

# Extraction of Dependencies from Javascript Files using High Performance Analysis

Neha Chauksey, Dr. A. S. Ghotkar

Department of Computer Engineering, PICT, Savitribai Phule Pune University, Maharashtra, India

## ABSTRACT

In various software development analyzers, it is necessary to know the dependencies amongst classes and other objects, so that the developers may get the information as to where are these dependencies currently in use, and which modules will be affected due to change in codes. This paper aims to build a dependency analyzer that takes multiple XML documents as the input, each representing a huge repository with an ever growing list of artifacts produced by large teams in an organization. This analyzer is designed such that it analyzes simple and recursive dependencies by utilizing the power of database to efficiently store and search the dependencies so as to represent the result in a tree format. This analyzer will act as an important decision making point for approving changes including the mitigation strategy for reducing the risk of change.

**Keywords:** Scripting languages, XML, Search interfaces, Restful web services

## I. INTRODUCTION

In the complex world of developing software, dependencies plays a key role. As the software keeps changing, each new change to one of its elements could cause regression in its dependents. Hence it becomes very crucial from a risk mitigation perspective to not only understand the dependencies between layers but also within those layers that makes up that software. Therefore an analyzer for extracting these dependencies is required.

Performance is one of the biggest challenges while performing recursive dependency analysis on a large dependency tree (XML). The results of a dependency analyzer are usually cryptic and hence would not be easy to understand. Aggregating the dependencies of multiple projects and performing analysis is cumbersome. The large dependency trees are usually time consuming to parse. The recursive analysis are complicated and time consuming on large trees and requires a high performing analyzer. Presenting the results with a high performing user interface is a challenge. This dependency analyzer is designed to not only scale in terms of searching the dependencies across an ever growing list of repositories but also to display huge results with optimal

performance using pagination techniques. A rich user interface that provides an easy to view dependency tree which accommodates on demand dependency information will provide a good user experience.

## II. LITERATURE SURVEY

Prior research has shown that customer reported software faults are often the result of violated dependencies that are not recognized by developers implementing software. Many types of dependencies and corresponding measures have been proposed to help address this problem. The objective of this research is to compare the relative performance of several of these dependency measures as they relate to customer reported defects.

Many direct attempts to address the issues related to this is not found, but rather a very few research papers that tangentially address these issues are available and discussed here.

Matthias Keil and Peter Thiemann [1] used formalization and abstract interpretation to perform a type-based dependency analysis for javascript. Results from this approach can be used to ensure confidentiality

and integrity of the data. Magnus Madsen, Benjamin Livshits and Michael Fanning [2] combined the pointer analysis and use analysis in order to obtain the analysis of Windows 8 javascript applications. This is performed so as to get the partial or full inference. Milos Savic et al. [3] showed that more precise software networks can be extracted with a tool in comparison with dependency finder that provides even language-independent extraction. They managed to do it using Dependency finder to extract software networks that belongs to different programming paradigms and fuzzy parsing mechanism.

Liang Huai Yang et al. [4] created a 2PXMIner. This tool was able to perform efficient and scalable query evaluation for frequent query patterns in an XML document. Kajal T. Claypool [5] used Label match Algorithm and Needleman Wunsch Algorithm to do approximate keyword search and context-specific searching providing a better precision and recall than exact keyword search. Binh Viet, Eric Pardede [6] proposed to extend the power and capability of XML with web service technologies and P2P architectures. Active XML (AXML) is extension of distributed XML databases. This was possible with the help of AXML-XML data exchange and AXML data query processing.

Tag extraction using DOM Structure, Knowledge base matching and classification when done simultaneously, average accuracy of 96.99% classification was obtained with less error rate by Krishna Murthy. A, Suresha [7]. Katalin Tunde et al. [8] used conceptual lattice, XFD Mining to mine functional dependencies through formal concept analysis which analyzes XML documents. Compare and rank was another method that provides speedy and accurate search in an XML document that is for E-Commerce Applications to locate document schemata. This method was proposed by Eric Jiu-Lin Lu and Yu-Ming Jung [9].

