Spotting Automatically Cross-Language Relations

Federico Tomassetti
Politecnico di Torino
10129 Turin, Italy
Email: federico.tomassetti@polito.it

Giuseppe Rizzo
Università di Torino, 10124 Turin, Italy
EURECOM, 06410 Biot, France
Email: giuseppe.rizzo@di.unito.it

Marco Torchiano
Politecnico di Torino
10129 Turin, Italy
Email: marco.torchiano@polito.it

***Abstract*—Nowadays most of the software projects are coded using several formal languages, either spread on different artifacts or even embedded in the same one. These formal languages are linked each other using cross-language relations, mainly framework specific and established at runtime. In this work we present a language agnostic approach to automatically detect cross-language relations to ease re-factoring, validation and to allow navigation support to the developer. We map a project in a set of Abstract Syntax Trees (ASTs); pair-wise we compute the intersection of the nodes and we pre-select potential candidates that can hold cross-relations. We then factorize the ASTs according to the nodes which surround the candidate and pair-wise we compute the semantic similarity of the factorized trees. We narrow down a set of statistically significant features and we map them into a predictive model. We apply such a procedure to an AngularJS application and we show that this approach spots cross-language relations at fine grained level with 93.2% of recall and a F-measure of 92.2%.**

I. Introduction

Most of the applications realized today are composed by artifacts written in different languages. The Web offers an excellent case study, since the majority of the applications use several languages for both server and client side. On the server side the typical scenario includes at least a general purpose language (GPL), SQL and some formats to store configuration (XML, JSON, etc.). On the client side HTML, CSS and Javascript are typically adopted. The different artifacts cooperate to execute some tasks, as part of the application, therefore they have to communicate and be linked together: a certain CSS rule affects a given tag, a XML file describes which Java classes have to be instantiated, the execution of a Ruby script is affected by the configuration reported in a YAML[1] file. Different languages can be mixed even in the same artifact, for instance consider CSS or Javascript in HTML pages or a Java function call receiving a string which happens to be SQL code. Framework or single projects can also adopt Domain Specific Languages (DSL) to realize specific facets of the complex system. In Listing 1 we report an example of a Java statement specifying a query to a database using the Hibernate Query Language (HQL), a DSL resembling SQL. The query retrieves all the rows from the table `Employee`. The resulting rows are then converted into corresponding Java objects by Hibernate[2], a well-known Object-Relational Mapper. By convention tables and corresponding Java classes have the same name (`Employee` in this case), therefore there is a cross-language relation between the Java class and the table reference in the query: if one of them changes, developers should consider to update the others. The specific rules which determine how the artifacts are composed depend on the language and the framework used: each can use its own logic to operate, creating run-time relations between artifacts. Considering Listing 1, the fact this particular call receives a string supposed to be a valid HQL code is defined by the implementation of Hibernate. Instead conventions are usually not formalized explicitly, but are nevertheless relevant to favor communication between developers and ease comprehension.

```
List<Employee> employees = session.createQuery("FROM Employee").
     list();
```

Listing 1. A snippet of HQL code in a Java statement.

The rules for cross-language relations, being determined by framework implementations or by conventions, have to be studied and to be always considered during the development. A violation of a hard rule leads to errors which are difficult to track because implicit relations spread across different files in different languages have to be considered all together. Hence, even just renaming a Java class can lead to a runtime error because the name of the class was not updated in few XML and property files referring to it. Violating conventions could instead lead to a code which is harder to maintain, because developers rely on these conventions for comprehension. While modern IDEs offer support to identify inconsistencies between two files written in the same language, the developer is typically left on his own when it comes to cross-language relations. Without refactoring support the developer has to replicate manually the update in all related artifacts. Without navigation support cross-language references are not immediately apparent, the developer has to know and remember the cross-language rules determining relations and to manually navigate to other files for retrieving related information. Without validation support, a broken link is not immediately apparent. An experiment performed by Pfeiffer *et al.* [1] shows that tool support for cross-language relation can greatly improve the developer performance. However all the existing approaches are framework specific: they require to manually specify the rules which govern the relations expected by a certain library. Each new language, each new DSL, each new framework require to adapt these tools. Considering that formalizing cross-language relations precisely can be quite difficult per se [2] this leads to a considerable effort to implement and maintain cross-language relations support.

