A Method for Removal of Smells by Refactoring Web Mashups

R.Kiruthiga, A Alice Gavya
Department of Computer Science and Engineering
Kalaignar Karunanidhi Institute of Technology Coimbatore 641402

Abstract

Web mashups are the web pages which are created by blending information from multiple sources. Yahoo! Pipes, the tool used for creating mashups suffers from various drawbacks like creation of smells which may lead to software failure. This work aims to study various complexities and common deficiencies in the user programmed web mashups, to identify the prevalence of code smells. To improve the maintainability, understandability and reusability of mashups, the smells can be removed by applying refactoring method. The quality of mashups before and after applying refactoring will be compared.

I. INTRODUCTION

A web mashup, is a web page that uses information from multiple sources to create a single graphical interface. It gathers and integrates content from various existing sources to create a single service. For example, housingmaps.com is a mashup which provides information like apartments for rent or homes for sale along with a map on the single site. Google earth, social networking sites, online news services, online shopping are some of the sites that are making use of these mashups. The major advantage of using these mashups is its low cost to develop.

There are three types of mashups that are widely in use. They are

(i) Business mashup
(ii) Customer mashup
(iii) Data mashup

Business mashups defines applications that combine its own data with that of other external services. Consumer mashups combines data from multiple public sources. For example Wikipediavision combines Google Map and Wikipedia API. Data mashups combines data from multiple sources and provides an entirely distinct service.

A. Yahoo! Pipes

Yahoo! Pipes is a web application which is used to create web mashups by integrating data from multiple sources. It enables users to "pipe" information from different sources and then set up rules for modifying content. To program these mashups, users drag and drop predefined modules onto the canvas, connect the modules via wires, and parameterize the modules by setting their field values [1]. The modules perform various predefined functions, such as retrieving data from a web source (fetch) or selecting a subset of the retrieved data (filter), and act as interfaces to an API.

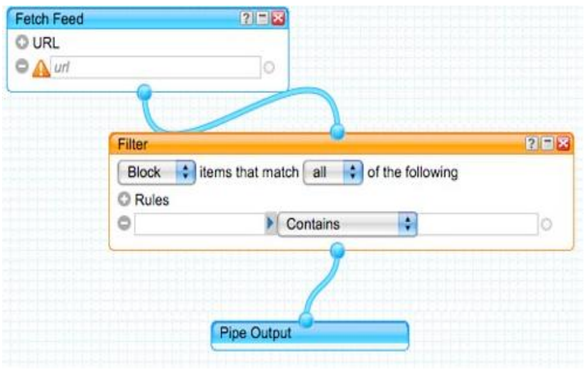


Fig.1. Sample yahoo! Pipes with Functions Fetch and Filter

B. Refactoring

Refactoring is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing the way it intended to work. Code refactoring is the process of restructuring existing computer code, without changing its external behavior, to improve the nonfunctional attributes of the software and also to create a more expressive internal architecture or object model to improve extensibility.

Code smell is an indication that there exists a problem in the code. Some of these code smells are present since its creation while some may be introduced when adding new features. Once recognized, such problems can be addressed by refactoring the source code, or transforming it into a new form that behaves the same as before but without any smells. The two general categories of benefits to the activity of refactoring are (i) Maintainability as it is easier to fix bugs since the source code is simple to read and the intent of its author is easy to grasp and (ii) Extensibility as it is easier to extend the capabilities of the application if it uses recognizable design patterns.

Prior to applying refactoring to a section of a code, unit tests are performed to ensure that the behavior of the module is correct before applying refactoring. If the test fails, it is better to fix the test and then perform refactoring because it may create the overhead of distinguishing between failures introduced by refactoring and failures that were already present.

With the help of refactoring techniques mashups created using Yahoo! Pipes are checked to find code smells and if they are present, they can be removed. Thus user can get the mashups which are easy to read, maintain and reuse.

C. Problem of Smells

In spite of increasing power and popularity of mashup environments, it has been observed that mashup programs tend to suffer from common deficiencies, such as being unnecessarily complex, using inappropriate or dated modules or sources of data, assembling non standard patterns, and duplicating values and functionality.

The presence of such deficiencies in mashup applications is not necessarily surprising, given the inexperience of the users. The deficiencies that end-user programmers of mashups encounter have similarities with those found by professional programmers and are often referred to as code smells—indications that something may be wrong with a section of code. Software engineers have at their disposal techniques and tools to maintain their code and address such smells by performing semantic preserving transformations on their programs to remove smells, a process called refactoring. This work focuses on understanding mashup maintenance through automated smell identification and refactoring. The smells that increase the complexity of the pipes can be identified and refactoring is applied to make the mashups easy to understand, maintain and reuse.

