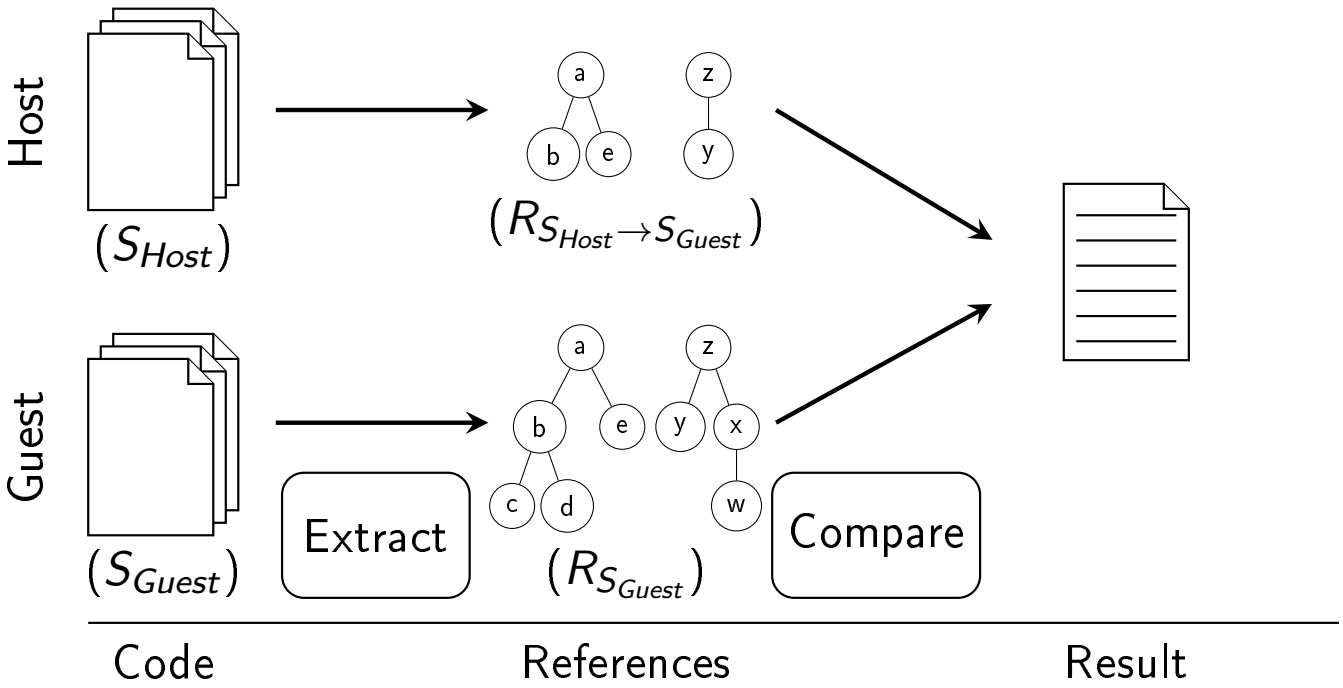
- A refactoring may not exist for all interacting languages
- Automatic refactoring may not available for all languages

Means to address the challenges

- Developers need to detect language interaction
- Developers need to know the mechanics of language interaction

A tool needs to enable developers to complete refactorings manually.

Concept



S... Source code

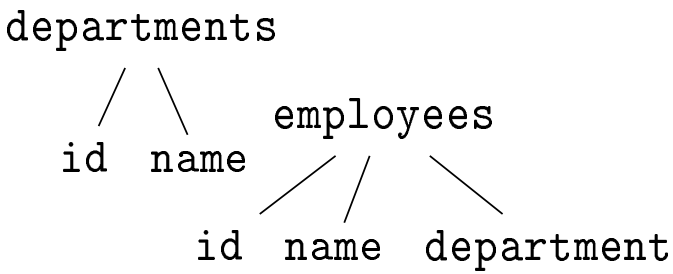
R... Elements involved in language interaction

Structure Graphs - An Example

ID	NAME
1	IT
2	HR
3	Sales

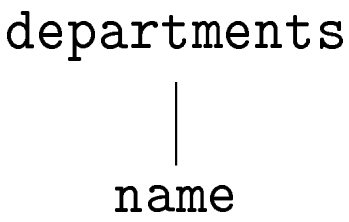
ID	NAME	DEPARTMENT
1	John Doe	1

Table: Tables departments and employees.



