WebDiff: A Generic Differencing Service for Software Artifacts

Nikolaos Tsantalis, Natalia Negara, Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, Canada
{tsantalis, negara, stroulia}@ualberta.ca

***Abstract*—Software evolution analysis plays an important role in several software engineering activities which are mainly related to maintenance tasks. For this purpose, several approaches and tools have been developed to identify and analyze code changes throughout the history of a software system. However, most existing tools are restricted to a specific type of software artifact, are language-dependent and can only be locally executed on a client machine. To overcome these limitations we propose WebDiff, a web-based generic differencing service for various types of software artifacts including source code and UML diagrams.**

Keywords-differencing techniques; design evolution; software analysis; web-based tools

I. Introduction

Many software engineering activities, and maintenance in particular, rely on software evolution analysis in order to understand the development steps and design decisions that brought a given system to its current state. A core problem in software evolution analysis is the detection of specific changes that occurred between successive versions or releases of a system. To this end, several methods and tools that identify and analyze code changes have been developed which in general suffer from the following limitations:

- They mainly provide support for source code differencing and thus do not take into account other types of software artifacts, such as design documents (in the form of UML diagrams) and requirements specifications (in text form).
- They are language-dependent and thus cannot be easily extended to support other programming languages or software systems using multiple programming languages.
- They require to be installed, configured and executed on a client machine. Most of the times the installation and configuration process is a rather arduous task. Moreover, some of the tools consume a significant amount of CPU and memory resources from client's machine.

To overcome the aforementioned limitations we have developed a web-based system (WebDiff) supporting the comparison of various types of software artifacts. To achieve the required level of independence from the specific characteristics of the examined software artifacts, we have employed *VTracker* [1], a generic domain-independent tree differencing algorithm that is able to handle any kind of XML document representing a partially-ordered labeled tree. WebDiff currently offers three types of software differencing services:

1) Comparison of two different versions of source code for the extraction of design changes (i.e., addition/deletion/change of operations, attributes, inheritance relationships, operation calls and attribute accesses).
2) Comparison of two different versions of UML diagrams (exported in XMI format) for the extraction of design changes.
3) Comparison of source code with UML diagrams for the detection of changes or inconsistencies between the models represented in the code and design, respectively.

WebDiff offers a web-based user interface (WUI) as a frontend, through which the user can provide the required input to the services and obtain the results of the comparison. The WUI is Ajax-enabled giving the user the feeling of a desktop application user interface. The provided services are real-time in the sense that the user is informed about the status of the processed jobs through a progress bar and the results of the comparison are dynamically generated and reported to the user during the processing of the jobs. The user can interrupt a processed comparison at any time.

The rest of the paper is organized as follows: Section II presents an overview of the related work. In Section III we describe the architecture and the individual components of WebDiff. Section IV describes a typical usage scenario for the services offered by WebDiff. Finally, in Section V we conclude and present a list of our planned extensions to WebDiff.

II. Related Work

A number of software differencing approaches and tools have been developed to help the developers in getting an overview of the changes that occurred during software system evolution.

LDiff [2] is a language-independent line differencing tool that can be applied to analyze the changes between different kinds of software artifacts such as source code, use cases and test cases. The *LDiff* algorithm adopts Unix *diff* which is based on the longest common subsequence algorithm, and employs a variety of hunk similarity metrics (Cosine, Jaccard, Dice, and Overlap) and different text extraction techniques (chars, words, C/C++ tokens, n-grams). The tool supports side-by-side visualization of the detected differences and provides a graphical map of the differences for quick navigation. The approach is not able to characterize the type of the changes based on the semantics of the code. Moreover, it requires the specification of hunk and line similarity thresholds.

Loh and Kim [3] presented a program differencing tool named *LSDiff* (Logical Structural Diff) which is based on an algorithm that infers systematic structural changes as logic rules. The systematic changes are defined as regular changes that occur to code elements which have common structural characteristics. To detect such systematic changes, *LSDiff* represents the program as a set of predicates that describe code elements (e.g., methods and fields) and their structural dependencies (field-accesses and overriding). *LSDiff* is implemented as an Eclipse plug-in and takes as input either local workspace snapshots or revisions from SVN repositories. The tool produces a list of change rules, as well as the corresponding textual differences. It also supports the developer to understand better the changes by providing a translator that generates English language descriptions for the resulting change rules.