William f. opdyke et al. describes the importance of refactoring in designing application frameworks with the help of Typed Smalltalk projects and found that the disadvantage is that the examples include some extraneous detail and complexity, but has the advantage that the examples show how refactoring are applied in actual design tasks[2]. They explained the three of the most complex refactorings in detail which includes generalizing the inheritance hierarchy, specializing the inheritance hierarchy and using aggregations to model the relationships among classes. The factors which make refactoring object-oriented frameworks difficult are (i)there is no theory of how people refactor frameworks and the kinds of refactorings they make (ii) some refactoring operations are difficult to perform (iii) difficult to solve the conflicts in behavior. The advantages of using these object-oriented frameworks are (i) it defines a set of program restructurings that people can apply to frameworks (iii) it shows how to automatically support refactorings in a way that preserves the behavior of a program.

T. Mens et al. analysed the dependencies between refactorings which can help the developers to identify the most suitable refactoring for their work [3]. They used AGG, a state-of-the-art graph transformation tool which has the built-in critical pair analysis algorithm and explored how critical pair analysis can help to detect and analyse conflicts and dependencies between refactorings. The drawbacks consisted of (i) some of the conflict situations where not detected because of insufficient specifications (ii) sequential dependency analysis still remains a manual process (iii) Problem of how to deal with conflicts once they have been detected.

Web mashups are created in thousands daily as the programmers creating new applications need to blend information from more than one site. The web mashup language that is been used widely suffers from various drawbacks which prevents the common people from using it. In Yahoo! Pipes, the pipes should be designed in such a way that it can be reused any number of times with slight modifications. The design should be efficient such that it makes use of fewer modules. Smells, which are the unnecessary coding that creates problem, should be found in advance and removed so that the newly designed mashup performs well without any error. Thus refactoring, which is the process of modifying the existing code, helps the users to create mashups that are reusable and easy to maintain without any complexities.

II. REFACTORING METHOD

To address the most prevalent code smells, a set of semantic preserving pipe refactorings have been defined. The smells are defined in terms of presence or absence of field values, modules, wires between modules and the values passed between modules. Two pipes are semantically equivalent, if the set of unique items that reaches each pipe's final output module are the same, ignoring duplicate items and items' order. Both preconditions and post conditions are specified for each transformation, which ensures that the internal behavior of the pipe is preserved. The refactoring is applied only if the conditions can be satisfied. Each refactoring transformation is decomposed into a set of more basic transformations that are applied to the smelly pipes. The refactored pipes will be free from the smells that caused the complexity of the pipes.

The proposed approach aims to

- Identify and define the most prevalent smells in Yahoo! Pipes environment.
- Design of domain-specific transformations to refactor smelly pipe-like mashups.

A. *Detection of Smells in Pipes*

Three different types of smells are identified in the mashups[4]. They are

1) Lazy smell
2) Redundancy smell
3) Environment smell

1) *Lazy Smell Detection*

These smells contain the modules or fields that do not lead to the output of the pipe, making it more complex. Lazy smells include (i) Noisy module and (ii) Unnecessary module. Noisy module is a module that has unnecessary fields like empty field and duplicated field, making the pipe harder to read and less efficient to execute. Empty field describes a blank field in a generator module. Duplicated field describes the case when two fields in a single module have the same value. Unnecessary module is a module whose execution does not affect the pipe's output. It includes (i) Cannot reach output describes a module that does not contain an outgoing wire to the output module, thus from this module output cannot be reached. (ii) Ineffectual path altering describes a path altering module with exactly one input and output wire. (iii) Swaying module are the modules that do not provide any data to the pipe and do not receive any incoming data.

2) *Redundancy Smell Detection*

Redundancy smells include duplicate strings, duplicate modules, identical subsequent operators and identical parallel operators. These smells when not identified and removed will add complexity to programs and affects readability and reusability

3) *Environment Smell Detection*

Environment smells include deprecated module which is a module that is no longer supported by the pipe environment and invalid source which is an external source when attempted to retrieve data result in error like 404 Not Found.

B. *Refactoring*

Refactoring is the process of making some changes to the code such that it reduces unnecessary complexities and improves readability thereby making it more efficient.

1) *Refactoring Lazy Smell:*

When the Lazy smells are identified the next step is to apply refactoring. Refactoring removes empty or duplicated fields within a module and then the non contributing modules if present, are removed.