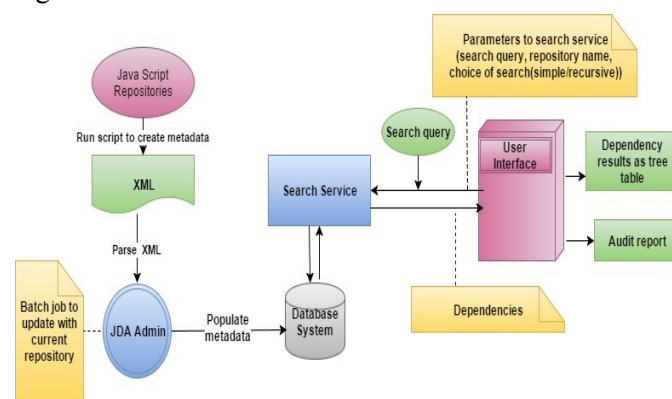
Describing XML Stream processing problems related to text processing and tree pattern matching using process XPath expressions with automata and pushdown automata was suggested by Dan Suci [10]. Susumu Nishimura, Keisuke Nakano [11] introduced the idea of attribute grammar composition and transformation and altSAX to develop stream transformation by giving the specifications for tree transformation. There were many

limitations for memory and boundness seen while adapting this approach. Multi-query evaluation and query compaction was the technique used by Jun-Ki Min et al. [12] that supports wide class of XPath queries for tree shaped expressions, order-based and nested predicates.

As of now, there is diversified research in all the techniques and platform this paper suggests. Particularly for analyzing javascript files that too from an input holding relational database approach has not being executed.

### III. METHODOLOGY

This system architecture aims to build a dependency analyzer that takes multiple XML documents as the input, each representing a huge repository with an ever growing list of artifacts produced by large teams in the organization.



**Figure 1.** Proposed Javascript Dependency Analyzer (JDA) Architecture

This analyzer is designed such that it analyzes simple and recursive dependencies by utilizing the power of database to efficiently store and search the dependencies so as to represent the result in a tree format. It will help programmers, testers and managers to efficiently analyze the risk from each of their perspective so that effective mitigation strategies can be applied for each change depending upon their occurrence in the software development lifecycle.

Fig. 1 shows the architecture of the proposed Javascript Dependency Analyzer (JDA).

The proposed system is modelled in two phases for getting a clear vision of the tasks the analyzer has to perform.

#### A. Phase I

- Evaluate different analyzers to handle huge XML
- Perform literature survey for gap identification
- Design and implement Rest based client for multi-user scenario
- Implement simple search through the database
- Design and implement audit trailing mechanism for every search query

In this analyzer, the input is an XML file that contains function metadata of the javascript files. This XML have details of functions i.e. where they are called from, declared and instantiated. The XML is parsed and all value from XML tags are stored in a JDA database. This database schema is predefined because the XML schema is fixed. The JDA database is populated and managed by the following two ways:

- JDA Admin
- JDA Search Service

**JDA Admin:** This module is a Java batch job that is designed to populate the JDA database. The input XML file is parsed with the help of StAX parser, in order to extract appropriate information of dependencies of various functions present in that particular repository. The job of this admin module is to populate the database not just once but in equal intervals so that updates in repository should be tracked.

**JDA Search Service:** In the search service, queries are fired in terms of function name for which the dependencies are to be found. This module will be able to execute the queries successfully if and only if the JDA Admin module has performed its operation. Phase I of this JDA analyzer aims to provide a vertical slicing of back-end services by getting the results through rest APIs. These APIs are designed for showing simple search results that contain the information of dependencies. Reports of audit i.e. tracking the search query so as to generate a report of what the users have searched for.

#### B. Phase II

- Code re-factoring
- Enhance the analyzer for multiple XML documents
- Implement recursive search through the database
- Design and implement the User Interface using OpenUI5 libraries and components

In this phase, the structure and implementation of first phase will be enhanced by re-factoring the classes and codes. The analyzer should be able to process multiple XML files, to get the results from different versions of the repository. A recursive search is to be designed in order to get the nested dependencies among the functions. And finally the user interface using OpenUI5 libraries and components will be created to make a smoother user experience. The results are planned to be displayed in a form of tree so as to make it visible in a user readable format.

