View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by PORTO Publications Open Repository TOrino

Classification of Language Interactions

Federico Tomassetti*, Marco Torchiano* and Antonio Vetro'*
*Dept. of Control and Computer Engineering
Politecnico di Torino
Torino, Italy
Email: [federico.tomassetti|marco.torchiano|antonio.vetro]@polito.it

***Abstract*—Context: the presence of several languages interacting each other within the same project is an almost universal feature in software development. Earlier work shows that this interaction might be source of problems.**

Goal: we aim at identifying and characterizing the cross-language interactions at semantic level.

Method: we took the commits of an open source project and analyzed the cross-language pairs of files occurring in the same commit to identify possible semantic interactions. We both defined a taxonomy and applied it.

Result: we identify 6 categories of semantic interactions. The most common category is the one based on shared ids, the next is when an artifact provides a description of another artifact.

Conclusions: the deeper knowledge of cross-language interactions represents the basis for implementing a tool supporting the management of this kind of interactions and the detection of related problems at compile time.

I. Introduction

pol·y·glot |'päli,glät|
adjective
knowing or using several languages.[1]

Polyglotism is largely recognized as an almost ubiquitous characteristic of modern software development projects: they use several different languages [**?**]. For instance, most trivial web applications are typically written in a general-purpose language, e.g. *Java*, include some *SQL* queries, are visually presented by means of *HTML*, formatted using *CSS* files, and with client-side processing implemented using *Javascript*.

An important side effect of polyglotism within a single project is the interaction between languages. From previous studies [**?**] [**?**] we learned that the majority of commits in open source projects are cross-language, i.e. they involve files written in different languages.

Identifying the interactions between artifacts in different languages (cross-language) is important because most development environments, with the notable exception of some platform-specific IDEs – e.g. some Android IDEs – do not provide any support for managing them. A good knowledge of cross-language interactions is a key factor in building a specific support into IDEs with the goal of supporting development, maintenance, and comprehension activities on polyglot applications.

Two main approaches to language identification are possible. Logical interactions [**?**] occur when two artifacts are modified in the same version control system commit. Semantic interactions occur when within an artifact we can find some elements that link it to another artifact.

In our previous study [**?**] we investigated logical interactions. In this paper we attempt to both verify how frequently logical interactions do actually correspond to semantic interactions and classify the different types of semantic interactions.

The goal of this paper is twofold: (i) on one hand we investigate the relationship between logical interactions and semantic interactions, and (ii) we attempt an initial classification of the semantic language interactions.

II. Related work

The authors of this paper previously worked on a preliminary evaluation of the effects of language interactions [**?**]. Results indicate that most of the commits involve files written using different languages. The prevalence of *cross-language* commits depends on the kind of activity being performed (e.g., implementation of a new feature involves the 30% of *cross-language* commits, while writing tests involves a mere 5% thereof). In particular the work showed the presence of a correlation between interactions involving certain pairs of languages (e.g. C-Java) and an increased defectiveness.

Later we proposed a prototypal solution [**?**] for language integration adopting a Language Workbench (the Jetbrains Metaprogramming System[2]). Our solution addressed only one particular kind of interaction (Shared ID) while in with this work we plan to present a comprehensive classification of possible interactions between artifacts written in different languages.

Gall et al. [**?**] proposed a definition of logical coupling between files based on the observation of a software repository. They defined as logically coupled two files which were changed together in at least one commit. This definition allows identifying possible relations which can not be easily found out with a more rigorous syntactic analysis. Another advantage of this approach is the possibility to apply it to all possible kind of artifacts. In later work [**?**], [**?**], Ratzinger et al. showed that logic couplings defined on the basis of a repository's history could be used to find artifacts which need to be refactored (reducing the coupling). This analysis complemented syntactic coupling.

[1]Definition from the Oxford American Dictionary

[2]http://www.jetbrains.com/mps/

Mayer and Schroeder [?] name the problems of references across artifacts written in different languages as "semantic cross-language links". Being these links out of scope of the individual programming language, they are ignored by most language-specific tools and are often checked only at runtime. They propose to explicitly express constraints for these links and present three possible approaches to do that: at source code level, using language-specific meta-models, and using language-spanning meta-models. Of these approaches they chose the second. We instead advocated the third in our previous work [?], because it permits to reuse a common API and, in the case of the MPS language workbench, it is already available without the need of developing it.

