Analysis and Modeling of Evolving Database-centric Web Applications

S. V. Madhava Krishna[+] Satyadeep Karnati[+]
IIT, Guwahati
Assam, India
{sista, karnati}@iitg.ernet.in

Abhishek Biswas
Old Dominion University
Norfolk, Virginia, USA
abiswas@cs.odu.edu

Jagannathan Srinivasan
Oracle Corporation
Nashua, NH, USA
jagannathan.srinivasan@oracle.com

Abstract

Database-centric web applications tend to evolve over time. However, there are no comprehensive tools to analyze and present the synopsis of changes for such applications. In this paper, we address the problem of analyzing an evolving application and presenting the synopsis of changes, which can be recursively drilled down in an interactive manner. Specifically, we analyze two versions of an application, each constituting of a hierarchy of pages, page regions, and region items, and model the synopsis of changes. In addition to analyzing the content of pages, our synopsis generation algorithm takes into account, the changes resulting from page layouts, page branching transitions, and page schema dependencies. Furthermore, the region pair-wise similarity is extended to show *m : n* evolution as well, which is common due to clone and edit operations typically employed during development. We have developed region similarity measures to aid the analysis and a bottom-up approach is used to label the regions and the container pages. We have used this approach to implement an Evolving Application Synopsis Tool (EAST), which can analyze database-centric web applications built using Oracle Application Express Tool. An experimental study done with four deployed applications and one beta version of application demonstrate the usefulness of our approach.

1. Introduction

A majority of today's web applications are database-centric. This can partly be attributed to maturity and robustness and scalability of RDBMS [1] and partly to the availability of free and/or open source rapid application development tools [2]. The latter has also allowed adoption of *Agile* software development methodology for development, where requirements and solutions tend to *evolve* over a period of time. A key challenge with such evolving applications is tracking changes from release to release, which are occurring at much shorter time intervals.

Although it is desirable to keep track of changes between software releases, it is not done especially for web applications, where application code and logic is dispersed behind various page items, event handlers, and page processes. These web applications are typically developed using a rapid application development tool such as Oracle Application Express [3].

Rapid application development tools aid in agile software development but makes the task of tracking changes difficult. This can be primarily attributed to the following:

- The link between pages and its code components are internally managed by the tool.
- For installation and maintenance purposes, the entire code (application dump) is available as a single monolithic file as opposed to at finer granularity.
- The tools typically do not support versioning especially at application component level.

One can hypothetically compare two versions of application dumps by using a traditional source code `'diff'` utility. However, the obtained diff is not coherent as the application dump is a mashed up version of the code supplied by developer, along with code automatically generated by the rapid application development tool. This problem is further compounded by the dependency on database schema objects and stored procedures.

Thus, the ability to automatically generate the synopsis of changes across versions of database-centric web applications would be very useful, which is the focus of this paper. Specifically, we address the problem of analyzing two versions of a database-centric web application and automatically generating the synopsis of changes.

The basic approach is as follows: We view each version of the application as a structured hierarchy of *web pages*, *page regions*, and *region items*. We establish page

16th International Conference on Management of Data
COMAD 2010, Nagpur, India, December 8--10, 2010
©Computer Society of India, 2010

[+]This work was done as part of a summer internship at Sarada Research Labs, Bangalore.

equivalence by name (in our case page identifiers) and hence can initially derive the status of *deleted*, *inserted*, and *identical* pages by comparing page ids in two versions (see Section 5 for the ramifications of this choice). Next, we perform pair-wise comparison of pages marked identical between the two applications in a bottom up manner, namely, detecting changes at item level, next page region level, and finally in container pages and appropriately labelling the corresponding component (as *changed*) if diff is found. For string matching, we make use of edit distance function [10] and for source code matching we use java library from [12].

The two labelled page branching trees are presented side-by-side thus succinctly depicting changes in page contents as well changes in page transitions.

We have developed *region similarity measures* to aid the analysis. Our similarity measure handles both differences arises due to changes in layout, types of region items, as well as behavioural changes present in underlying source code. In addition, our scheme is able to capture *m:n* evolution of a region that typically can occur if developer uses *clone and edit* operation to create multiple regions from a single source region.

We also augment the above page content change analysis with schema dependencies changes.

We are able to present the page content analysis changes by taking into account layout of the regions within the page. This information is derived by consulting the page template used for rendering the page. We allow synchronized browsing across the side-by-side page view, so user can easily track the modified pages between the two versions of the application.

User can recursively drill down from page branching tree to view diffs at page level, region level, and item level. Additional labels (identical, changed) are associated at each level to capture corresponding schema dependencies changes, which can also be examined, if desired.

Using this approach, we have built an *Evolving Application Synopsis Tool* (EAST), which is yet another database-centric web application developed with Oracle Application Express (APEX) Tool [3]. A key aspect of APEX is that it maintains the application metadata also in Oracle Database, which is made available as a collection of views. This allowed us to analyze the applications easily.

An experimental study conducted with four deployed applications and one beta version of application of Sarada Research Labs, Bangalore at various Ramakrishna Missions demonstrate the usefulness of our approach.

The key contributions of the paper are:

- To the best of our knowledge, this is first attempt to automatically analyse and model synopsis of evolving database-centric web applications.
- The region similarity measures, and the overall page change analysis algorithm, and
- The EAST tool and its use in studying evolving applications that demonstrates the usefulness of our approach.

1.2 Related Work

The text based file comparators, popularly known as *diff* became first available as part of UNIX system (the first implementation was based on [4]) and has been around since 1975. Several options are supported, including *normal* (with lines marked with 'a' added, 'd' deleted and 'c' changed), *context* (provided by including additional unchanged lines), and *unified* formats (compact version of context format), as well as generating *edit script* (which can convert old file to new). The *diff* utility has also been extended to work on binary files. We do rely on diff utility to do basic source SQL and PL/SQL code comparisons.