Taking inspiration by this, in this paper we motivate the following research question: is it possible to automatically spot cross-language relations in a variety of projects written with different languages? Exploiting the semantics of the language and relying on a predictive model, our approach is

[1] http://yaml.org
[2] http://hibernate.org

able to spot cross-language relations with 92.2% of F1[3]. The experiments have been conducted on an in-house benchmark which is, together with the source code of the framework and the experiment settings, publicly available at http://github.com/CrossLanguageProject. The reminder of the paper is organized as follows. In Section II we further explain our motivation and prior work. We present the benchmark used for the experiments in Section III. Our approach is presented in Section IV and in Section V we present the experiment results. Finally in Section VI we discuss the results and we foresee possible outlook.

II. Related work

Research attempts on cross-language relations are quite recent. Generally they can be summarized as: *i)* to offer a classification of cross-language relations, *ii)* to characterize empirically the effects of cross-language relations, *iii)* to provide prototyping tool support.

Classification: in [3] the authors presented a classification of different forms of cross-language relations, identifying six different types: shared ID, shared data, data loading, generation, description, and execution. Of all those types the most commonly used is shared ID. In the context of this work we focus exclusively on this kind of cross-language relations. Meyer and Schroeder [2] classify exclusively cross-language links implemented in Java framework in respect to XML artifacts. The relations they consider correspond to the category "shared ID", according to the classification proposed in [3]. They built metamodels of the languages involved (Java and XML) for this particular purpose and specified the rules controlling the cross-language relations of three Java framework, deriving from them common patterns. Among the main results, they report that specifying manually rules for cross-languages relations is difficult.

Empirical results: in [4] the authors investigated how many of the commits of the Hadoop[4] project involved more than one language and the effect of being involved in cross-language commits on defectivity. Results show that some relations are particularly negative. However results of this paper are based on a coarse proxy for the identification of cross-language relations; considering projects hosted on a repository, they relied on the logs for looking at the files which have been committed at the same time. But this approach leaves the burden of spotting manually the cross-language relations. In fact, a method to automatically identify cross-language relations at a finer level can permit more precise empirical investigation on their effects on a large scale, where manual identification is not feasible. Pfeiffer *et al.* [1] used TexMo in a controlled experiments with 22 subjects to demonstrate the effects of tool support for cross-language references. According to their results, developers having access to tool support for cross-language references were significantly faster and more frequently correct in locating sources of errors. Developers without this type of support instead have difficulty to reconduct the errors which they encountered at run-time to their ultimate source, a broken cross-language relation.

[3]By F1 we mean the F-Measure with $\beta = 1$. It corresponds to the harmonic mean of precision and recall.

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

Description	Values
no. of different files	12
no. of formal languages involved	4
no. of lines among all files	2927
no. of manually identified cross-language relations	142

TABLE I. Statistics of the benchmark proposed in this paper. Any artifact is considered, excluded pictures. The number of language involved considers also the natural language text. The number of cross-language relations is computed considering HTML and JS artifacts (excluded lib artifacts).

Specific tool support: possible solution to the problem consist in *i)* developing specific IDE support, *ii)* substituting existing languages with families of integrated languages, *iii)* implementing proper language integration inside language workbenches. The first approach was adopted by Pfeiffer *et al.* [5], [6]: they realized different prototypes integration tool support for cross-language relations into mainstream IDEs named as TexMo and Tengi. An example of *family of languages* comes from Groenewegen *et al.* [7]: upon observing that the amalgam of languages used in a single web application project are typically poorly integrated they proposed the adoption of an unique language to model all the different concerns of web applications: WebDSL. Finally regarding *language integration* in the context of Language Workbenches is described by Tolvanen *et al.* [8]. In their paper they describe their experience in integrating Domain Specific Modeling (DSM) languages. They considered only DSM realized in the context of the MetaEdit+ system, without integration with GPLs. GPLs integration is instead possible in another Language Workbench: Jetbrains MPS. An example in this direction is described in this paper [9]. Integration in mainstream IDEs has the great advantage to leverage environments which are already familiar to most of the developers, but they require the implementation of specific support for each single framework considered. On the contrary, the other solutions require a migration but offer deeper integration, attainable with a limited effort.