Listing 7: Query for department names.

```
1 String stmt = "SELECT name "
2               + "FROM   departments "
3               + "WHERE  name LIKE ?";
4 PreparedStatement query =
5   con.prepareStatement(stmt);
6 // snip
```



Implementation - General

- Extraction of structure graphs of host and guest language
- Comparison of structure graphs
- Augmentation of results with language-specific information

Implementation - Extraction of Structure Graphs

- Extract elements involved in language interaction that

Implementation - Extraction of Structure Graphs

- Extract elements involved in language interaction that
  - may be hidden in string

Listing 8: Invocation of `sum` with GroovyClassLoader.

```
1 inv.invokeFunction("sum", params)
```

Implementation - Extraction of Structure Graphs

- Extract elements involved in language interaction that
  - may be hidden in string
  - may be encapsulated in framework-specific functions

Listing 9: Definition of column *id*.

```
1 var id = new DataColumn("id", typeof(int)); // DataTable
```

Implementation - Extraction of Structure Graphs

- Extract elements involved in language interaction that
  - may be hidden in string
  - may be encapsulated in framework-specific functions
  - may use different type system

Listing 10: Read of column *amount*.

```
1 var amount = reader.GetDecimal(1); // DataReader
```

Implementation - Extraction of Structure Graphs

- Extract elements involved in language interaction that
  - may be hidden in string
  - may be encapsulated in framework-specific functions
  - may use different type system
- Agreement on the structure graph

Implementation - Preparation of Results

- The comparison returns missing and non-matching elements

Implementation - Preparation of Results

- The comparison returns missing and non-matching elements
- Additional information can be retrieved from the results

Implementation - Preparation of Results

- The comparison returns missing and non-matching elements
- Additional information can be retrieved from the results

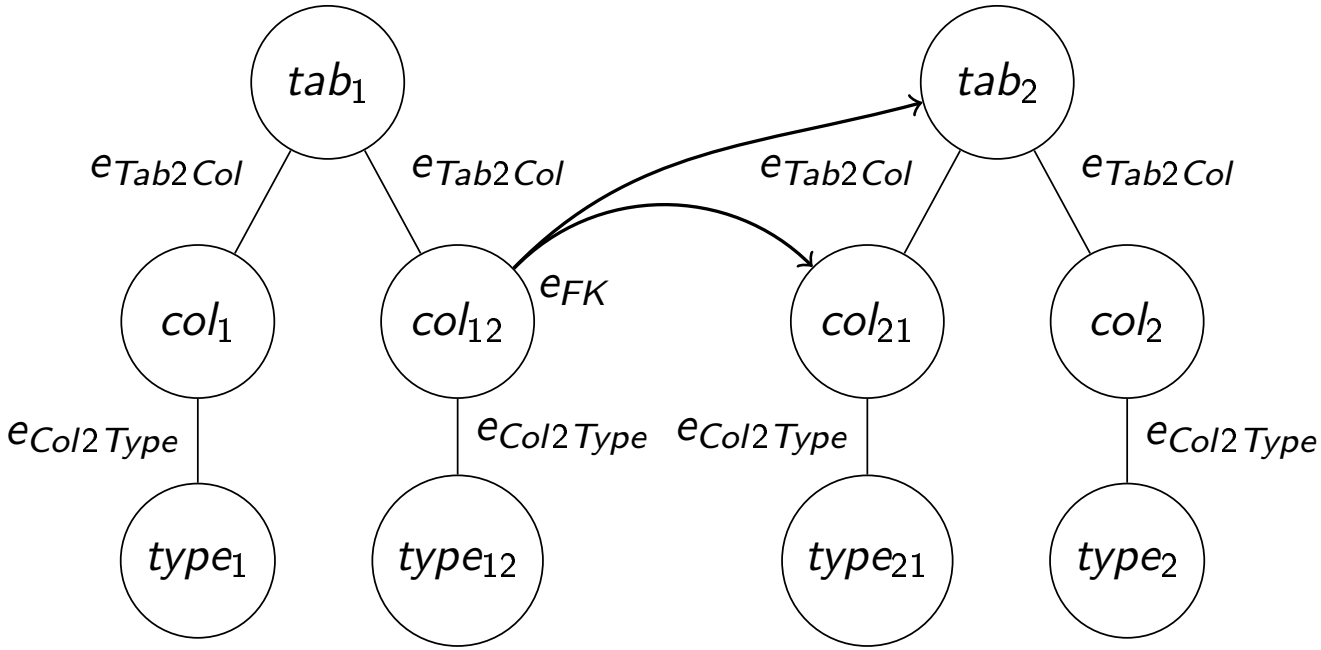


Figure: General database schema with a foreign-key relationship.