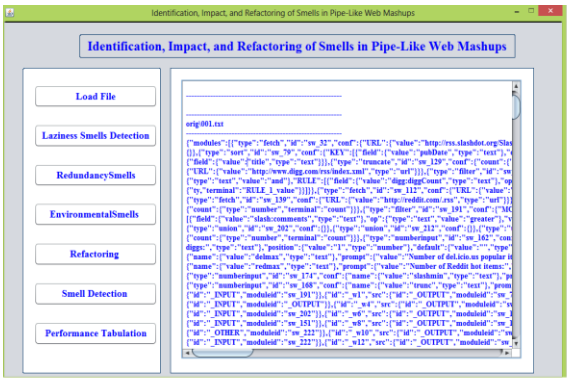
2) *Refactoring Redundancy Smell:*

When the redundant smells are detected refactoring should be applied to remove these smells. This can be done by merging redundant modules and by aggregating paths to a single path to simplify the pipe structure

3) *Refactoring Environment Smell:*

Environmental errors are detected and then the refactoring is applied which removes these smells by replacing deprecated modules and invalid sources.

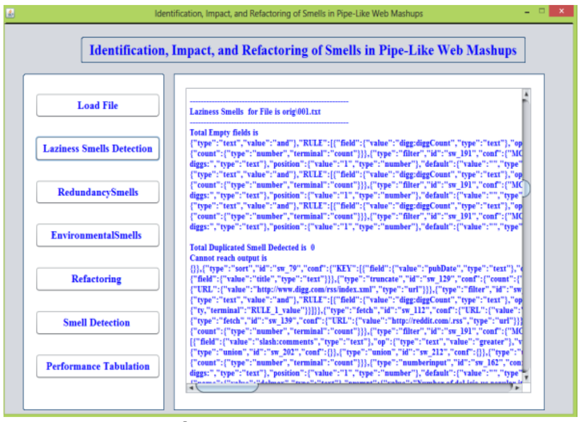
III. EXPERIMENTAL RESULTS


Identification, Impact, and Refactoring of Smells in Pipe-Like Web Mashups
Identification, Impact, and Refactoring of Smells in Pipe-Like Web Mashups
Load File
Laziness Smells Detection
RedundancySmells
EnvironmentalSmells
Refactoring
Smell Detection
Performance Tabulation

```
orig\001.txt
{"modules":[{"type":"fetch","id":"sw_32","conf":{"URL":{"value":"http://rss.slashdot.org/Slas
{}},{"type":"sort","id":"sw_79","conf":{"KEY":[{"field":{"value":"pubDate","type":"text"},"
{"field":{"value":"title","type":"text"}}},{"type":"truncate","id":"sw_129","conf":{"count":{
{"URL":{"value":"http://www.digg.com/rss/index.xml","type":"url"}}},{"type":"filter","id":"sw
{"type":"text","value":"and"},"RULE":[{"field":{"value":"digg:diggCount","type":"text"},"op
{"ty,"terminal":"RULE_1_value"}}]}},{"type":"fetch","id":"sw_112","conf":{"URL":{"value":"
{"type":"fetch","id":"sw_139","conf":{"URL":{"value":"http://reddit.com/.rss","type":"url"}}
{"count":{"type":"number","terminal":"count"}}},{"type":"filter","id":"sw_191","conf":{"MC
[{"field":{"value":"slash:comments","type":"text"},"op":{"type":"text","value":"greater"},"v
{"type":"union","id":"sw_202","conf":{}},{"type":"union","id":"sw_212","conf":{}},{"type":"
{"count":{"type":"number","terminal":"count"}}},{"type":"numberinput","id":"sw_162","con
diggs:","type":"text"},"position":{"value":"1","type":"number"},"default":{"value":"","type"
{"name":{"value":"delmax","type":"text"},"prompt":{"value":"Number of del.icio.us popular it
{"name":{"value":"redmax","type":"text"},"prompt":{"value":"Number of Reddit hot items:",
{"type":"numberinput","id":"sw_174","conf":{"name":{"value":"slashmin","type":"text"},"p
{"type":"numberinput","id":"sw_168","conf":{"name":{"value":"trunc","type":"text"},"prom
{"id":"_INPUT","moduleid":"sw_191"}},{"id":"_w1","src":{"id":"_OUTPUT","moduleid":"sw_
{"id":"_INPUT","moduleid":"_OUTPUT"}},{"id":"_w4","src":{"id":"_OUTPUT","moduleid":"sw
{"id":"_INPUT","moduleid":"sw_202"}},{"id":"_w6","src":{"id":"_OUTPUT","moduleid":"sw_
{"id":"_INPUT","moduleid":"sw_151"}},{"id":"_w8","src":{"id":"_OUTPUT","moduleid":"sw_
{"id":"_OTHER","moduleid":"sw_222"}},{"id":"_w10","src":{"id":"_OUTPUT","moduleid":"sw
{"id":"_INPUT","moduleid":"sw_222"}},{"id":"_w12","src":{"id":"_OUTPUT","moduleid":"sw_
```