III. Benchmark

To the best of our knowledge, no other research attempts have been spent to spot at fine grained level the cross-language relations. It results in a lack of gold standards for benchmarking the performance of proposed approaches. To fill this gap, in this paper we propose an in-house benchmark as compendium of our approach. As described previously, the Web offers a vast number of projects written using different formal languages each. In addition, the most used formal languages for Web applications (HTML, JS, CSS) share intrinsically numerous cross-language relations, which are usually hidden, making the task extremely challenging. We have then selected a web project based on the AngularJS[5] framework named angular-puzzle[6]. Table I summarizes the statistics of the proposed benchmark.

Two human experts have been involved in the creation of the benchmark; the aim was to manually detect the cross-language relations between artifacts of two different extensions (JS and HTML), to which we excluded the AngularJS library artifacts. We also excluded the CSS files, since the identification of cross-language relations is easier to be spotted due to

[5]http://angularjs.org

[6]http://github.com/pdanis/angular-puzzle

the tag selection. The relations have been reported pair-wise; the dataset lists the *src* and *dst* files, *row* and *column* where the relation have been spotted from both artifacts and the *surface form* (shared ID) of the relation per each cross-language relation. The overall agreement score reached so far was good. After the first annotation step, we have dedicated a cleansing step for fixing the errata spots. The benchmark is released as public license at http://github.com/CrossLanguageProject/goldstandards.

IV. Method

Most of the cross-language relations are implemented using a shared identifier. For example a Javascript statement could refer a particular tag in a HTML document by its ID. When one of the two ends of the relation is changed, the other one has also to be updated, if the relation wants to be preserved. However not all the instances of the same terms are related, because not all of them identify the same entity. For instance, if we consider pairs of instances of the same term appearing in different files, written in different languages, chances are high that the two identified entities will be different and unrelated. Our method aims to automatically and independently identify the cross-language relations from the languages considered and the used framework.

To perform the classification we factorize the AST using the candidate spots as pivots and we exploit the *context* of each pair. The semantic similarity measure between each pair of local factorized ASTs is computed. In details the proposed method consists of the following steps:

- for each artifact an AST is derived. An AST may host sub-ASTs of different languages, corresponding to snippets of embedded languages;
- for each node the set of nodes corresponding to its context is collected;
- pairs of nodes corresponding to the same term and enclosed in artifacts of different formal languages are spotted as potential candidates. For each pair a semantic similarity is computed. This process is meant to narrow down a set of features which are then used by the classification stage;
- all the candidate spots become observations of the classification model; of these pairs of nodes the ones related will be positive examples, the candidate which are not related but hold the same term will be the negative examples. The learning stage is used to train the algorithm of the predictive model.

A. ASTs construction

The first stage of the process is to map each source file to its corresponding AST representation. This allows to distinguish between keywords and relevant values present in the source code (i.e. identifiers and literals) and to organize the data in a logical structure on which is possible to reason about relations between nodes. The host language of a file is determined by its extension (the assumption made is grounded on the fact that a Java file usually contains a host Java AST). Inside the host AST, foreign ASTs can be added. They represent snippets of other languages embedded in the original file.

Consider the example shown in Figure 1: a snippet of HTML is reported. The attribute `onclick` of the `div` tag contains Javascript code, in particular a call to the function `showStats`. From this piece of code is derived an AST having as root a HTML node. The Javascript snippet is appropriately parsed and a corresponding Javascript AST is obtained. The Javascript AST is then embedded in the host HTML AST as a child of the HTML attribute which contains the Javascript code.