Work has also being done to compare two versions of a program by considering programming language syntax [6] as well as by capturing semantic changes [5].

However, our work is more similar to finding structural changes as reported in [8] and detecting changes in XML document [9,11], both of which address the issue of dealing with hierarchically structured data. We employ a simpler algorithm that exploits the domain knowledge known in our case about evolving web pages (See Section 2 for more details).

In the database world, the versions of database schema objects have been compared to understand schema evolution [7]. However, for us, in addition to schema object evolution, we also need to track changes resulting from differences in schema dependencies between versions of the application at varying level of granularity (pages, regions, and items).

1.3 Organization of rest of the paper

Section 2 gives the key concepts pertaining to analysis and modelling of evolving application synopsis. Section 3 gives an overview of building the EAST tool. Section 4 describes the experimental study conducted with deployed applications and a beta version of application. Section 5 contains discussion and Section 6 concludes the paper and outlines future work.

2. Key Concepts

This section presents the key concepts of analysis and modelling of evolving application synopsis.

2.1 Overview

Database-centric web application development tools typically use the Model-View-Controller (MVC) architecture [13] as the basic development model. Thus, we follow a similar architecture to analyse changes in evolving applications and model the synopsis. We analyse the application differences along two hierarchies corresponding to the view and model components of the MVC architecture:

- *View Hierarchy*: This hierarchy of pages, page-regions and region-items, models the user interface. Under this hierarchy, we generally look at the page layout, page navigation, reports, and forms, etc.
- *Model Hierarchy*: This hierarchy of pages, page-events, event-processes and process schema dependency, models the backend code triggered by user interaction. This component analyses changes in application code encoding business rules and their schema dependency. The controller is an inherent part of this hierarchy and is not modelled separately.

These two change hierarchies (see Figure 1) are analysed in a bottom-up order. However, they are presented in a top down order with ability to drill down recursively. The above two hierarchies are, in general, present in any web application.

In addition, for a database centric application, we can analyse the application evolution from a database centric dimension. This can be modelled as an inverse hierarchy of schema objects and dependent page components. This schema dependency evolution hierarchy is useful in tracking changes in schema objects and in turn their dependents (Figure 2).

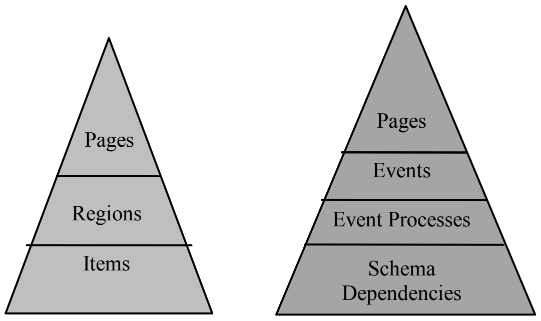


Figure 1: View & Model Hierarchy (in MVC-context)

The above three hierarchies is discussed in the following sections. Although, the discussion is presented in context of applications developed used APEX, the concepts are applicable to database-centric web applications, in general, unless otherwise mentioned.

2.2 View Hierarchy Analysis

The first task in analysis of an evolving application is to establish page equivalence between two versions of the application. Page equivalence can be established using heuristic techniques, however, we found that page *equivalence by name* (identifiers in our case), suffices for applications generated by APEX. During page equivalence generation, we also derive list on inserted or deleted pages. Pages common to both versions of the application are analysed further to deduce if any changes have been made.

It is a common practice to split a HTML page into regions or frames using different multi-view constructs. Region equivalence cannot be derived directly as page equivalence by name, as two different regions may have same name, or the regions may not have been named at all. Moreover, as explained earlier, *m:n* similarity of the regions is possible. In order to tackle these challenges, a *region similarity measure* Φ is introduced to compare certain properties of the regions.

Region Similarity Measure & Evolution: In this section we derive the region similarity measure and discuss the algorithm to mark modified regions. A region can be considered as a container containing components from a predefined set of HTML and APEX controls, ordered by, their sequence identification numbers. The similarity measure identifies attributes of the regions not expected to change significantly and applies the similarity between these attributes to recognize evolved regions.
The similarity measure is defined as:

$$\phi(r_1, r_2) = \begin{cases} 0, if \quad r_1 \neq r_2 \\ 1, if \quad r_1 = r_2 \\ (0,1), if \quad partial \quad match \end{cases}$$

where, r_1 and r_2 are regions being compared. The comparison criterion based on the similarity measure Φ(r_1,r_2) is defined by the Boolean function:

$$C(r_1, r_2) = \begin{cases} 1, if \quad \phi(r_1, r_2) \geq T : T \in (0,1], \phi \in [0,1] \\ 0, if \quad \phi(r_1, r_2) \leq T : T \in (0,1], \phi \in [0,1] \end{cases}$$

where, T is the threshold value set by experimentation. A higher threshold results in increased number of mismatches. On the other hand, a lower threshold fails to find out good matches. To calculate the similarity measure, we take into account four different similarity scores as discussed below:

- *Region Type Score*: If the region type is different, a score of 0 is returned. In case of custom region types if only one such region is allowed then a score of 1 is returned if exactly matched.

$$\phi_{RgType} = \begin{cases} r_1.type = r_2.type \quad then \quad 1 \\ r_1.type \neq r_2.type \quad then \quad 0 \end{cases}$$

- *Region Name Score*: An edit distance similarity function is applied to match the region names and converted to a value between 0 and 1. The score is calculated as

$$\phi_{RgName} = \max\left(1 - \frac{e(r_1.name, r_2.name)}{T_n + 1}, 0\right)$$

where *e(text, text)* is the edit distance function and T_n is the maximum edit distance acceptable. If *e(r1.name, t2.name)* << $T_{n,}$ the score is ~1 indicating a good match and if *e(r1.name, t2.name)* > T_n the max function returns 0 indicating no match.

