StORMeD: Stack Overflow Ready Made Data

Luca Ponzanelli, Andrea Mocci, Michele Lanza
REVEAL @ Faculty of Informatics, University of Lugano, Switzerland

***Abstract*—Stack Overflow is the de facto Question and Answer (Q&A) website for developers, and it has been used in many approaches by software engineering researchers to mine useful data. However, the contents of a Stack Overflow discussion are inherently heterogeneous, mixing natural language, source code, stack traces and configuration files in XML or JSON format.**

We constructed a full island grammar capable of modeling the set of 700,000 Stack Overflow discussions talking about Java, building a heterogeneous abstract syntax tree (H-AST) of each post (question, answer or comment) in a discussion. The resulting dataset models every Stack Overflow discussion, providing a full H-AST for each type of structured fragment (*i.e.,* JSON, XML, Java, Stack traces), and complementing this information with a set of basic meta-information like term frequency to enable natural language analyses. Our dataset allows the end-user to perform combined analyses of the Stack Overflow by visiting the H-AST of a discussion.

I. Introduction

Among the different sources of information available online, Q&A websites have gained a prominent role in the daily working life of developers [1]. The archetypal example is Stack Overflow, where more than 6,000 questions are asked every day [2], providing "archives with millions of entries that contribute to the body of knowledge in software development" [3]. Indeed, according to the last data dump of September 2014[1], Stack Overflow relies on a community counting 3.5M users, who produced about 8M posts, out of which 700K[2] questions and 1.35M related answers concern the Java language.

This considerable amount of data attracted the interests of researchers in the last years. It is possible to find discussions on Stack Overflow collecting various studies performed on the various data dumps provided by Stack Exchange[3]. In many cases, the phase of mining Stack Overflow resulted to be the fundamental part for studies concerning, for example, recommender systems [4], [5], [6], extraction of code in informal documentation [7], post quality modeling and analysis [2], [8], [9], or to understand how developers use APIs [10], [11].

A fundamental aspect of any mining approach involving Stack Overflow posts concerns the intrinsic heterogeneity of the data: Posts are composed of both *unstructured fragments* representing the natural language part of the discussion, and *structured fragments* representing (*e.g.,* Java code, XML, JSON), both co-existing in the same artifact. Separating these elements from each other is a necessary step to perform targeted analyses on quality data. Rigby *et al.* [7] exploit regular expressions to isolate and extract a set of Java constructs of interests (*i.e.,* method chains), which is conceptually similar to the source model construction of the reflexion model approach [12]. Alternatively, Bacchelli *et al.* [13] defined an island grammar [14] capable of identifying and parsing Java constructs and stack traces, separating them from natural language narrative.

However, the pure extraction of constructs of interest from natural language leaves a conceptual "hole" in the process. For example, an analysis can focus on identifying relationships between XML configuration files (*e.g.,* for the Android platform) and code samples, but after the extraction of structured fragments, the data comes still in the form of text. The next step is thus to model the extracted information and discover the semantic links among these elements to actually perform the specific analysis. In any case, and especially when the analysis involves code fragments, modeling requires full-fledged parsing of the structured fragments to reconstruct an AST, which must be done subsequently and separately. For example, a Java class fragment in a discussion does not provide direct information about its method invocations, but it rather requires additional parsing to extract this information.

In this data paper, we alleviate this burden by providing an already modeled dataset of Stack Overflow discussions. In particular, we fully exploit island parsing not only for fragment extraction but also for constructing a *heterogeneous abstract syntax tree (H-AST)*, whose nodes model the contents of a Stack Overflow discussions according to the kind of structured fragment it represents. The dedicated island grammar to extract and construct the H-AST all the Stack Overflow discussions tagged with the <java> tag. The H-AST includes the following possible structured fragments:

- **Java constructs**, including incomplete fragments like method and class declarations lacking the body;
- **Stack traces** and incomplete stack trace lines;
- **XML/HTML** documents, tags and elements;
- **JSON** fragments.

Moreover, our dataset contains already available meta-information for common analyses, like term frequency (*tf*) data, variable names, and mentioned reference types.