Apiwattanapong et al. [4] developed the *JDiff* tool for Java programs that performs differencing based on the extended graph representation of a traditional Control Flow Graph (ECFG). The new representation enables to model behaviors caused by object-oriented features. Based on this extended graph representation, *JDiff* identifies behavioral changes resulting from structural changes and relates them to the point of the code where this different behavior occurs.

Xing and Stroulia [5] presented *JDEvAn* (Java Design Evolution Analysis) tool, an Eclipse plug-in, which has the following main components: *JDEvAn core* that extracts and analyzes the design facts of a Java program; and *JDEvAn Viewer* that allows the visualization, exploration, and annotation of the comparison results in the form of change trees or UML class diagrams. The core of *JDEvAn* is the *UMLDiff* algorithm, which is a differencing algorithm designed to automatically identify structural changes between two logical models of a system. The reported changes are additions, removals, matches, moves, renames of design elements, changes to their attributes, and changes to their relationships.

A major limitation of the aforementioned tools is that they require to be installed and executed on a client machine. The installation and configuration process is a rather arduous task for some of the tools. For example, *JDEvAn* requires the installation of the PostgreSQL database server, *LDiff* requires the installation of the Perl programming language and depends on the Unix *diff* command line tool, while both *JDEvAn* and *LSDiff* require the installation of the Eclipse IDE. Furthermore, some of the tools require a significant amount of CPU and memory resources in order to run efficiently. On the contrary, WebDiff is installed on a powerful server (2 quad-core Intel Xeon Processors E5410, 16 GB RAM, 5.5 TB disk) and offers a web interface allowing to be accessed through any web browser. As a result, it does not require any kind of installation or configuration process and does not waste CPU and memory resources on client's machine.

Another limitation of the aforementioned tools (except from *LDiff*) is that they support the differencing of a single type of software artifact (i.e., source code) written in a specific programming language, namely Java. Our approach employs a generic tree differencing algorithm that can be configured to compare any kind of documents in XML format allowing the expansion of the offered differencing services to several types of software artifacts. Currently, WebDiff supports the comparison of source code and UML diagrams exported in XMI format, but the employed algorithm has been also tested on HTML documents and WSDL specifications.

III. Architecture

WebDiff is built on the client-server model allowing multiple clients to be served simultaneously. The architecture of WebDiff along with its individual components is shown in Figure 1. The core components of WebDiff are the Diff service and the JSF Web application[1]:

A. Diff Service

The diff service consists of three major sub-components, namely VTracker, XML Generator and Edit-script Parser.

VTracker is based on the Zhang-Shasha's tree-edit distance [6] algorithm, which calculates the minimum edit distance between two trees, given a cost function for different edit operations (e.g., change, deletion, and insertion). VTracker extends the Zhang-Shasha algorithm in three important ways. First, it uses an *affine-cost policy*, which adjusts the cost of each edit operation based on other related operations. In other words, the deletion/insertion cost of a node is context sensitive. For example, if all the children of a node are also candidates for deletion, this node is more likely to be deleted as well, and thus the deletion cost of that node is considered less than the regular deletion cost. The same also holds for the insertion cost. To reflect this affinity heuristic, the cost of deleting/ inserting such a node is discounted by 50 %. Second, in a post-processing step, VTracker applies a simplicity-based filter to discard the more

[1] http://hypatia.cs.ualberta.ca:8084/WebDiff/

unlikely solutions from the solution set produced by the tree-alignment phase. The first simplicity heuristic advises the algorithm to "prefer minimal paths". When there is more than one different path with the same minimum cost, the one with the least number of deletion and/or insertion operations is preferred. The second simplicity heuristic advises the algorithm to "prefer contiguous similar edit operations". Intuitively, this rule says that contiguous same-type operations could be considered as a single edit operation. When there are multiple different paths with the same minimum cost and the same number of editing operations, the one with the least number of changes (refractions) of operation types along a tree branch is preferred. The third simplicity heuristic advises the algorithm to maximize the number of sibling nodes along a tree branch to which the same edit operation is applied. Third, it allows for back cross-references between nodes of the compared trees. Essentially, VTracker considers referenced elements as being part of the referring elements during the matching process.