- *Region Item Counts Score*: Similar regions are expected to have a large number of common items. So, this score is based on the count of the common, inserted and deleted items in the two regions under comparison. First we establish one to one correspondence among the items on the page. Page items like textboxes, select lists and calendar controls have unique names in a page for referencing and dereferencing purpose as they are used during submitting and requesting a page. Hence we establish equivalence of items based on their name to find the common, inserted and deleted items and group them by their container regions.

  Let *a* be the total number of items in the first region and *b* be the total number of items the second region. Also, let *c* be the number of common items in the two regions. Then, the total number change is given by:

$$TotalChanges = (a-c) + (b-c) = a + b - 2c$$

  The total number of changes divided by total number of items is the fraction of change and the similarity score bases on item count is given by:

$$1 - \left(\frac{a+b-2c}{a+b}\right) = \frac{2c}{a+b}$$

$$\phi_{RgItemCount} = \begin{cases} 0 \quad if \quad a=b=c=0 \\ \dfrac{2c}{a+b} \quad otherwise \end{cases}$$

  Certain regions are report regions which only display data in a table. The score for such regions is calculated using the table columns instead of the items.

- *Region Source Score*: The region source diff is calculated by using [12] which gives the characters to be added or deleted to convert one text to other. The diff score can be normalized by calculating the ratio of the common characters to the total characters in the following manner:

$$\phi_{RgSrcDiff} = \begin{cases} 0 \qquad if \quad \#text_1 + \#text_2 = 0 \\ 1 - \dfrac{\#deleted + \#added}{\#text_1 + \#text_2} \quad otherwise \end{cases}$$

The final similarity measure between two regions is calculated using a weighted average. Let w_1, w_2, w_3 and w_4 be the weights assigned to the four similarity scores respectively where,

$$w_3 = \begin{cases} 0 \quad if \quad a=b=c=0 \\ w_3 > 0 \qquad otherwise \end{cases}$$

$$w_4 = \begin{cases} 0 \quad if \quad \#text_1 + \#text_2 = 0 \\ w_4 > 0 \qquad otherwise \end{cases}$$

Then the similarity measure is computed as:

$$\phi(r_1, r_2) = \frac{w_1\phi_{RgType} + w_2\phi_{RgName} + w_3\phi_{RgItemCount} + w_4\phi_{RgSrcDiff}}{w1 + w2 + w3 + w4}$$

Using the similarity measure and comparison criterion $C(r_1, r_2)$, pair wise equivalence can be established between the regions of the pages from the two applications. A table with page number, regions identifier and the similarity measure is populated. Regions which fail to match are marked deleted if they belong to the older application, otherwise, marked inserted. Matched regions are marked modified if the similarity measure is less than 1 as changes have been detected in the two regions during the basic similarity score calculation. In case of exact matches, the regions attributes like display order of the items in the region, display position of the region on the page, the region source and the display conditions are compared further to obtain the modification status. The algorithm for populating the region modification table is outlined below:

Algorithm: Populate Region Comparison

```
Input: Region Page Numbers $P_{old}$ and $P_{new}$
        Threshold: T
Output: Region Similarity & Modification Table ($P_{old}$.Region, $P_{new}$.Region, $\Phi$, Status)
Algorithm:
 1: for a each Region in $P_{old}$ do
 2:     matchRegionFound = FALSE;
 3:     for a each Region in $P_{new}$ do
 4:         score = $\Phi$ ($P_{old}$.Region, $P_{new}$.Region)
 5:         if(score > T) then //Regions similar
 6:             matchRegionFound = TRUE;
 7:             if($\Phi$ ==1) then //Regions match exactly
 8:                 if(compare_full($P_{old}$.Region,$P_{new}$.Region)) then
                      //checking other properties
 9:                     Status = "same"
10:                 else
11:                     Status = "modified"
12:                     Update status of page as "modified"
13:                 end if
14:             else //declared as partial match
15:                 Status = "modified"
16:                 Update status of page as "modified"
17:             end if
18:             Insert ($P_{old}$.Region,$P_{new}$.Region, $\Phi$, Status)
19:         end if
20:     end for
21:     if (matchRegionFound == FALSE)
22:         Insert ($P_{old}$.Region,NULL, 0, 'deleted')
23:     end if
24: end for
25: for a region in $P_{new}$ do
26:     if($P_{new}$.Region NOT IN RegSimilarityTable. $Reg_{new}$)
27:         Insert (NULL, $P_{new}$.Region, 0, 'inserted')
28: end for
```

It is important to note that a region can be matched with multiple regions in the other application due to clone and edit operations by developers. Small form regions often fall into this category. They can be copied and used multiple times with small changes. Such regions are difficult to match with a one to one correspondence. So, they are matched with multiple regions and correct match can be entered by a human reviewer.

Once the regions similarity is established, we move on to item similarity. Modification status of the matched items is computed by comparing properties like label, display sequence number, and display conditions. The modification status of the container region and parent page is simultaneously updated for modified items in bottom up order.