The dataset, named StORMeD, is provided as a set of JSON files, one for each discussion, in both pretty-printed and compact form. Moreover, we also include a documented Scala API that can be used to parse the JSON files and obtain objects corresponding to the HAST, that can be visited and processed to implement the desired analysis.

[1] https://archive.org/details/stackexchange
[2] after removing posts without owner defined
[3] http://goo.gl/nen09t

II. Dataset Construction

To create StORMeD, we rely on the Stack Overflow data dump of September 2014. We focus on the discussions targeting Java-related topics, and thus we focus on the 708K discussions tagged with the <java> tag.

A. Island Grammars and Parsing

Figure 1 show an example of a discussion. Each Stack Overflow discussion is stored as a sequence of HTML elements, each one identifying an *information unit*. The HTML tagging is performed by the authors and eventually modified by the crowd. Information units tagged with (and other similar tags) mainly contain narrative, while the ones tagged with <code> should contain structured fragments, *e.g.,* Java snippets. However, users often tag with <code> arbitrary contents, *e.g.,* error messages like in Figure 1. Thus, units tagged with <code> may contain narrative. We decided to apply *island parsing* in both cases to reconstruct the H-AST.

I am migrating from xml based spring configuration to "class" based configuration using the corresponding **@Configuration** annotation.

I came across the following problem: I want to create a new bean, which has a reference to another (service) bean. Therefore I autowired this class to set this reference during bean creation. My configuration class looks as follows:

<code>

```
@Configuration
@ComponentScan(basePackages = {"com.akme"})
public class ApplicationContext {

   @Resource
   private StorageManagerBean storageManagerBean;

   @Bean(name = "/storageManager")
   public HessianServiceExporter storageManager() {
      HessianServiceExporter hessianServiceExporter =
                                    new HessianServiceExporter();
      hessianServiceExporter.setServiceInterface(StorageManager.class);
      hessianServiceExporter.setService(storageManagerBean);
      return hessianServiceExporter;
   }
}
```

</code>

But this doesn't work, because the causes a **BeanNotOfRequiredTypeException** exception during startup.

<code>

```
Bean named 'storageManagerBean' must be of type
[com.akme.StorageManagerBean], but was actually of type
[com.sun.proxy.$Proxy20]
```

</code>

The **StorageManagerBean** is annotated with an **@Service** annotation. And the xml based configuration worked as expected:

<code>

```
<bean name="/storageManager"
  class="org.springframework.remoting.caucho.HessianServiceExporter">
  <property name="service" ref="storageManagerBean"/>
  <property name="serviceInterface" value="com.akme.StorageManager"/>
</bean>
```

</code>

Fig. 1. Example of Stack Overflow question with HTML tagging.

Island grammars are grammars that contain detailed rules describing the constructs of interest (the *islands*), and generic productions that capture the remainder (the *water*), in our case the natural language narrative [14]. We extend the approach by Bacchelli *et al.* [13] by implementing an island grammar able to parse not only Java and stack traces, but also XML and JSON fragments, which we found frequently appearing in Java related discussions. We ran the island parser on each discussion, composed of posts (questions or answers) and comments to construct a H-AST. Parsing happens in two steps.

First, we parse the post representation using **HTML tag rules** to extract the information units. We keep track of the human tagging and we generalize the tagged contents to two different types of information units:

- **Text Unit**: Whatever is not tagged as <code> at the top level, like textual decorations (*e.g.,* <b>, <hr>), lists (*e.g.,* , <ul>), paragraph (*i.e.,*).
- **Structured Fragment Unit**: Every contents tagged as <code> at the top level. It exposes a H-AST node modeling the island identified by the parser.

The second phase concerns the effective use of the heterogenous island grammar. Once we identified the units we run the the island parser to identify the constructs of interest. In the case of a textual fragment, the goal is to identify likely short fragments of non textual information. For example, the user can refer to a method invocation or a type, and we are able to capture this information by using island parsing. When it comes to *structured fragments*, the process is similar. We perform a full parsing of the contents and we collect all the identified constructs in a H-AST island node.