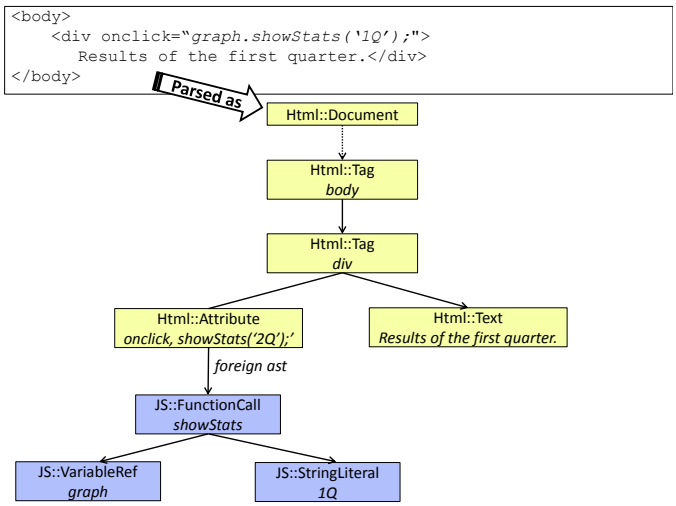

Fig. 1. A Javascript AST embedded in a HTML AST.

B. Context

The role of the context for spotting cross-language relations has been previously introduced by Mayer *et al.* [2]. Inspired by this preliminary consideration, we have started over an exhaustive investigation on the context in the field of spotting automatically cross-language relations. We then consider important the context to discriminate between instances of terms that just happen to have the same surface form from instances which are concretely related.

Consider the example shown in Figure 2: the term `title` appears two times in *index.html* and five times in *app.js*. The first appearance in *index.html* is related to the first two appearances in *app.js* while the remaining instances in the two files are also reciprocally related. They can be intuitively distinguished on the basis that the first group of instances is hosted in the context of `types`, while the second is hosted in the context of `puzzles`.

To translate this consideration into a formalized approach we devise an algorithm to traverse siblings node in ASTs which is language agnostic. More details about the context extraction algorithms can be found in the source code of the project. The outcome of the context identification steps is a set of AST nodes which constitute the context of a given input node.

C. Features derivation

The candidate pairs of nodes, which share the same surface forms, are compared by means of their contexts; the resulting comparison is therefore executed on the two contexts. From

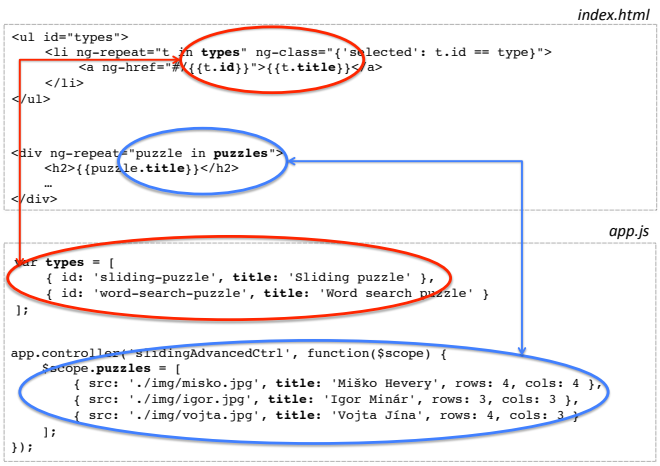

Fig. 2. Example of cross language relations organized in hierarchies.

each pair of contexts two different set of surface forms are extracted. These sets are compared using state-of-the-art algorithms for instance matching such as Levenshtein algorithm (working at word grained level), Jaccard, Jaro, and Tversky [10]. These algorithms provide a coefficient of similarity, when the coefficient is superior to a given threshold the nodes are considered related, otherwise they are not. Therefore they provide a raw idea whether a candidate spot actually defines a valid cross-language relation. For this matter, we have considered them as baselines in our proposal.

Further features have been derived from the context similarity, such as: the number of words appearing in both contexts; the sum of tf-idf value of the words appearing in both contexts; the sum of the itf-idf value of the words appearing in both context (the itf is defined as $log(1/tf)$); the percentage of words of each of the two contexts which appear also in the other; the number of words of each of the two contexts which do not appear also in the other; the percentage of words of each of the two contexts which do not appear also in the other. Table II proposes a recap of the above features.

D. Classification

Starting from the derived features, we map the task of spotting cross-language relations to a predictive task. The features are meant to: *i)* define the model and *ii)* train the classifier. Inspired by Mayer *et al.* [2] who proposed a set of manually created rules, we built the classifier in order to create a list of rules that can automatically predict whether the observation is actually a cross-language relation. To achieve such a scope we used the Random Tree (RT) algorithm. We then compared this algorithm with K-nearest neighbors (K-NN) and Naive Bayes (NB) algorithms.