Pfeiffer realized a system called TexMo [?] which permits to express references between artifacts written in different languages (corresponding the category of interaction that we named as *Shared ID*), but not to express other kind of constraints. It is realized as an Eclipse plugin and it is intended to be used instead of the original editors provided inside Eclipse. Our prototypal approach [?] does not require recreating the editors but instead permit to simply enrich the industrial-strength editors already available in MPS. It does not resort on a limited universal metamodel, but instead uses the MPS representation of the language, allowing to consider every aspect of the language.

Pfeiffer et al. [?] used TexMo in a controlled experiments with 22 subjects to demonstrate the effects of tool support for cross-language references. Results show a significative improvement in the ability to correctly locate the source of errors due to broken cross-language references.

III. Method

We devised a research method to achieve the dual objective of identifying interaction categories and classifying the occurrences of interactions in a real project.

The procedure we followed is made up of the six steps below:

1) *screening*: we identified logical interactions by selecting the *cross-language* commits using the approach based on file extensions, defined in [?].
   We adopted this approach to focus on a limited number of candidate pairs since examining all possible connections between every pair of files in a large project requires both a deep knowledge of the project itself and a huge effort.
2) *commit selection*: we selected the bug-fixing commits from the project version control system.
   This choice is motivated by the fact that they typically represent a focused modifications involving a limited set of files. Considering that we are interested in binary relations between files, a large number of files could lead to an exponential number of possible pairs of files to be analyzed.
3) *manual verification*: we verified the language of the files to confirm the presence of *cross-language* logical interactions in a predefined temporal range (the same used in the previous study).
   Since the previous step is based on the file extensions alone, some false positive are possible. Where different extensions actually correspond to the same language or viceversa the same extension (or lack of) correspond to different languages.
   This manual inspection led us to classify as bash scripts files that had not an extension. In a few cases that left us with a commit where only bash files were modified, therefore the commit was clearly not a cross-language commit and so it was excluded from further examination.
4) *semantic interaction manual confirmation*: we manually inspected the files modified simultaneously in the same commit, using mainly the contextual diffs of the involved files and the relative log message to identify *cross-language* interactions and to assign them to a class.
   In this way we progressively constructed a taxonomy of semantic interactions.
5) *revision of the classification*: we discussed the classification built in the previous step, merged similar categories, and defined more meaningful labels.
   The goal of this step is to come up with a clear and precise definition of the *cross-language* interaction categories and provide representative examples. The results at this stage are presented in section IV.
6) *semantic interaction classification*: we re-processed all the commits and performed a definitive classification of the cross-language interactions according to the final taxonomy.
   When several instances of the same relation were found between the same pair of files (e.g. many *Shared ID*) just one occurrence was reported. Though the same relation could possibly be counted more than once, if it appears in distinct commits.
   The result at this final stage is a set of *cross-language* semantic interactions identified over a set of commits. We then conducted an analysis – presented in section V – of such data aimed to: i) verify the precision of the logic interaction approach in terms of semantic interactions, ii) define a frequency profile of *cross-language* interaction categories.

IV. Categories

As a result of step 5) in the procedure delineated in section III, we built a taxonomy of the cross-language semantic interactions. The interactions between different languages can occur in several different forms. While it is extremely common to use more than one language in a single project, it could even be the case that different languages are used in the same file. For instance consider the presence of an utterance of SQL embedded in a valid expression of an host language (typically a General Purpose Language like Java or PHP) or the preprocessing languages as the C preprocessor language or M4. In our work we assumed that it is possible to identify

TABLE I
CATEGORIES FOR THE IMPLEMENTATION OF LANGUAGE INTERACTIONS AMONG DIFFERENT ARTIFACTS

Category	Definition
Shared ID	The same ID is used among the artifacts involved in the interaction.
Shared data	A piece of data have to hold exactly the same value among the different artifacts involved.
Data loading	A data from one of the file involved is loaded by the code in another file involved.
Generation	From one of the file involved the other files involved are completely or partially generated. Also the modification of part of a file is accepted.
Description	One of the file involved contained a description of the content of another file (a part or the whole file).
Execution	One file execute the code contained in another file.

a main or host language in which a certain artifact (e.g., a file) is expressed. We decided to focus only on interactions between distinct artifacts taking into consideration the main language of each file.

We emphasize that the identified categories are not mutually exclusive. For example a Java class could load an XML file (*Data loading* relation) and then perform some processing on specific part of it, using identifiers for the navigation. In that case normally the same identifier is present both in the Java and in the XML file (*Shared ID* relation). In our example therefore there will be both *Data loading* and *Shared ID* relations on the same pair of files.

In Table I we report the definition of the categories we identified. In the rest of the section we present an example for each category. Examples are derived from the interactions classified according to the procedure in section III.