B. Strict Parsing

In Stack Overflow, all structured constructs are essentially immersed in natural language. Thus, we need to reduce as much as possible possible ambiguities. When considering the Java language, constructs like class declarations or method declarations are easily separable from the natural language. However, when we want a finer granularity of the constructs and extract (*e.g.,* method invocations, qualified identifiers, or class mentions), ambiguity increases. Consider method invocations: If we follow the Java language grammar, thus admitting spaces between tokens, the island parser would extract from the string "the list (1,2,3) is a list of integer" the invocation *list(1,2,3)*. To avoid this problem, we require the absence of spaces between the method name and the parenthesis.

Similarly, we exploit naming conventions to extract likely mentions of classes, using some heuristics to solve ambiguities. For example, if the mention is not qualified (*i.e.,*, without the package location), we require two camel case instances (*e.g.,* ArrayList) to avoid extracting all words starting with uppercase in the narrative. However, if the mention is qualified (*e.g.,* java.lang.String) we relax the camel case constraint but we ensure that the identifiers have no spaces (*e.g.,* "... package. This is"). Finally, we relax the constraint if the likely class has a valid type parameter (*e.g.,* List<String>) but requiring that the left angular parenthesis is not separated by a space (to exclude ambiguities with XML/HTML tags).

Java annotations cannot be searched in the text as-is. In fact, on Stack Overflow, it is common to refer to a user with @<*username*>. This construct is ambiguous with annotations.

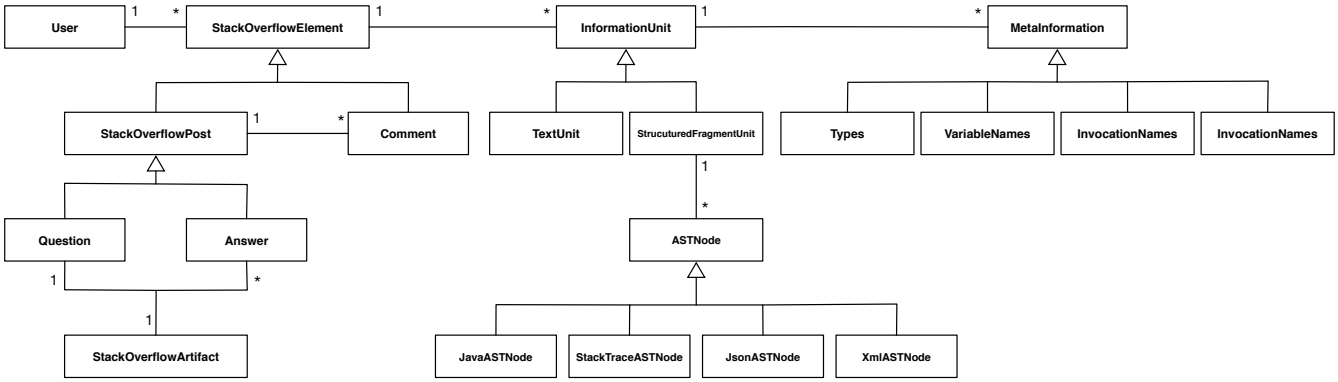

Fig. 2. Partial Object Model of the dataset

To solve the ambiguity, we enforce similar constraints to the one used for class mentions (since the annotations follow the same naming conventions).

C. Simple Meta Information Mining

At the end of the island parsing phase we obtain all the H-AST nodes for the information units. To enrich the dataset, and to provide ready made data for simple analyses, we run a visitor to extract simple meta information, like term frequency data or the overall types mentioned in a unit.

III. Data Schema

Figure 2 shows the partial object model of our dataset. The island parser and the model have been implemented in the Scala language using the parboiled2[4] parser. Once we compute the meta information, we serialize every object representing the Stack Overflow discussion as JSON object which is then exported to a text file. The total time required to extract the data from the dump, parsing it, and writing to disk took approximately 5.5 hours on a server with 16 Intel 2.10Ghz Xeon processors and 128 GB of RAM.

Table I shows some simple statistics for the dataset, showing the occurrences of some interesting H-AST nodes. The raw uncompressed data dump consists of 90G of pretty-printed JSON files, reduced to 29GB in the case of compact ones.