2.3 Model Hierarchy Analysis

During a web page rendering process back-end application code can be executed during a page load i.e. before the final HTML file is sent to the client for rendering or during a post i.e. a client generates a request by some event. APEX provides a set of events, to which PL/SQL handler routines can be attached, to process user requests and encode business rules. Providing a set of predefined events is a standard practice in event based programming and followed by most rapid web development tools. As shown in figure 1, the predefined events and the handler processes form the two levels of the model hierarchy under each page. The schema dependency level computes [14] and stores the schema dependencies of the handler processes.

Since, page equivalence has been established earlier, here, we compare the handler processes attached to each event and derive the matched, inserted and deleted processes based on the process name. Matched processes are further compared by a source code diff algorithm, implemented by [12], to generate the modification status. The results are entered in a table (P_{old}, P_{new}, EventType, P_{old}.Event.Process, P_{new}.Event.Process, Status). Inserted and deleted processes are also recorded in this table with P_{new} or P_{old} set as null respectively.

APEX also allows us to attach some SQL or PL/SQL code to a HTML item as a special source attribute for rendering the initial value. These code snippets are not attached to a specific event and therefore are compared under the view hierarchy as an item attribute.

2.4 Schema Dependancy Evolution Hierarchy Analysis

The schema dependency evolution hierarchy (Figure 2) tracks the change in the schema dependency of application components between different versions.

This hierarchy is built in a bottom up order with page and page component equivalence established by matching the page identifiers and page component names respectively. The schema dependencies of the page components like processes, region source, item source etc. are computed by the process in [14] and stored in table (SchObjectId, AppVersion, PageId, ComponentId).

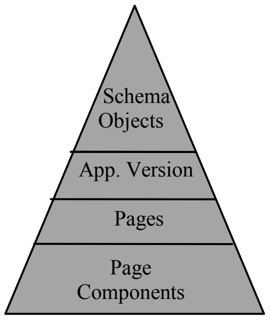


Figure 2: Schema Dependency Evolution Hierarchy

The list of referenced schema objects for each component is queried from the table above. For each pair of matched components, in the two applications, the schema reference lists are compared to derive the schema dependency change at the component level. Similarly, the list of referenced schema objects for each page is derived by grouping the schema objects dependencies of all the components on the page. Then, for each pair of matched pages the schema object dependency list is compared to derive the schema dependency change at the page level. Finally, two application wide schema dependency lists are generated and compared to compute the change in schema dependency at the application level.

To simplify implementation, we ignore the dependency type change i.e. read or write. So, in the above analysis, a schema object is either referenced or not referenced by a component and the modification status can only be inserted or deleted. If we consider dependency type we will need to add a third category capturing the possible change in the type of dependency. Then for each schema object we will have three possible modification status at each level, i.e. inserted, deleted and modified.

In addition to change in references to schema objects, the referenced schema objects can themselves evolve. In [17], authors discuss the generic schema matching problem and presents algorithms for establishing schema object equivalence between two independently developed schemas. These algorithms and measures could be used for matching renamed schema objects (if any) that occur between two versions of the application. For most practical purposes, a database specific tool, like Oracle Schema Diff (part of Oracle SQL Developer [18]) can be used to identify schema objects that evolved between the two versions of the database schema. The output from the database diff tool can be used to accordingly update the schema dependency evolution hierarchy.

2.5 Visual Modelling of Evolution Synopsis

The application diff computed above is visually rendered as described below.

Application Level Diff: At the root level each application is depicted as a summary tree [14] with the pages as nodes and the navigation between the pages as tree edges. The two applications page-branching trees are shown side-by-side for visual inspection. The inserted, deleted, and modified pages are identified by different color codes. Clicking on a page highlights the corresponding pages on the other application and hence the changes in the page branching can be detected easily.

Page Level Diff: By selecting a modified page from the summary tree, one can drill down to the page level diff, which shows the page in the two versions side by side displaying the regions in their respective display positions. The theme and template of the page from the APEX views are used to place each region in its position thus allowing us to depict the page layout changes.

Inserted, deleted and modified regions are displayed in different colors and selecting a region highlights the corresponding matched region in the other version. Regions with multiple similarities, i.e., regions having more than one matched regions crossing the threshold are highlighted in a different color on selection and the correct match can be manually selected from the list of matched regions. Regions can be selected from either versions of the application and a list of matching regions from the other version is shown for manual selection.

Region Diff: Page diff allows a drill down to region diff showing the changes in the two regions due to region source, report columns and region items.

All the components of the region like columns, items and buttons are shown using the region template. The items in the region are color coded just as regions in the page diff. Also, as a score based scheme is used for pair-wise matching of regions, we could have multiple region similarity scores crossing the match threshold. Hence, many to many matching may occur among the similar regions of the two versions. Such close matches occur due to copy and paste of regions in a page with small changes.

Item Diff: This is the lowest level of the diff in the view hierarchy, which compares the important properties of an item in a tabular form and also displays the item source diff.

Process Diff: The page level diff visualization also shows the model hierarchy. The specific events and the attached handler processes are shown in a tabular format identifying the inserted, deleted and modified by colour codes. Clicking on a process shows the diff of the various process properties along with the process source code diff.

Schema Dependency Diff: The schema dependency reports are provided at each level in the pages, page-region and region-item hierarchy following the same drill down traversal as the view hierarchy. The page schema dependency changes are also shown in the page branching summary tree used for application diff. Special colour code markers are used to signify schema dependency change of a page. The schema dependency for the application on the whole can also be viewed. This gives us the list of schema objects used by both the applications and those which have been added or deleted.

2.6 Time Complexity Analysis

At each stage during the analysis of the application diff along any of the three hierarchies, we have to perform a two step process.