V. Experiment and Results

The benchmark proposed above reports all the cross-language relations that two human experts have manually spotted in the context of the angular-puzzle project. Anyway, such a task covers only the true positive spots of the domain, leaving the burden to spot also the true negative ones. The

Name	Description
shared length	number of words appearing in both contexts
shared tf-idf	sum of tf-idf value of the words appearing in both contexts. See http://en.wikipedia.org/wiki/Tf-idf
shared itf-idf	sum of the itf-idf value of the words appearing in both context. The itf is defined as $log(1/tf)$
perc. shared length [min,max]	the percentage of words of each of the two contexts which appear also in the other
diff [min,max]	the number of words of each of the two contexts which do not appear also in the other
perc. diff [min,max]	the percentage of words of each of the two contexts which do not appear also in the other
Levenshtein distance*	similarity distance between two set of chars (either two simple words or sequence of them). In this paper we use the second version of it
Jaccard distance	similarity distance between two sequences of words
Jaro distance	similarity distance between two sequences of words
Tversky distance	similarity distance between two sequences of words

TABLE II. List of the features exploited in the proposed work.

population of interest includes all the pairs of nodes which i) are contained in files with different extensions, and ii) share a common word; the negative ones are those pairs which satisfy these conditions and are not semantically related yet. These latter pairs were automatically individuated. The union of the two sets forms the benchmark over which we have run our experiments. Hence, the results of our experiments are reported in terms of correctly spots of cross-language relation in case of actually a positive case (the pair holds a cross-relation) or correctly spot that the pair does not hold a valid cross-language relation.

Using a correlation matrix, we verify the correlation each feature has with the class to predict (originally the class spans from positive when the observation details a cross-language relation, negative otherwise). The results are reported in Table III.

	class
shared_length	0.0616
tfidf_shared	-0.0084
itfidf_shared	0.0861
perc_shared_length_min	0.0864
perc_shared_length_max	0.1703
diff_min	0.0560
diff_max	-0.0446
perc_diff_min	-0.0173
perc_diff_max	-0.0549
context	0.0374
jaccard	0.0969
jaro	0.0340
tversky	0.1061

TABLE III. An excerpt of the correlation matrix, where we highlighted the correlation scores of the features to the class. From it we observe that tfidf, perc_diff_min, diff_max, and perc_diff_max are inversely correlated with the class to predict. For such a reason, we leave these four features out of the predictive model.

We performed a 10-fold cross validation and used WEKA-3.7.9[7] for running the classifiers. Table IV reports the figures achieved so far by our approach according to three different classifiers. The RT performs better such kind of task, proving the intuition that Mayer *et al.* had in their paper, that a rule based approach can help for deciding whether a shared ID actually holds a cross-language relation. Our approach is extremely competitive, nearly solving the problem.

	P	R	F1
Naive Bayes (NB)	90.3	86.3	88.1
K-nearest neighbor (K-NN)	91.3	93.0	91.9
Random Tree (RT)	**91.6**	**93.2**	**92.2**

TABLE IV. PRECISION (P), RECALL (R) AND F-MEASURE (F1) RESULTS OF OUR APPROACH USING THREE DIFFERENT CLASSIFIERS.

Finally, in Table V we compare the figures achieved by our approach with the ones obtained by simple approaches which leverage on instance matching algorithms. It is evident how the cross-language spotting task cannot be solved by just context similarity evaluation.

	P	R	F1
$\text{Levenstein}_{d=1}$	6.0	100	11.8
$\text{Tversky}_{d=0.8}$	12.9	43.8	19.9
$\text{Jaccard}_{d=0.8}$	13.9	35.0	19.9
Random Tree (RT)	**91.6**	**93.2**	**92.2**

TABLE V. PRECISION (P), RECALL (R) AND F-MEASURE (F1) RESULTS OF THE BASELINES AND OUR PROPOSED APPROACH.

VI. DISCUSSION AND OUTLOOK

We believe in the benefit of adopting the most suitable language to implement each facet of the system. Using the best language for the task leads to polyglot systems which include artifacts written in many languages; it also requires proper coordination to smooth the development, improving productivity and the quality of the developed systems. To obtain good language integration we need first of all to recognize cross-language interactions: the approach detailed in this paper aims to do that and it seems promising, considering the results obtained on the proposed case-study.