Node Type	Occurrences	Node Type	Occurrences
Java Nodes		**XML Nodes**	
Identifiers	41,350,546	Nodes	1,306,372
Reference Types	7,403,224	Attributes	1,374,014
Variable Declarators	3,377,317	Comments	40,069
Expressions	14,992,718	**JSON Nodes**	
Operators	3,496,329	Members	126,459
All Statements	6,055,351	Objects	49,519
Primitive Types	1,476,423	**Text Nodes**	
Blocks	1,927,615	Fragments	15,492,077
Method Declarators	1,073,261	**Stack Trace Nodes**	
Import Declarators	622,603	Lines	923,630
Annotations	993,909	Traces	29,377
Class Declarators	969,021		
For Loops	181,243		
Try Catch Statements	99,719		
While Loops	85,625		
Interface Declarators	21,908		
Enum Declarators	8,513		
Synchronized Statements	7,366		
Do While Loops	4,207		
Assertion Statements	1,799		
Annotation Declarators	1,374		

TABLE I
H-AST NODES OCCURRENCES IN THE DATASET.

A. Meta Information

Every information unit exposes a set of precomputed meta information obtained by traversing the H-AST. In StORMeD, we provide the following precomputed meta information:

- **Types**, containing the set of Java types mentioned in a unit, including qualified types (reference types) and primitive types (*e.g.,* int, double);
- **Variable Names**, containing all the H-AST nodes matching a variable (or field) name;
- **Invocation Names**, containing all the H-AST nodes matching a method invocation;
- **Natural Language**, containing the term frequency (*tf*) vector that can be used to calculate, for example, textual similarities. The *tf* vector is generated using Apache Lucene[5]. We split text on case change, on digits and symbols, we lower the case, we remove stop words, and we apply the snowball stemmer[6] to the obtained terms.

B. The StORMeD Kit

The StORMeD Kit is available at http://stormed.inf.usi.ch. Together with the two JSON datasets, one prettified, and one non-prettified, we provide a developer kit containing an API for Scala, but usable also with Java representing the object model. The API allows to deserialize JSON files into Scala instances, provides simple visitors for the H-AST, and also allows to implement custom visitors.

[4]https://github.com/sirthias/parboiled2

[5]http://lucene.apache.org

[6]http://snowball.tartarus.org/

IV. Limitations

The main limitations of our dataset concerns the typology of data we provide. Comparing this dataset with the original data dump of Stack Overflow, we do not include all the meta-data concerning the community. We map posts to users, but we do not include votes and badges. However, given the JSON nature of the dump, this information can be easily added by the end-user by managing the dataset with a non-relational database like MongoDB or CouchDB[7]. Another limitation concerns the parsing phase. As explained in Section II, we apply strict parsing to some constructs of interests, in particular Java types and qualified names. Even though the adopted rules are conservative, it is not possible to reach perfect precision, and some corner cases may appear in the dataset, for example when some spurious fragments match the heuristics we used for class name fragments (*e.g.,*, *PRyLwCgqd*). By a manual inspection of a sample of documents, we found that examples similar to the aforementioned one are rare, therefore the limitation should not affect the reliability of the dataset. We plan to conduct a detailed study on the quality of our dataset.

V. Research Opportunities

An explicit advantage concerns the construction and replication of mining experiments, since our dataset is essentially ready made. Without our dataset, a researcher would need to know which data needs to be parsed a priori, construct an extractor, parse the extracted fragments and construct a model. If the extracted information is not suitable for the experiment, the researcher needs to modify the parser and repeat the process. With our dataset, the researcher only needs to traverse the H-AST, which requires definitely less computational effort than parsing. Analyzing a dataset like Stack Overflow could become a computationally intensive process, in particular if we consider the parsing process. With StORMeD, researches do not have to care about the extraction of structured information from text, but instead, they can focus on analysis. In particular, our dataset easily enables combined analyses, which is an interesting topic for the MSR community. For example, according to the CFP of the mining challenge of 2015[8], one suggested idea is *"to compare the predictive power of three settings on the number of votes on a Stack Overflow question: natural language text alone, code fragments alone, and the combination of text and code fragments"*. A competitor in the mining challenge could collect text fragments and real code fragments by simply traversing the H-AST provided by our dataset to then run her approach and evaluate it. A competitor could take advantage of the meta-information already available (*i.e.,* code types) to improve the approach.

