CrossLanguageSpotter: A Library for Detecting Relations in Polyglot Frameworks

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

Giuseppe Rizzo
Università di Torino, Turin, Italy
EURECOM, Sophia Antipolis, France
giuseppe.rizzo@di.unito.it

Raphaël Troncy
EURECOM, Sophia Antipolis, France
raphael.troncy@eurecom.fr

ABSTRACT

Nowadays, most of the web frameworks are developed using different programming languages, both for server and client side programmes. The typical scenario includes a general purpose language (e.g. Ruby, Python, Java) used together with different specialized languages: HTML, CSS, Javascript and SQL. All the artifacts are connected via different types of relations, most of which depend on the adopted framework. These cross-language relations are normally not captured by tools which require the developer to learn and to remember those associations in order to understand and maintain the application. This paper describes a library for detecting cross-language relations in polyglot frameworks. The library has been developed to be modular and to be easily integrated in existing IDEs. The library is publicly available at `http://github.com/CrossLanguageProject/crosslanguagespotter`.

Categories and Subject Descriptors

D.2.6 [**Programming Environments**]: [Programmer workbench]

Keywords

Polyglot development, Cross-language relations, Tool support

1. INTRODUCTION

Modern programming practices include the integration of pieces of code coming from very diverse frameworks where the developer aims to better exploit the strengths of different programming languages and existing resources. IDEs[1] offer support to identify inconsistencies between two artifacts written in the same language and to enable code refactoring of embedded languages such as Javascript[2]. However, the developer is often left on his own when it comes to cross-language relations. Without refactoring support, the developer has to replicate manually the updates in all related artifacts. Without navigation support, cross-language references are not immediately visible, and the developer has to know and remember the cross-language rules determining the relations and to manually navigate to other files for retrieving related information. Without validation support, a broken link remains invisible.

A preliminary analysis on a sample of over 2,500 GitHub projects reports that 96% of projects are polyglot by nature, i.e. they use more than one language (considering programming languages, scripting languages, data languages, etc.) and nearly 2 projects out of 3 make use of more than one programming language. If this scenario is common in the majority of projects, this problem is even accentuated within web frameworks. Figure 1 shows two artifacts, written in different languages, from the angular-puzzle project[3] linked by cross-relations. The term *title* appears twice 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. The role of the context is what makes hard to automatically distinguish between related pairs of elements and pairs which are not related, even if they are represented by the same identifier.

We believe that the ability to detect automatically cross-language relations is worth to be explored for two main reasons: *i)* cross-language relations are difficult to formalize since they are related to the frameworks implementation and they cannot generally be translated into clear rules; *ii)* frameworks keep evolving and, as they evolve, the rules for cross-language relations change. Since our approach leverages on the semantics of the involved artifacts, it naturally adapts without manual work.

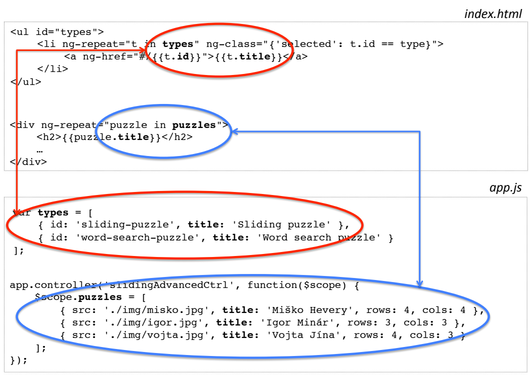
In this paper, we describe a library that automatically detects cross-language relations leveraging on the semantics of the container artifacts. This approach has been tested to spot relations in an existing web framework and we observed that it is able to detect cross-language relations with 92.2% of F1 [5].

[1]Integrated Development Environment such as `http://www.eclipse.org`

[2]See for example `http://www.sublimetext.com` together with `https://github.com/s-a/sublime-text-refactor`

[3]https://github.com/pdanis/angular-puzzle

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04..



```
<ul id="types">
    <li ng-repeat="t in types" ng-class="{'selected': t.id == type}">
        <a ng-href="#/{{t.id}}">{{t.title}}</a>
    </li>
</ul>


<div ng-repeat="puzzle in puzzles">
    <h2>{{puzzle.title}}</h2>
    …
</div>
```

```
var types = [
    { id: 'sliding-puzzle', title: 'Sliding puzzle' },
    { id: 'word-search-puzzle', title: 'Word search puzzle' }
];

app.controller('slidingAdvancedCtrl', function($scope) {
    $scope.puzzles = [
        { src: './img/misko.jpg', title: 'Miško Hevery', rows: 4, cols: 4 },
        { src: './img/igor.jpg', title: 'Igor Minár', rows: 3, cols: 3 },
        { src: './img/vojta.jpg', title: 'Vojta Jína', rows: 4, cols: 3 }
    ];
});
```

Figure 1: Example of cross language relations organized in hierarchies.