In this work we proposed a language agnostic approach to spot automatically cross-language relations. To measure the goodness of the approach we have created an in-house benchmark, which, to the best of our knowledge, is a first attempt in this direction. We believe that the case study we chose is particularly daunting for the presence of different mechanisms of interactions and the mix of languages appearing even in the same artifacts. Since we are looking forward to enlarge the current benchmark, we have released it publicly and we hope that the community will contribute to extend and improve it: refinements of our work as well as alternative solutions will be easily compared in this way. Using a predictive algorithm, we are able to nearly solve the problem of spotting cross-language relations with a F1 of 92.2%. The features used have been extracted from the context of the factorized ASTs compared pairwise. Alternative solutions are neither generic nor automatic, therefore they require a major framework-specific effort. Our approach instead permits to freely adopt new frameworks and libraries, maintaining cross-language tool support.

Our prototypal implementations is based on a library wrapping a set of existing parsers[8]. Support for HTML, Javascript, Ruby, Java, XML, and properties files is already implemented. This implementation is offered to the community with the hope it will serve as a component for the realization of alternative approaches. Our implementation is designed in such a way that the support for other languages may be easily plugged in. To support additional languages all the language-specific work that is required is: *i)* thin wrapper around an existing parser, *ii)* the specification of the conditions under which another language can be added in the one considered.

As next step we want to work on generalization of our approach: we want to test the devised algorithms on different projects, realized using different frameworks. Then we plan to conduct empirical validation of *i)* the performance of our approach and *ii)* the benefits of such approach of polyglot software development. We also plan to further investigate cross-language relations which span to more than 2 different artifacts by time. That means considering the lattice of the combinations among the nodes extracted from the different artifacts.

REFERENCES

[1] R.-H. Pfeiffer and A. Wasowski, "Cross-Language Support Mechanisms Significantly Aid Software Development," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7590, pp. 168–184.

[2] P. Mayer and A. Schroeder, "Patterns of cross-language linking in java frameworks," in *21^{st} International Conference on Program Comprehension (ICPC'13)*, 2013.

[3] F. Tomassetti, M. Torchiano, and A. Vetro', "Classification of Language Interactions," in *7^{th} International Symposium on Empirical Software Engineering and Measurement (ESEM'13)*, 2013.

[4] A. Vetro', F. Tomassetti, M. Torchiano, and M. Morisio, "Language Interaction and Quality Issues: An Exploratory Study," in *6^{th} International Symposium on Empirical Software Engineering and Measurement (ESEM'12)*, 2012.

[5] R.-H. Pfeiffer and A. Wasowski, "Texmo: A multi-language development environment," in *8^{th} European Conference on Modelling Foundations and Applications (ECMFA'2012)*, 2012.

[6] ——, "Tengi Interfaces for Tracing between Heterogeneous Components," in *Generative and Transformational Techniques in Software Engineering IV*, ser. Lecture Notes in Computer Science, R. Lammel, J. Saraiva, and J. Visser, Eds. Springer Berlin Heidelberg, 2013, vol. 7680, pp. 431–447.

[7] D. Groenewegen and E. Visser, "Declarative Access Control for WebDSL: Combining Language Integration and Separation of Concerns," in *8^{th} International Conference on Web Engineering (ICWE '08)*, 2008.

[8] J.-P. Tolvanen and S. Kelly, "Integrating models with domain-specific modeling languages," in *10^{th} Workshop on Domain-Specific Modeling (DSM'10*, 2010.

[9] F. Tomassetti, A. Vetro', M. Torchiano, M. Voelter, and B. Kolb, "A Model-Based Approach to Language Integration," in *ICSE Workshop on Modeling in Software Engineering (MISE'13)*, 2013.

[10] G. Navarro, "A Guided Tour to Approximate String Matching," *ACM Computing Survey*, vol. 33, no. 1, pp. 31–88, 2001.

[7] http://www.cs.waikato.ac.nz/ml/weka

[8] See https://github.com/ftomassetti/codemodels and its plugins.