A. Shared ID - Example

A configuration file written in XML (Listing 1) contains the qualified name of a Java class (Listing 2). The class is named `S3FileSystem` and it is contained in package `org.apache.hadoop.fs.s3`; the fully qualified name is therefore `org.apache.hadoop.fs.s3.S3FileSystem`.

```
<property>
  <name>fs.s3.impl</name>
  <value>org.apache.hadoop.fs.s3.S3FileSystem</value>
  <description>The FileSystem for s3: uris.</description>
</property>
```

Listing 1. Snippet from file File src/java/core-default.xml at commit 1058343

```
public class S3FileSystem extends FileSystem {
```

Listing 2. Snippet from file File src/java/core-default.xml at commit 1058343

B. Shared data - Example

Two different configuration files of Ivy have to specify the same version of a particular library. The library is the Google Protobuffer and the value of the version is "2.4.0a". The files involved are an XML file (Listing 3) and a properties file (Listing 4).

```
<dependency>
  <groupId>com.google.protobuf</groupId>
  <artifactId>protobuf-java</artifactId>
  <version>2.4.0a</version>
</dependency>
```

Listing 3. Snippet from file File ivy/hadoop-common-template.xml at commit 1134857

```
protobuf.version=2.4.0a
```

Listing 4. Snippet from file File ivy/libraries.properties at commit 1134857

C. Data loading - Example

Configuration data is loaded by Java code from an XML file, to implement a unit test on the `Configuration` class.

In Listing 5 you can read the code of the XML file, while in Listing 6 is reported the line responsible for loading the XML file.

```
<!-- This file is a fake version of a "default" file like
  core-default or mapred-default, used for some of the unit
      tests.
  -->
<configuration>
  <property>
    <name>tests.fake-default.new-key</name>
    <value>tests.fake-default.value</value>
    <description>a default value for the "new" key of a
        deprecated pair.</description>
  </property>
</configuration>
```

Listing 5. Snippet from file File src/test/test-fake-default at commit 1126719

```
static {
  Configuration.addDefaultResource("test-fake-default.xml");
}
```

Listing 6. Snippet from file src/test/core/org/apache/hadoop/conf/Test/ConfigurationDeprecation.java at commit 1126719

D. Generation - Example

A script used for setup may generate different files. For example the bash in Listing 7 file generates the actual mapred-site.xml from a template.

In this case the repository contains the template file but not the generated file, which would be present in a project using Hadoop.

```
...
template_generator ${HADOOP_PREFIX}/share/hadoop/common/
     templates/conf/mapred-site.xml ${HADOOP_CONF_DIR}/mapred-
     site.xml
...
```

Listing 7. Snippet from file src/main/packages/hadoop-setup-conf.sh at commit 1190035

E. Description - Example

The documentation of the Access Control List functionalities reported in Listing 8 describes a functionality expressed in class AccessControlList (path src/java/org/apache/hadoop/security/authorize/AccessControlList.java).

```
<tr>
  <td>mapreduce.cluster.acls.enabled</td>
  <td>Boolean, specifying whether checks for queue ACLs and job
      ACLs are to be done for authorizing users for doing queue
       operations and job operations.</td>
  <td>If <em>true</em>, queue ACLs are checked while submitting
      and administering jobs and job ACLs [..]. </td></tr>
```

Listing 8. Snippet from file src/docs/src/documentation/content/xdocs/cluster_setup.xml at commit 998001

F. Execution - Example

A POM file executes the code of Java class (see Listing 9).

```
<doclet>org.apache.hadoop.classification.tools.
    IncludePublicAnnotationsStandardDoclet</doclet>
```

Listing 9. Snippet from file pom.xml at commit 1195817

V. Classification

The real project we selected for the analysis is Hadoop[3] (See step 1 in Section III). We considered 39 bug-fixing commits from the Hadoop project (step 2), that were classified in [**?**] as cross-language (because they contain logical interactions). After a first inspection we discarded 3 commits because they were not cross-language (step 3).

Out of the remaining 36 commits we found semantic cross-language relations which we could classify in 27 cases (75%). More in details, in 11 commits we found one interaction, in 10 cases two relations, in 3 commits we found 3 relations, in two cases we found 4 occurrences, and in one case even 8 interactions.

We can conclude that using logical interaction as a proxy to identify semantic interactions has an estimated average precision of 75% with a 95% confidence interval ranging from 57% to 87%, estimated using a proportions test.

Figure 1 reports the frequency of the interaction categories. Of course here we report only the relations which we were able to identify. We cannot exclude the presence of other relations that we were unable to detect. Thus the number of *cross-language* interactions we identified could be interpreted as a lower-bound of possible existing relations. Some relations could be expressed implicitly, for example a file could load the content of another file using a library method of which we do not know the semantics.