### IV. RESULTS AND DISCUSSION

#### A. Huge XML Parser Analysis

In this paper huge XML files are taken as input to the system. In order to parse this various parser were tested but sTax parser worked really well for the requirement. The StAX specification defines a number of use cases for the API:

- Data binding
  - Unmarshalling the XML document
  - Marshalling the XML document
  - Parallel document processing
  - Wireless communication
- Virtual data sources
  - Viewing as XML data stored in databases
  - Viewing data in Java objects created by XML data binding
  - Navigating a DOM tree as a stream of events
- Parsing specific XML vocabularies
- Pipelined XML processing

The execution time required for parsing the XML file was comparatively lesser than DOM parser. Its navigation as stream of events helped understanding the schema well and creation of models became much easier for other backend processing.

## B. MyBatis & Postgres SQL Performance Analysis

Unlike Object-relational mapping frameworks, MyBatis does not map Java objects to database tables but Java methods to SQL statements.

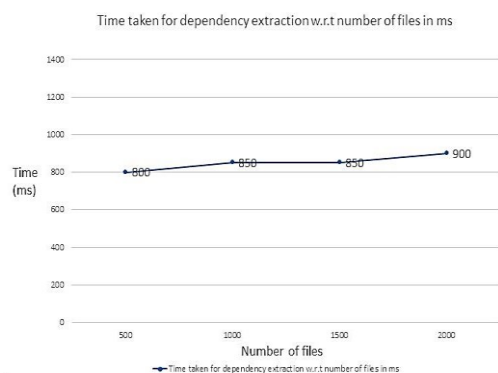
MyBatis lets you use all your database functionality like stored procedures, views, queries of any complexity and vendor proprietary features. It is often a good choice for legacy or de-normalized databases or to obtain full control of SQL execution. It simplifies coding compared to JDBC. SQL statements are executed with a single line.

MyBatis provides a mapping engine that maps SQL results to object trees in a declarative way. SQL statements can be built dynamically by using a built-in language with XML-like syntax or with Apache Velocity using the Velocity integration plugin. MyBatis integrates with Spring Framework. This feature allows one to build business code free of dependencies.

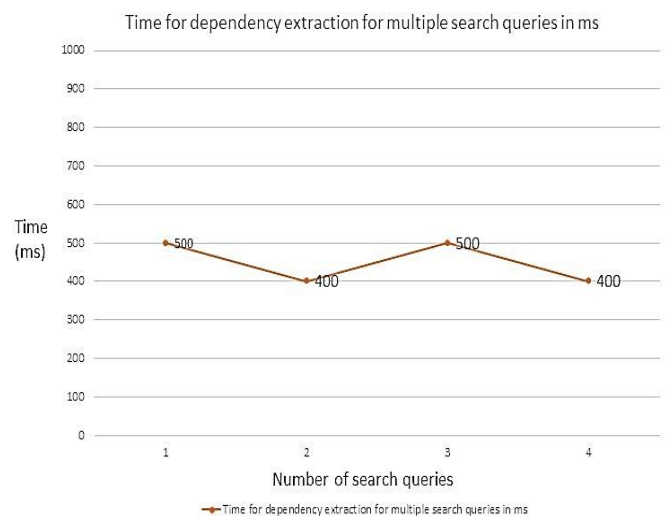
MyBatis supports declarative data caching. A statement can be marked as cacheable so any data retrieved from the database will be stored in a cache and future executions of that statement will retrieve the cached data instead hitting the database. MyBatis provides a default cache implementation based on a Java HashMap and default connectors for integrating with:

OSCache, Ehcache, Hazelcast and Memcached. It provides an API to plug other cache implementations. Without using Mybatis and connection pooling the database was uploaded as 1700 records in 45 min

Whereas, when the feature of pooling of connections to the postgres for each records in the XML was used, the population of database for 1700 records in 1 min