2. RELATED WORK

Three approaches are usually adopted for detecting cross-language relations: *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.* [2, 3]: they implemented 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.* [1]: upon observing that in a single web application project, the amalgam of languages used are typically poorly integrated, they proposed the adoption of a unique language to model the different concerns of web applications: WebDSL. Finally, *language integration* in the context of Language Workbenches is described by Tolvanen *et al.* [4]. In this work, 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 [6]. 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.

3. IMPLEMENTATION

CrossLanguageSpotter is a library designed to operate with different languages. Currently, Java, Ruby, Javascript, XML, HTML, and Properties are supported. First, the library builds the models from the source code files (Abstract Syntax Trees - AST). It looks then for relation pairs, which are held by shared ids (such as name of variables or name of functions that we term *entities*) among all projects artifacts written in different languages. The library extracts pariwise the contexts from the two artifacts that surround a shared id, and computes the similarity indexes. Those values are used as inputs of a classifier, that, finally, gives a Boolean decision regarding whether a relation actually exists or not.

3.1 Components and general design

The main language chosen for the implementation of *CrossLanguageSpotter* is JRuby. The library integrates *Codemodels*[4] for building ASTs. Codemodels offers support for different languages, and it is based on the same meta-metamodeling facility (based on an RGen, a Ruby clone of EMF). New plugins for Codemodels can be conveniently developed by reusing existing wrappers written either in Java or Ruby (since Codemodels is written in JRuby). *CrossLanguageSpotter* can navigate all the ASTs provided looking for pairs of nodes containing the same identifiers. When it founds such pairs, it calculates a set of 10 features (see Table 2 in [5], which are then used by a Random Tree classifier to classify real cross-language relations from pair of nodes having the same identifier. The classifier is based on Weka[5], a well-known Java library for machine learning. The classifier needs to be trained providing examples of classification.

3.2 Core modules

The core components of the proposed library are an *i)* AST builder and *ii)* a Relation spotter.

3.2.1 AST builder

The library defines a common representation of ASTs named *Codemodels*. Any supported language has its own adapter. An adapter provides a language metamodel and a parser. The language metamodel complies with the Codemodels specifications. From the source code, the parser produces an AST compliant to the language specific metamodel (and indirectly to Codemodels). Later in the paper, we refer to this parser as *ls-parsers*.

A ls-parser is normally implemented as a wrapper around an existing parser. The wrapper converts the AST obtained from the original parser (*o-parser*) to a proper instance of the Codemodels-compliant metamodel. The library being based on JRuby, parsers written either in Ruby or Java can be conveniently wrapped. As a reference, the efforts to wrap a Java parser in Codemodels are quantifiable to write 528 lines of JRuby code, while wrapping a Javascript parser requires 873 lines[6].

The first step in processing a file is to invoke the ls-parser obtaining an AST. Later, this AST is inspected to verify the presence of other language utterances. Typical examples include the presence of Javascript code embedded in an HTML document (e.g. in attribute values or DOM nodes) or the inclusion of SQL statements in Java string literals. When utterances of other languages are found, they are parsed, usually with variants of the corresponding ls-parser. Those variants are able to parse language snippets instead of complete files (e.g. a Java expression instead of a full Java source file). The obtained ASTs are called *embedded ASTs*. They are inserted in the original ASTs (the *host AST*) into the position corresponding to the elements of the original language hosting the utterances of the embedded language. The resulting AST will contain, possibly, ASTs which are instances of different language specific metamodels but all the nodes are valid Codemodels nodes.

3.2.2 Relations spotter

Each node of the AST has a set of properties. Typically, they are literal values and identifier names. Nodes sharing a common property value may be related. To determine if

[4]https://github.com/ftomassetti/Codemodels

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

[6]See https://github.com/ftomassetti/Codemodels-java and https://github.com/ftomassetti/Codemodels-js

it is the case ,the library compares the contexts of those two nodes. We name the context of a node, its set of surrounding nodes. It includes both the ancestors (i.e., all the nodes going from the node itself to the root of the AST) and all the descendants of the node (i.e., all of its children, and their children recursively). Then, all node properties that are part of the context are collected.