Implementation - Scalability

Adding new languages scales because...

- ...structure graph comparison is language independent
- ...a structure graph for a given guest language can be reused for new host language
- ...preparation of results can be reused for new host languages

The structure graph approach scales over the host languages.

Related Approaches

- Common Meta Model [Strein et al., 2006]
  - Interacting languages share common syntax elements
  - A refactoring exists for all interacting languages
  - Applicable only for certain types of language interaction
- Linking Model [Mayer and Schroeder, 2014]
  - Basis is a model of all syntax elements
  - Binding resolvers establish links between the model's elements
  - Binding resolvers encapsulate the mechanics of language interaction

Conclusion and Discussion

Conclusion

- MLRs are hard to implement due to the structure of MLSAs
- Without MLRs it is error prone to refactor MLSAs
- Structure graphs support detection of broken interactions[1]

[1] Evaluation confirms applicability and performance

Discussion

Some *provocative* questions...

- Establishing language interaction at run-time impedes refactoring: Should we deem it an anti-pattern inside MLSAs?

Discussion

Some *provocative* questions...

- Establishing language interaction at run-time impedes refactoring: Should we deem it an anti-pattern inside MLSAs?
- Should standardization drive language interaction (like JPA or JSR-223) to avoid clutter of APIs?

Discussion

Some *provocative* questions...

- Establishing language interaction at run-time impedes refactoring: Should we deem it an anti-pattern inside MLSAs?
- Should standardization drive language interaction (like JPA or JSR-223) to avoid clutter of APIs?
- To avoid MLR entirely, should we favor APIs (and API versioning) also within MLSAs?

Discussion

Some *provocative* questions...

- Establishing language interaction at run-time impedes refactoring: Should we deem it an anti-pattern inside MLSAs?
- Should standardization drive language interaction (like JPA or JSR-223) to avoid clutter of APIs?
- To avoid MLR entirely, should we favor APIs (and API versioning) also within MLSAs?
- Should we refactor a common platform to avoid MLR entirely (see for instance Common Metal Model)?

References


Chen, N. and Johnson, R. E. (2008).
Toward Refactoring in a Polyglot World Extending Automated Refactoring Support Across Java and XML.
In *Proceedings of the 2nd ACM Workshop on Refactoring Tools (WRT 2008), Nashville, TN, USA, October 19, 2008*. ACM.


Ford, N. (2008).
The Productive Programmer.
O'Reilly Media, Inc., Sebastopol, CA, USA.


Fowler, M. (1999).
Refactoring: Improving the Design of existing Code.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.


Mayer, P. and Schroeder, A. (2014).
Automated Multi-Language Artifact Binding and Rename Refactoring between Java and DSLs Used by Java Frameworks.
In *Proceedings of the 28th European Conference Object-Oriented Programming (ECOOP 2014), Uppsala, Sweden, July 28 - August 1, 2014*. Springer.


Opdyke, W. F. (1992).
Refactoring Object-Oriented Frameworks.
PhD thesis, University of Illinois, Champaign, IL, USA.


Schink, H. (2013).
sql-schema-comparer: Support of Multi-Language Refactoring with Relational Databases.
In *Proceedings of the 13th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2013), Eindhoven, Netherlands, September 22-23, 2013*. IEEE Computer Society.


Schink, H., Broneske, D., Schröter, R., and Fenske, W. (2016a).
A Tree-Based Approach to Support Refactoring in Multi-Language Software Applications.
In *Proceedings of the 2nd International Conference on Advances and Trends in Software Engineering, Lisbon, Portugal, February 21-25, 2016*. IARIA.


Schink, H., Kuhlemann, M., Saake, G., and Lämmel, R. (2011).
Hurdles in Multi-language Refactoring of Hibernate Applications.
In *Proceedings of the 6th International Conference on Software and Data Technologies (ICSOFT 2011), Seville, Spain, July 18-21, 2011*. SciTePress.


Schink, H., Siegmund, J., Schröter, R., Thüm, T., and Saake, G. (2016b).
A Study on Tool Support for Refactoring in Database Applications.
Softwaretechnik-Trends, 36(2).


Strein, D., Kratz, H., and Löwe, W. (2006).
Cross-Language Program Analysis and Refactoring.
In *Proceedings of the 6th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2006), Philadelphia, Pennsylvania, USA, September 27-29, 2006*. IEEE Computer Society.