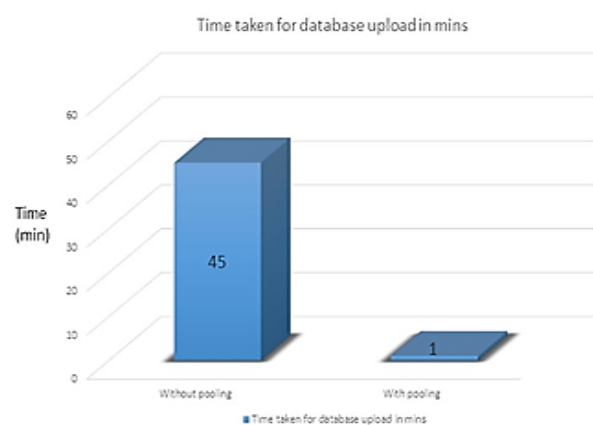
**Figure 2.** Graph Showing Time taken to Extract Dependencies with respect to Number of Files



**Figure 3.** Graph Showing Time taken to Extract Dependencies for Multiple Search Queries

## V. CONCLUSION

The proposed javascript dependency analyzer represents an approach to get the dependencies of the functions to the programmer. It is capable of handling huge and multiple XML files with lesser complexity. The proposed design includes usage of postgresSQL relational database, which in turn produces cost effective and faster results. This kind of analyzer will be beneficial for the programmers to make important decisions for approving changes including the mitigation strategy for reducing the risk of change.



**Figure 4.** Graph Showing Time taken for Database Upload with and without Connection Pooling

## VI. FUTURE ENHANCEMENTS

The research can be extended to explore the details of javascript files so as to create a superset analyzer for javascript. If a script for multiplatform coding languages is designed, this analyzer can be utilized for all object oriented programming languages.

## VII. REFERENCES

- [1]. Matthias Keil and Peter Thiemann, "Type-based Dependency Analysis for javascript", in ACM 978-1-4503-2144-0/13/06, June 2013.
- [2]. Magnus Madsen, Benjamin Livshits and Michael Fanning, "Practical Static Analysis of javascript Applications in the Presence of Frameworks and Libraries", in Microsoft Research Technical Report, MSR- TR-201266.
- [3]. Milos Savic, Gordana Rakic, Zoran Budimac, Mirjana Ivanovic, "A language-independent approach to extraction of dependencies between source code entities", in Elsevier: Information and Software Technology, 2014.
- [4]. Liang Huai Yang, Mong Li Lee, Wynne Hsu, Decai Huang, Limsoon Wong, "Efficient Mining of frequent XML query patterns with repeatingsiblings", in Elsevier: Information and Software Technology 50 pp. 375389, 2008.
- [5]. Kajal T. Claypool, "SUSAX: Context-specific searching in XML documents using sequence alignment techniques", in Elsevier: Data and Knowledge Engineering 65 pp. 177-197, 2008.
- [6]. Binh Viet, Eric Pardede, "Active XML (AXML) research: Survey on the representation, system architecture, data exchange mechanism and query evaluation", in Elsevier: Journal of Network and Computer Applications, 2013.
- [7]. Krishna Murthy. A, Suresha, "XML URL Classification based on their semantic structure orientation for Web Mining Applications", in Elsevier: Procedia Computer Science 46 pp. 143-150 (2015).
- [8]. Katalin Tunde, Janosi-Ranez, Viorica Varga and Timea Nagy, "Detecting XML Functional Dependencies through Formal Concept Analysis", in IEEE transactions, Sept-20, 2010.
- [9]. Eric Jiu-Lin Lu and Yu-Ming Jung, "XDSearch: an efficient search engine for XML document schemata", in PERGAMO: Expert systems with Applications 24 pp. 213-224, 2003.
- [10]. Dan Suciu, "From searching text to querying XML streams", in Elsevier: Journal of Discrete Algorithms 2 pp. 17-32, 2004.
- [11]. Susumu Nishimura, Keisuke Nakano, "XML Stream transformer generation through program composition and dependency analysis", in Elsevier: Science of Computer Programming 54 pp. 257-290, 2005.
- [12]. Jun-Ki Min, Myung-Jae Park and Chin-Wan Chung, "XTREAM: an efficient multi-query evaluation on streaming XML data", in Elsevier: Information Sciences 177 pp. 3519-3538, 2007.