First step is to establish equivalence between the application components based upon a unique identifier or a similarity measure. Matching based upon unique identifier or a name is *O(nlogn)* operation, whereas matching based upon similarity measure like the one used in region equivalence computation in section 2.2 is $O(n^2)$ operation, where n is the number of components.

The second step is to further categorize the similar items found in the first step as same or modified, which can each be done in constant time. Thus, the overall complexity of this step depending on the how equivalence is established is *O(nlogn)* or $O(n^2)$.

3. Evolving Application Synopsis Tool

This section describes the EAST Evolving Application Synopsis Tool that was built using the proposed scheme.

3.1 Overview

The EAST application and the applications being analyzed are developed using Oracle APEX [3], Version 3.2, a Rapid Application Development tool with Oracle Database 10g Express Edition[15] as the database.

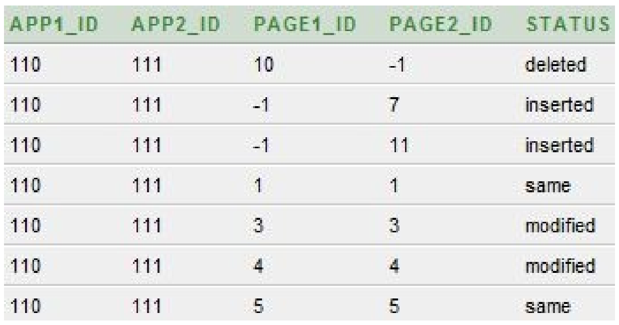
APEX maintains the metadata of the whole web-application in a database schema and provides a set of application views containing information about almost all aspects of the application. We have used tables to store the results of the application comparison at various levels which are discussed in the Section 3.2.

The procedural code for populating these tables is written in Oracle's PL/SQL Database Programming Language [16]. The similarity measures (Section 2.2) are implemented as PL/SQL functions. Once all the tables are populated during analysis, the visual modelling of the Diff is rendered by using the value of the modification status of the components from the corresponding tables.

3.2 Database Schema

In order to show the summary tree mentioned in Section 2.5, we store the following information in the *Page Transitions* [14] table organized as the ids of the pages in an application along with the parent page ids. To present the Page Diff of Section 2.5, we store the page id's of the two compared application along with their modification status in the *Page Diff* table (Figure 4).

The *Region Diff* table stores the Region identifiers of the regions contained in the pages of the *Page Diff* table, along with their status. Similar is the case for the *Process Diff* and *Item Diff* tables.



APP1_ID	APP2_ID	PAGE1_ID	PAGE2_ID	STATUS
110	111	10	-1	deleted
110	111	-1	7	inserted
110	111	-1	11	inserted
110	111	1	1	same
110	111	3	3	modified
110	111	4	4	modified
110	111	5	5	same

Figure 4: Portion of the PAGE DIFF Table

In order to show the Schema Dependency Diff (Section 2.5) we use the *Dependent Component* table and the *Referenced Object* table. The *Dependent Component* table stores all the components *viz.*: regions, items, process along with their page ids and a unique id (dependent id) assigned to each component by EAST which is the primary key. The *Referenced Object* table stores the ids of the schema objects which are referenced by each component in the *Dependent Component* table with dependent id as the foreign key. The relation between various tables is shown in Figure 5.

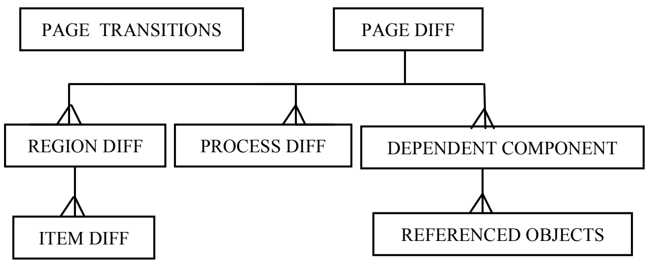
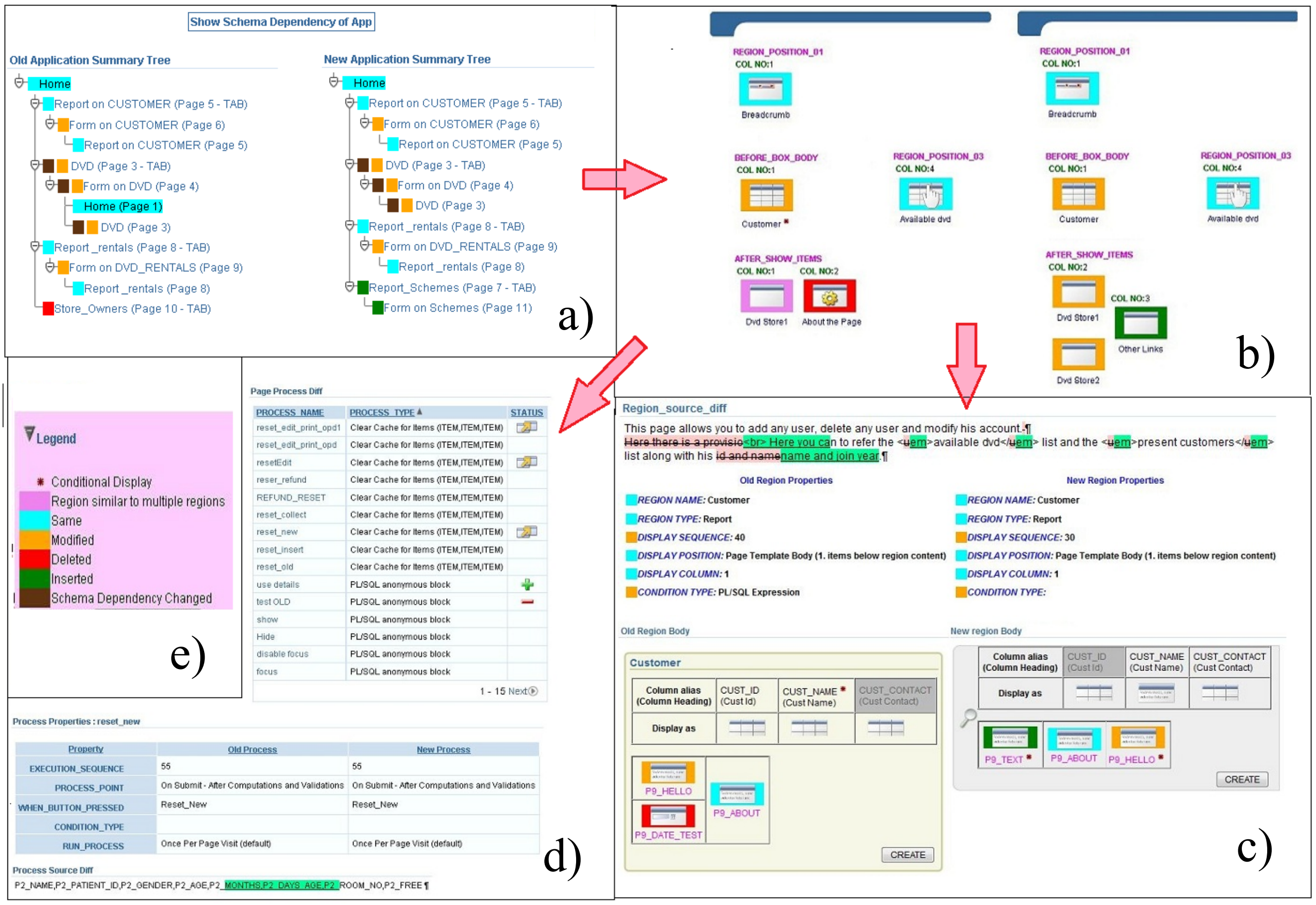