The VTracker core algorithm for tree-edit distance calculation is generic, so that it can be applied to any application domain, as long as the domain elements can be represented as partially ordered labeled trees. In addition, it requires a minimal configuration process in order to specify the most important attribute (i.e., *id attribute*) for each type of node present in the tree structure.

The XML Generator component is responsible for the generation of the XML files that will be compared by VTracker. It takes as input a high-level model representation of an object-oriented software system and transforms it into a set of XML documents. In order to improve the efficiency of VTracker the model is decomposed into separate XML files where each one of them represents a different kind of dependency relationship, such as membership, inheritance, association and usage. Moreover, this model decomposition allows the user to perform partial model comparison on the dependencies of interest.

The Edit-script Parser component is responsible for enriching the outcome of VTracker with domain knowledge. It takes as input the edit-scripts produced by VTracker for each kind of dependency relationship and generates a human readable report by mapping the tree edit operations to design change descriptions.

B. *JSF Web Application*

The Web application component employs the JavaServer Faces technology allowing the simple integration of rich and Ajax-enabled web-based user interface components. These components make possible to offer a desktop-like experience to the user through a web browser. The web application is responsible for performing two major tasks.

First, it validates and processes the input parameters provided by the user. The validation phase examines whether the specified URL to a version control system has a correct format and the files to be uploaded have appropriate extensions. The processing phase performs all necessary steps for the extraction of the high-level model representation that will be given as input to the Diff service. In case of source code input, the specified revision is checked-out from the repository, the code is compiled and the AST Parser component is called to generate the corresponding model. In case of UML diagram input, the XMI file is uploaded to the server (and uncompressed if necessary) and the XMI Parser component is called to generate the corresponding model. All the files that have been uploaded by the user or have been checked-out from version control systems to the server are stored in a separate directory (based on the session id) and are automatically deleted after the end of the session. The input parameters are saved in the session variables so that the user does not have to re-type some of them in order to execute another comparison.

Second, it executes and monitors the Diff service. The user is informed about the status of the processed job through a progress bar and has the ability to interrupt it at any time and start a new one with different options regarding the compared revisions or classes and the dependency relationships to be analyzed. The results of the comparison are displayed in a tree structure which is dynamically updated during the processing of the job whenever a change is reported by the Diff service. Each parent node in the tree structure represents a class of the analyzed system and the reported changes are displayed as child nodes nested under the corresponding class node.

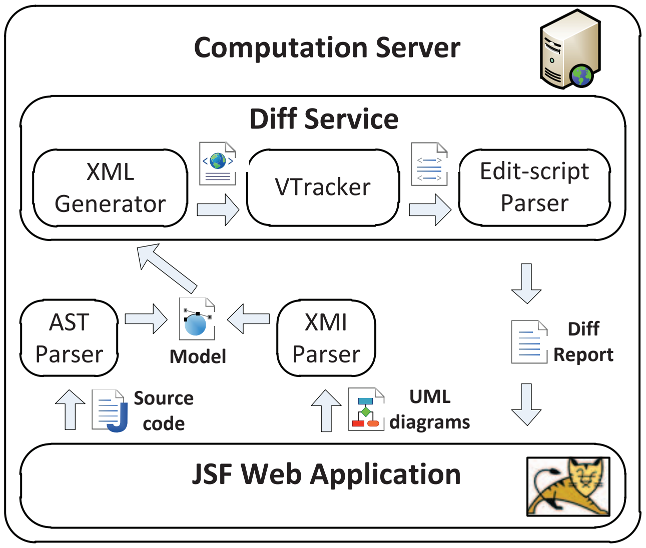

Figure 1. Architecture of WebDiff.

IV. Usage Scenario

WebDiff offers three comparison services, which mainly differ on the type of software artifacts given as input by the user. After parsing and analyzing the software artifacts, the initialization of the diff process (by selecting the classes to be compared and the dependency relationships to be examined) is exactly the same for all services from the user's perspective.

For the source code comparison service, the user navigates to the *Source code Diff* tab at the top of the web application user interface. Next, the user specifies the SVN repository URL from which the compared source code versions will be checked-out, the credentials required for the authentication to the repository and the revision numbers to be compared. By clicking on the *Checkout and compile* button the selected revisions are locally checked-out from the remote repository and their source code is compiled and analyzed. The user is informed about the status of processed jobs through a progress bar.