Fig.2. Loading the JSON Representation of the Pipe

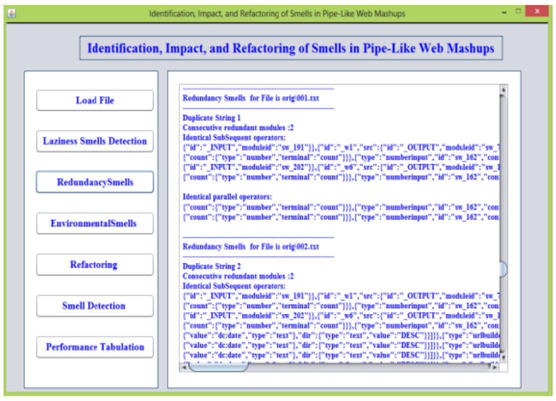
The first step is the loading of JSON representation of the pipe. JSON is derived from Javascript and is language-independent data format [5]. It is the most common data format used for asynchronous browse, replacing XML which is used by AJAX.


Identification, Impact, and Refactoring of Smells in Pipe-Like Web Mashups
Identification, Impact, and Refactoring of Smells in Pipe-Like Web Mashups
Load File
Laziness Smells Detection
RedundancySmells
EnvironmentalSmells
Refactoring
Smell Detection
Performance Tabulation

```
Laziness Smells  for File is orig\001.txt
Total Empty fields is
{"type":"text","value":"and"},"RULE":[{"field":{"value":"digg:diggCount","type":"text"},"op
{"count":{"type":"number","terminal":"count"}}},{"type":"filter","id":"sw_191","conf":{"MC
diggs:","type":"text"},"position":{"value":"1","type":"number"},"default":{"value":"","type"
{"type":"text","value":"and"},"RULE":[{"field":{"value":"digg:diggCount","type":"text"},"op
{"count":{"type":"number","terminal":"count"}}},{"type":"filter","id":"sw_191","conf":{"MC
diggs:","type":"text"},"position":{"value":"1","type":"number"},"default":{"value":"","type"
{"type":"text","value":"and"},"RULE":[{"field":{"value":"digg:diggCount","type":"text"},"op
{"count":{"type":"number","terminal":"count"}}},{"type":"filter","id":"sw_191","conf":{"MC
diggs:","type":"text"},"position":{"value":"1","type":"number"},"default":{"value":"","type"

Total Duplicated Smell Dedected is  0
Cannot reach output is
{}},{"type":"sort","id":"sw_79","conf":{"KEY":[{"field":{"value":"pubDate","type":"text"},"
{"field":{"value":"title","type":"text"}}},{"type":"truncate","id":"sw_129","conf":{"count":{
{"URL":{"value":"http://www.digg.com/rss/index.xml","type":"url"}}},{"type":"filter","id":"sw
{"type":"text","value":"and"},"RULE":[{"field":{"value":"digg:diggCount","type":"text"},"op
{"ty,"terminal":"RULE_1_value"}}]}},{"type":"fetch","id":"sw_112","conf":{"URL":{"value":"
{"type":"fetch","id":"sw_139","conf":{"URL":{"value":"http://reddit.com/.rss","type":"url"}}
{"count":{"type":"number","terminal":"count"}}},{"type":"filter","id":"sw_191","conf":{"MC
[{"field":{"value":"slash:comments","type":"text"},"op":{"type":"text","value":"greater"},"v
{"type":"union","id":"sw_202","conf":{}},{"type":"union","id":"sw_212","conf":{}},{"type":"
{"count":{"type":"number","terminal":"count"}}},{"type":"numberinput","id":"sw_162","con
diggs:","type":"text"},"position":{"value":"1","type":"number"},"default":{"value":"","type"
```

Fig.3. Detection of Empty Fields and Duplicate Smells

The next step is the detection of lazy smells if it is present in the pipe. The JSON representation of the webpage is checked to identify and remove such smells.