Figure 5: Database Schema

3.3 Change Detection and Presentation

In this Section, we present the screenshots of the EAST application analyzed for an application. It depicts all the aspects described in Section 2.5.

Figure 3-a presents the Application Summary trees of the two versions of an analyzed application side- by-side along with the color codes showing inserted, deleted,



Show Schema Dependency of App

Old Application Summary Tree

- 🟦 Home
  - 🟦 Report on CUSTOMER (Page 5 - TAB)
    - 🟧 Form on CUSTOMER (Page 6)
      - 🟦 Report on CUSTOMER (Page 5)
  - 🟫 🟧 DVD (Page 3 - TAB)
    - 🟫 🟧 Form on DVD (Page 4)
      - 🟦 Home (Page 1)
      - 🟫 🟧 DVD (Page 3)
  - 🟦 Report _rentals (Page 8 - TAB)
    - 🟧 Form on DVD_RENTALS (Page 9)
      - 🟦 Report _rentals (Page 8)
  - 🟥 Store_Owners (Page 10 - TAB)

New Application Summary Tree

- 🟦 Home
  - 🟦 Report on CUSTOMER (Page 5 - TAB)
    - 🟧 Form on CUSTOMER (Page 6)
      - 🟦 Report on CUSTOMER (Page 5)
  - 🟫 🟧 DVD (Page 3 - TAB)
    - 🟫 🟧 Form on DVD (Page 4)
      - 🟫 🟧 DVD (Page 3)
  - 🟦 Report _rentals (Page 8 - TAB)
    - 🟧 Form on DVD_RENTALS (Page 9)
      - 🟦 Report _rentals (Page 8)
  - 🟩 Report_Schemes (Page 7 - TAB)
    - 🟩 Form on Schemes (Page 11)

a)

b)

Region_source_diff

This page allows you to add any user, delete any user and modify his account.¶
~~Here there is a provisio~~
 Here you can to refer the <~~u~~em>available dvd</~~u~~em> list and the <~~u~~em>present customers</~~u~~em> list along with his ~~id and name~~name and join year.¶

Old Region Properties

🟦 *REGION NAME:* Customer
🟦 *REGION TYPE:* Report
🟧 *DISPLAY SEQUENCE:* 40
🟦 *DISPLAY POSITION:* Page Template Body (1. items below region content)
🟦 *DISPLAY COLUMN:* 1
🟧 *CONDITION TYPE:* PL/SQL Expression

New Region Properties

🟦 *REGION NAME:* Customer
🟦 *REGION TYPE:* Report
🟧 *DISPLAY SEQUENCE:* 30
🟦 *DISPLAY POSITION:* Page Template Body (1. items below region content)
🟦 *DISPLAY COLUMN:* 1
🟧 *CONDITION TYPE:*

Old Region Body

Customer

Column alias (Column Heading)	CUST_ID (Cust Id)	CUST_NAME (Cust Name)	CUST_CONTACT (Cust Contact)
Display as			

New region Body

Column alias (Column Heading)	CUST_ID (Cust Id)	CUST_NAME (Cust Name)	CUST_CONTACT (Cust Contact)
Display as			

c)