For the comparison of UML diagrams, the user navigates to the *Design Diff* tab and specifies two XMI files to be uploaded on the server. By clicking on the *Upload and parse* button the files are uploaded on the server and their content is parsed and analyzed. Again, the user is informed about the status of processed jobs through a progress bar.

For the comparison of source code with UML diagrams, the user navigates to the *Source vs Design Diff* tab and specifies a combination of the input required for the two aforementioned services (i.e., a source code revision to be checked-out from a remote SVN repository and an XMI file to be uploaded on the server). The analysis of the software artifacts is performed by pressing the *Extract models* button.

After the analysis of the software artifacts and the extraction of the models, the user can perform two types of comparison. By navigating to the *Common Classes* sub-tab, the user can select from an item list any subset of the classes being common in both models and perform a comparison on them. Furthermore, the user can control the dependency relationships to be examined by enabling/disabling the corresponding check boxes. By pressing the *Compare selected classes* button the diff process starts and the user is informed about the number of the classes that have been compared through a progress bar. The diff process can be interrupted at any time by clicking on the *Stop* button. The other comparison feature can be used to find classes that were possibly renamed between the two examined model versions. By navigating to the *Removed/Added Classes* sub-tab, the user can select from two item lists a subset of the "removed" classes (i.e., classes being present in the first model, but not in the second one) to be compared against a subset of the "added" classes (i.e., classes being present in the second model, but not in the first one). By pressing the *Compare selected pairs of classes* button all combinations between the selected removed/added classes are examined in order to find similarities within their contents (i.e., existence of the same attributes and operations) indicating the renaming of a class.

V. Conclusion

In this paper we presented WebDiff, a web-based generic differencing service that supports the comparison of various types of software artifacts. WebDiff employs a domain-independent tree differencing algorithm (VTracker), which is able to handle any kind of XML document representing a partially-ordered labeled tree. As a result, our diff engine does not depend on specific types of artifacts or programming languages and can be easily extended to support any software-related document that can be represented as an ordered labeled tree. Furthermore, WebDiff is built on the client-server model allowing multiple clients to be served simultaneously, or even the execution of multiple differencing processes by the same user. In addition, our approach does not require any software installation or configuration on the client-side, since the service is accessible through any web browser. Finally, the client-server model enables the transfer of the computation cost to the server and minimizes the waste of resources on the client-side.

We are planning to extend WebDiff in four main ways. First, we plan to support the analysis of more programming languages. This can be achieved by integrating additional AST parser components for the languages of interest without modifying the current Diff service infrastructure. Second, we plan to support more version control systems from which source code can be checked-out (e.g., the IBM Jazz source control and Git SCM). Third, we plan to support more versions of the XMI standard by extending the corresponding parser component (currently XMI version 1.2 is supported for UML 1.4) and enable the comparison of diagrams specified in different UML versions. Finally, we plan to expand our Diff service on more types of UML diagrams and software artifacts by developing appropriate high-level model representations.

Acknowledgment

The authors would like to acknowledge the generous support of NSERC, iCORE, and IBM.

References

[1] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau, "An Empirical Study on Web Service Evolution," 9th IEEE International Conference on Web Services (ICWS'2011), Washington DC, USA, July 4-9, 2011.

[2] G. Canfora, L. Cerulo, and M. Di Penta, "Ldiff: An enhanced line differencing tool," 31st ACM/IEEE International Conference on Software Engineering (ICSE'09), pp. 595-598, Vancouver, BC, May 16-24, 2009.

[3] A. Loh, and M. Kim, "LSdiff: a program differencing tool to identify systematic structural differences," 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10), vol. 2, pp. 263-266, Cape Town, South Africa, May 2-8, 2010.

[4] T. Apiwattanapong, A. Orso, and M. J. Harrold, "JDiff: A differencing technique and tool for object-oriented programs," Automated Software Engineering, vol. 14, no. 1, pp. 3-36, March 2007.

[5] Z. Xing, and E. Stroulia, "API-Evolution Support with Diff-CatchUp," IEEE Transactions on Software Engineering, vol. 33, no. 12, pp.818-836, December 2007.

[6] K. Zhang, and D. Shasha, "Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems," SIAM Journal of Computing, vol. 18, pp. 1245-1262, December 1989.