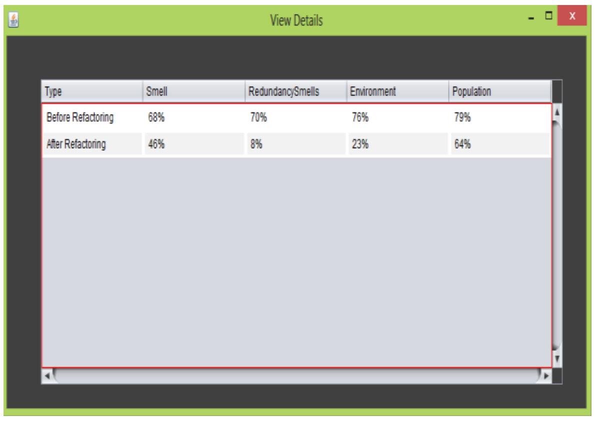

Identification, Impact, and Refactoring of Smells in Pipe-Like Web Mashups
Identification, Impact, and Refactoring of Smells in Pipe-Like Web Mashups
Load File
Laziness Smells Detection
RedundancySmells
EnvironmentalSmells
Refactoring
Smell Detection
Performance Tabulation

```
-----------------------------------------------------
Redundancy Smells  for File is orig\001.txt
-----------------------------------------------------
Duplicate String 1
Consecutive redundant modules :2
Identical SubSequent operators:
{"id":"_INPUT","moduleid":"sw_191"}},{"id":"_w1","src":{"id":"_OUTPUT","moduleid":"sw_7
{"count":{"type":"number","terminal":"count"}}},{"type":"numberinput","id":"sw_162","con
{"id":"_INPUT","moduleid":"sw_202"}},{"id":"_w6","src":{"id":"_OUTPUT","moduleid":"sw_1
{"count":{"type":"number","terminal":"count"}}},{"type":"numberinput","id":"sw_162","con

Identical parallel operators:
{"count":{"type":"number","terminal":"count"}}},{"type":"numberinput","id":"sw_162","con
{"count":{"type":"number","terminal":"count"}}},{"type":"numberinput","id":"sw_162","con


-----------------------------------------------------
Redundancy Smells  for File is orig\002.txt
-----------------------------------------------------
Duplicate String 2
Consecutive redundant modules :2
Identical SubSequent operators:
{"id":"_INPUT","moduleid":"sw_191"}},{"id":"_w1","src":{"id":"_OUTPUT","moduleid":"sw_7
{"count":{"type":"number","terminal":"count"}}},{"type":"numberinput","id":"sw_162","con
{"id":"_INPUT","moduleid":"sw_202"}},{"id":"_w6","src":{"id":"_OUTPUT","moduleid":"sw_1
{"count":{"type":"number","terminal":"count"}}},{"type":"numberinput","id":"sw_162","con
{"value":"dc:date","type":"text"},"dir":{"type":"text","value":"DESC"}}]}},{"type":"urlbuild
{"value":"dc:date","type":"text"},"dir":{"type":"text","value":"DESC"}}]}},{"type":"urlbuild
{"value":"dc:date","type":"text"},"dir":{"type":"text","value":"DESC"}}]}},{"type":"urlbuild
```

Fig 3. Detection of Redundant Modules

The next step is the detection and removal of redundant modules present in the pipe.

Fig.4. Detection of Invalid Sources



Type	Smell	RedundancySmells	Environment	Population
Before Refactoring	68%	70%	76%	79%
After Refactoring	46%	8%	23%	64%

Fig.5. Comparison Table

This tabulation gives the percentage of smells present before and after refactoring. Thus the provided JSON representation had 68% of lazy smells, 70% of redundancy smells and 76% of environment smells which are reduced to 46%, 8% and 23% respectively.

CONCLUSION

Mashups are being developed rapidly as it helps the end users to combine information from multiple sites into a single interface with a low cost and effort. However, the pipe-like mashups tend to suffer from various kinds of smells like lazy smells, redundancy smells and the environment smells. These smells are the unnecessary coding which creates problems when not identified and removed at the right time. Refactoring, which is the process of introducing some changes to the existing code, is the technique that is applied here to remove all the smells that are present. The performance tabulation shows the decrease in the percentage of smells that were present in the pipes. Thus it proves that the amount of smells present in the pipes can be reduced by using refactoring.

REFERENCE

[1] "Yahoo! Pipes", https://pipes.yahoo.com/pipes/.

[2] William F. Opdyke,"Refactoring Object-Oriented Frameworks", PhD thesis, University of llinois at Urbana-Champaign Champaign, USA, 1992

[3] T. Mens, G. Taentzer, and O. Runge, (2007), "Analysing Refactoring Dependencies Using Graph Transformation," in Software and Systems Modeling, vol. 6, no. 3, pp. 269-285.

[4] Kathryn T. Stolee, Sebastian Elbaum, "Refactoring Pipe-like Mashups for End-User Programmers", Software Engineering (ICSE), 2011 33rd International Conference on 21-28 May 2011

[5] JSON," http://www.json.org/.