From the set of values, we are only interested in those which appear in both languages: if we are considering a pair of nodes from Javascript and HTML, we discard the values which do not appear in both Javascript and HTML nodes contained in the project. This step reduces the amount of noise. Once the potential pair candidates are selected, distance indexes are computed for measuring the similarity contexts [5]. All these indexes are sent as inputs to a classifier which predicts whether or not the relation actually exists. The prediction is performed using the Random Tree (RT) classifier.

4. USAGE

In Listing 1, we show how the library can be used from JRuby[7]. The example shows the complete process, from training (lines 1-2), to the calculation of cross-language relations (line 5).

Listing 1: An example of a complete usage

```
1  oracle_loader = OracleLoader.new
2  classifier = oracle_loader.build_weka_classifier('
       oracle-src-dir','oracle.GS')
3  spotter = CrossLanguageSpotter::Spotter.new()
4  project = Project.new('projects-dir')
5  relations = spotter.classify_relations(project,
       classifier)
```

4.1 Inputs

The library works with two different kinds of inputs: an oracle path and a project path. Both oracle and project paths point to two sets of source files that can be utterances of all supported languages. The oracle has a file (GS file) which contains a list of pairs of nodes. During the training process, the library loads all the source files of the oracle and finds all node pairs which contain the same identifiers. If those pairs are listed in the GS file, they are classified as positive example, while otherwise, they will be classified as negative examples. For all examples, either positive or negative, 10 different features are computed [5]. The resulting data is used to train the Random Tree classifier. The GS file specifies pairs of nodes by indicating the containing file of each node, the starting line, column, and the portion of the source code representing the node.

4.2 Output

The output returned by the `classify_relations` method is a list of records. Each record is composed of two identifier objects plus a Boolean value which indicates whether the relation exists or not. Each identifier object is a vector *o=(shared_id, source_file, start_line, end_line, start_column, end_column)*, where *shared_id* is the surface form (generally named entity), $source_{f}ile$ is the file where the identifier belongs to, and the others entries are the offsets of the shared id in the $source_{f}ile$.

[7]For an integration in Java code, please refer to `https://github.com/jruby/jruby/wiki/JRubyAndJavaCodeExamples`

4.3 Performance

For our in-house testing, the order of magnitude is tens of seconds (circa 600 lines) to build ASTs from the source files and to train the classifier. The classification process is, instead, faster (the order of tens of milliseconds for a small project).

4.4 Integration

The library output is based on source file positions. This choice makes possible to easily integrate the library itself with existing IDEs or other development tools. If the tool which integrates the library would use a different format of ASTs, the integrator would need to find out which AST node corresponds to the given position.

5. OUTLOOK

In this paper, we present a prototype library for detecting automatically cross-language relations. The road-map includes the automatic detection of programming languages deploying Markov Chains techniques, the support for more programming languages (by extending *Codemodels*), and the improvement of the feature selections to boost the performance of the classifier. Finally, we plan to integrate this library in an existing IDE such as Sublime Text[8] or Light Table[9]. For a full IDE integration, we plan to implement a caching system of the AST of unchanged files to reduce bootstrap latencies. In the long run, we could use incremental parsing techniques to improve the performance.

6. REFERENCES

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

[2] R.-H. Pfeiffer and A. Wasowski. Texmo: A multi-language development environment. In *8th European Conference on Modelling Foundations and Applications (ECMFA'12)*, 2012.

[3] R.-H. Pfeiffer and A. Wasowski. Tengi Interfaces for Tracing between Heterogeneous Components. In R. Lammel, J. Saraiva, and J. Visser, editors, *Generative and Transformational Techniques in Software Engineering IV*, volume 7680 of *Lecture Notes in Computer Science*, pages 431–447. Springer, 2013.

[4] J.-P. Tolvanen and S. Kelly. Integrating models with domain-specific modeling languages. In *10th Workshop on Domain-Specific Modeling (DSM'10)*, 2010.

[5] F. Tomassetti, G. Rizzo, and M. Torchiano. Spotting Automatically Cross-Language. In *IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE'14)*, 2014.

[6] 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.

[8]`http://www.sublimetext.com/`

[9]`http://www.lighttable.com`