Page Process Diff

PROCESS_NAME	PROCESS_TYPE ▲	STATUS
reset_edit_print_opd1	Clear Cache for Items (ITEM,ITEM,ITEM)	✎
reset_edit_print_opd	Clear Cache for Items (ITEM,ITEM,ITEM)	
resetEdit	Clear Cache for Items (ITEM,ITEM,ITEM)	✎
reser_refund	Clear Cache for Items (ITEM,ITEM,ITEM)	
REFUND_RESET	Clear Cache for Items (ITEM,ITEM,ITEM)	
reset_collect	Clear Cache for Items (ITEM,ITEM,ITEM)	
reset_new	Clear Cache for Items (ITEM,ITEM,ITEM)	✎
reset_insert	Clear Cache for Items (ITEM,ITEM,ITEM)	
reset_old	Clear Cache for Items (ITEM,ITEM,ITEM)	
use details	PL/SQL anonymous block	+
test OLD	PL/SQL anonymous block	−
show	PL/SQL anonymous block	
Hide	PL/SQL anonymous block	
disable focus	PL/SQL anonymous block	
focus	PL/SQL anonymous block	

1 - 15 Next

Process Properties : reset_new

Property	Old Process	New Process
EXECUTION_SEQUENCE	55	55
PROCESS_POINT	On Submit - After Computations and Validations	On Submit - After Computations and Validations
WHEN_BUTTON_PRESSED	Reset_New	Reset_New
CONDITION_TYPE		
RUN_PROCESS	Once Per Page Visit (default)	Once Per Page Visit (default)

Process Source Diff

P2_NAME,P2_PATIENT_ID,P2_GENDER,P2_AGE,P2_MONTHS,P2_DAYS_AGE,P2_ROOM_NO,P2_FREE¶

d)

Legend

✱	Conditional Display
🟪	Region similar to multiple regions
🟦	Same
🟧	Modified
🟥	Deleted
🟩	Inserted
🟫	Schema Dependency Changed

e)

Figure 3: a) Application Level Diff b) Page level Diff c) Region Level Diff d) Process Level Diff e) Legend

same and modified pages according to the Legend shown in Figure 3-e. For example, the page *'Form on Customer'* is labeled in orange to indicate it has been modified.

Similarly, the *'DVD page'* is labeled in orange (to indicate change in content) as well as brown (to indicate that change in its dependency on schema objects). We also know that the *'Store ow*ners' page is deleted (labeled in red in left side branching tree corresponding to old version of the application), whereas *'Reports Scheme'* page has been added (labeled in green in right side branching tree corresponding to new version of the application). Note that showing a side-by-side diff allows for synchronized browsing, that is, clicking on a page in one, highlights its occurrence in other. Also, the differences in branching transitions are easy to examine.

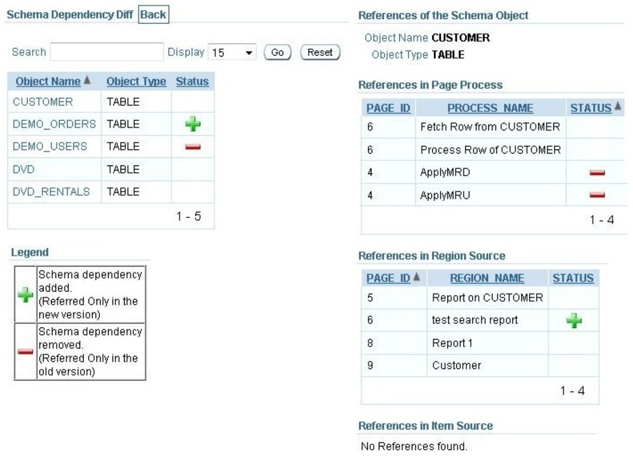
The change in the Schema dependency between two applications can be examined by clicking on *Show Schema Dependency for Application*. One can drill down to see the Page level diff of any page by selecting the corresponding page here.

Next, we examine the page level diff for '*Report on Customer'* page (shown in Figure 3-b). The two versions of the pages are shown consisting of page-regions. The same color code is used to label the regions to indicate the status of deleted, inserted, same, and modified. Here one can notice the template of the page and the regions placed in their respective positions. Also, the icons capture the type of region (e.g., a breadcrumb, columnar report, etc.).

An interesting aspect is the labeling of region with the purple color to indicate it matches with multiple regions. When such a region is selected, all similar regions occurring in other page is highlighted. User can then pick up one of the similar regions to drill down further.

Figure 3-c shows the result of performing region diff on *'Customer Report'* region, which is known to be modified (by the associated orange label). Here we show the properties diff, content diff, layout diff, and source diff. The region properties are presented side by side for easy comparison. We can see that the display point of the region has changed in the two pages. The content diff is easily seen, namely, the difference in the columns appearing in the report. The column shaded in grey color indicates it is hidden. In previous version, the CUST_CONTACT column was hidden, whereas in the newer version the CUST_ID column is hidden. We can also see the layout changes. Furthermore, user can select an item and corresponding item in other region is automatically highlighted. Also, note that we are able to capture the aspect that an item or report column is conditionally displayed by labeling it with '*'. Finally, the region source is shown as a single source with marked inserted and deleted portions.

Figure 3-d shows the process diff presented in a single tabular form. Here the signs '+' and '–' are used to indicate addition and deletion of processes. The modified process is shown with a 'modified' icon which on clicking shows the differences of process properties like event point, execution sequence, process source etc. For example, 3-d shows the changes for *reset_new* process. As can be seen from the Figure the source for the process is modified highlighted in green.



Object Name	Object Type	Status
CUSTOMER	TABLE	
DEMO_ORDERS	TABLE	➕
DEMO_USERS	TABLE	➖
DVD	TABLE	
DVD_RENTALS	TABLE	

1 - 5

Legend

➕	Schema dependency added. (Referred Only in the new version)
➖	Schema dependency removed. (Referred Only in the old version)

References of the Schema Object

Object Name **CUSTOMER**
Object Type **TABLE**

References in Page Process

PAGE ID	PROCESS NAME	STATUS
6	Fetch Row from CUSTOMER	
6	Process Row of CUSTOMER	
4	ApplyMRD	➖
4	ApplyMRU	➖

1 - 4

References in Region Source

PAGE ID	REGION NAME	STATUS
5	Report on CUSTOMER	
6	test search report	➕
8	Report 1	
9	Customer	

1 - 4

References in Item Source

No References found.