The most frequent category of relation is by far *Shared ID* (27 instances). In 12 cases we found a *Shared data* relation, in 10 *Description*, in 4 *Data loading*, in 2 *Generation*, and in 1 *Execution*. In this case of a *Generation* relation the repository normally contains the file which represents the source of the generation – typically a template file – but not the generated file, which would be present in a running configuration of the system.

The most frequently involved files were xml (42 cases), followed by java (30), properties (16) and sh (11). In 3 cases each also ac, am and spec files were involved. In only one case we found file with avpr and c extension.

VI. Conclusions and future work

We conducted and investigation of the cross-language semantic interactions in an open-source project. A very simple approach based on logic links – co-presence in the same commit – is able to indicate the presence of confirmed semantic interactions with a limited though acceptable precision (75%).

Based on the actual instances we defined a taxonomy of semantic interactions, which provide us with a deeper understand of cross-language relations. The relations we individuated are: *Data loading*, *Description*, *Execution*, *Generation*, *Shared data*, *Shared ID*.

[3]http://hadoop.apache.org

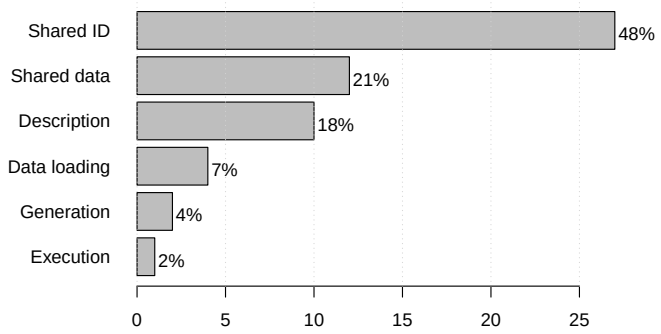

Fig. 1. Frequency of semantic cross-language interaction categories

We also computed the frequency of occurrence of the individual categories. Apparently about 50% of the interactions take places by means of shared ids.

An ongoing work is being devoted to the implementation of tool support for cross-language interactions. The knowledge on the interaction categories is the main starting point for designing tool support for cross-language interactions. In addition the information about the frequency allow defining priorities among the different interaction categories when building the supporting tool.

References

[1] D. Wampler, T. Clark, N. Ford, and B. Goetz, "Multiparadigm programming in industry: A discussion with neal ford and brian goetz," *IEEE Software*, vol. 27, no. 5, pp. 61–64, 2010. [Online]. Available: http://dx.doi.org/10.1109/MS.2010.121

[2] A. Vetro', F. Tomassetti, M. Torchiano, and M. Morisio, "Language interaction and quality issues: an exploratory study," in *Proc. of the ACM-IEEE int. symposium on Empirical soft. eng. and measurement*, ser. ESEM '12. New York, NY, USA: ACM, 2012, pp. 319–322. [Online]. Available: http://doi.acm.org/10.1145/2372251.2372309

[3] F. Tomassetti, A. Vetro', M. Torchiano, M. Voelter, and B. Kolb, "A model-based approach to language integration," in *Modeling in Software Engineering (MISE), 2013 ICSE Workshop on*, 2013.

[4] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *Software Maintenance, 1998. Proc., Int. Conf. on*, 1998, pp. 190–198. [Online]. Available: http://dx.doi.org/10.1109/ICSM.1998.738508

[5] J. Ratzinger, M. Fischer, and H. Gall, "Improving evolvability through refactoring," in *Proc. of the 2005 int. workshop on Mining software repositories*, ser. MSR '05. New York, NY, USA: ACM, 2005, pp. 1–5. [Online]. Available: http://doi.acm.org/10.1145/1082983.1083155

[6] J. Ratzinger, T. Sigmund, P. Vorburger, and H. Gall, "Mining software evolution to predict refactoring," in *Empirical Soft. Eng. and Measurement, 2007. ESEM 2007. First Int. Symp. on*, 2007, pp. 354–363.

[7] P. Mayer and A. Schroeder, "Cross-language code analysis and refactoring," in *Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th Int. Working Conf. on*, sept. 2012, pp. 94 –103.

[8] R.-H. Pfeiffer and A. Wasowski, "Texmo: a multi-language development environment," in *Proc. of the 8th European Conf. on Modelling Foundations and Applications*, ser. ECMFA'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 178–193.

[9] ——, "Cross-language support mechanisms significantly aid software development," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, vol. 7590. Springer Berlin Heidelberg, 2012, pp. 168–184.