VI. Conclusion

In this paper we presented StORMeD, a dataset for Stack Overflow that models the posts by building a H-AST for each discussion in the data dump. Our dataset allows to navigate the contents of a discussion by differentiating among Java code, XML, JSON, stack traces, and natural language fragments. It also provides ready made meta-information like term frequency vectors and mentioned types, and implicitly enables combined analyses that leverage the heterogeneity of Stack Overflow posts.

Acknowledgments. Ponzanelli and Lanza thank the Swiss National Science foundation for the financial support through SNF Project ESSENTIALS, No. 153129.

[7]See http://www.mongodb.org/ and http://couchdb.apache.org/

[8]http://2015.msrconf.org/challenge.php

References

[1] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest Q&A site in the west," in *Proc. of CHI 2011 (29th Conference on Human factors in computing systems)*. ACM, 2011, pp. 2857–2866.

[2] L. Ponzanelli, A. Mocci, A. Bacchelli, and M. Lanza, "Understanding and Classifying the Quality of Technical Forum Questions," in *Proceedings of QSIC 2014 (14th International Conference on Quality Software)*. IEEE CS Press, 2014, pp. 343–352.

[3] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web? (nier track)," in *Proceedings of ICSE 2011 (33rd International Conference on Software Engineering)*, ACM, Ed., 2011, pp. 804–807.

[4] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza, "Mining StackOverflow to Turn the IDE into a Self-confident Programming Prompter," in *Proceedings of MSR 2014 (11th Working Conference on Mining Software Repositories)*. ACM, 2014, pp. 102–111.

[5] J. Cordeiro, B. Antunes, and P. Gomes, "Context-based Recommendation to Support Problem Solving in Software Development," in *Proceedings RSSE 2012, (3rd International Workshop on Recommendation Systems for Software Engineering)*. IEEE Press, 2012, pp. 85–89.

[6] M. Rahman, S. Yeasmin, and C. Roy, "Towards a context-aware IDE-based meta search engine for recommendation about programming errors and exceptions," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, 2014, pp. 194–203.

[7] P. C. Rigby and M. P. Robillard, "Discovering essential code elements in informal documentation," in *Proceedings of ICSE 2013 (35th International Conference on Software Engineering)*. IEEE Press, 2013, pp. 832–841.

[8] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton, "Improving low quality stack overflow post detection," in *Proceedings of ICSME 2014 (30th International Conference on Software Maintenance and Evolution, Industrial Track)*, 2014, pp. pp. 541–544.

[9] D. Correa and A. Sureka, "Chaff from the Wheat : Characterization and Modeling of Deleted Questions on Stack Overflow," in *Proceedings of WWW 2014 (23rd international conference on World Wide Web*. ACM, 2014.

[10] W. Wang and M. W. Godfrey, "Detecting API usage obstacles: a study of iOS and Android developer questions," in *Proceedings of MSR 2013 (10th International Working Conference on Mining Software Repositories)*. IEEE, 2013, pp. 61–64.

[11] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in *Proceedings of MSR 2014 (11th Working Conference on Mining Software Repositories)*. ACM, 2014, pp. 112–121.

[12] G. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: bridging the gap between design and implementation," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, pp. 364–380, Apr 2001.

[13] A. Bacchelli, A. Cleve, M. Lanza, and A. Mocci, "Extracting structured data from natural language documents with island parsing," in *In Proceedings of ASE 2011 (26th IEEE/ACM International Conference On Automated Software Engineering)*, 2011, pp. 476–479.

[14] L. Moonen, "Generating robust parsers using island grammars," in *Proceedings of WCRE 2001 (8th Working Conference on Reverse Engineering)*. IEEE CS, 2001, pp. 13–22.