Figure 6: Schema Dependency Diff

The visualization of schema Diff is similar to process diff which is shown in Figure 6. Clicking on the schema object in the left table shows the list of all application components referencing the object in the right table. Here, the symbols '+' and '–' indicates that the reference is added and deleted respectively.

4. EAST Experimental Study

This section presents the experimental study used to evaluate the EAST tool. It was conducted on a Intel® Pentium® Processor E5400 (2.70GHz), 2GB RAM, Ubuntu OS 9.04, using Oracle Database 10g Express Edition, and APEX generated database applications. In addition, a usability study was conducted with EAST, where developers found it useful for analyzing changes across versions of a web application [21]. However, due to space constraints, the results of that study are omitted.

4.1 Applications Used for Analysis

Five deployed applications of Sarada Research Labs, Bangalore were used for analysis as listed in Table 1.

Table 1: Application Evolution Characteristics

	Version Diff (in months)	*App. Maturity (O,N)	Pages (O,N)	Regions (O,N)	Items (O,N)
TBReg.	0.75	(60,85)	9,9	32,36	51,51
OPD	12.50	(90,95)	27,37	106,136	228,296
IPD	12.50	(82,89)	35,57	93,229	225,594
VPrabha	7.50	(85,100)	41,41	117,117	250,250
TBTMS	1.50	(65,75)	44,44	158,163	317,324

*Application Maturity in % (as rated by the developer of application)

The IPD had evolved the most, with number of pages increasing from 35 to 57, regions increasing from 93 to 229, and items increasing from 225 to 594. On the other hand the Vivek Prabha had evolved the least in which the total number of components remained the same.

It is interesting to note that the maturity level of application varies significantly and is not correlated with the time interval between releases of the two versions. For example, the OPD and IPD though released together vary in evolution characteristics due to difference in their maturity level, which can be partly be attributed to the application complexity.

4.2 Experiment I: Application Analysis Overheads

We measured the average time taken to perform the analysis and the overheads in various subtasks. Figure 7 shows the time spent vs. # application components (# pages + # regions + # items) in the evolved version of the application. The effect of each component individually on this behavior is elucidated in the following paragraphs.

As expected the use of page ids for establishing page equivalence results in minimizing the overhead. The task primarily involves issuing queries against APEX views to mark pages as inserted, deleted, or same. The overhead observed was (1.37s, 5.50s, 4.52s, 6.66s, 5.32s) respectively for the five applications in page analysis. Thus, it is evident that page comparison is much smaller in comparison to the overall analysis time (Figure 7).

The region analysis involves computing pair-wise similarity measure that dominates the processing cost. The analysis time depends on the number of same+ modified regions and very less on the number of deleted + inserted regions as the inserted and deleted regions do not require comparison of the properties. Also, the presence of exactly similar regions raises the time as it involves comparing all the properties (Algorithm of Section 2.2).

Regions like HTML and PL/SQL having significantly large source demand more time as they involve an asynchronous system call to java application [12] to perform source diff. The use of asynchronous system calls is a limitation due to the choice of Oracle XE, which does not support execution of java stored procedures.

The above two factors, pair-wise region similarity computation, and pair-wise region source diffs, are clearly visible in the experiments giving the results of (10.80s, 106.80s, 33.70s, 54.0s, 36.29s) respectively for the region analysis time for five applications. TB Registry having less number of same + modified regions attributed to the very less analysis time. Vivek Prabha and TBTMS having higher number of same +modified regions than the other applications contribute to its higher analysis time. Vivek Prabha though having lesser regions compared to TBTMS has higher analysis time as it has many regions with source code thereby requiring us to do source code diffs.

Establishing the item equivalence by their name reduces the analysis time to a large extent. Thus, the item analysis overhead (6.31s, 32.46s, 26.90s, 41.11s, 27.88s) is lower than the region analysis overhead despite number of items are much larger than number of regions. Here also only the same + modified regions contribute to the analysis time, for which we need to compare various properties.

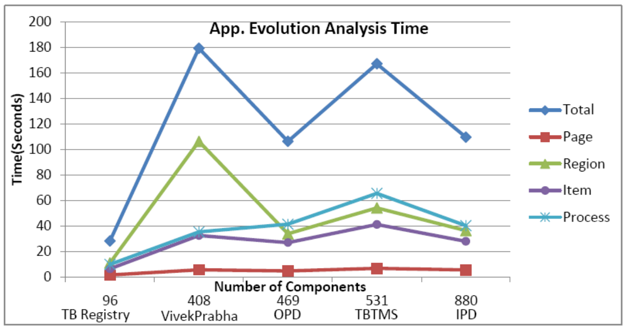


Figure 7: # of App Components vs. Analysis Time

Process overhead cost (9.69s, 35.23s, 41.27s, 65.53s, 40.15s) is in general higher than other subtasks as it involves invoking the java application to perform source component diff. The effect of process on overall analysis time is similar to that for items.

Overall, the total time for analysis, was 28.18s, 179.49s, 106.39s, 167.32s, 109.65s respectively for TB Registry, Vivek Prabha, IPD, TBTMS, and OPD applications, which is acceptable.

Thus, the overall time taken for analysis increases not only with the total component count, but also with the number of same plus modified components. This is evident from the two peaks in the graph at Vivek Prabha and TBTMS, which have fairly large number of same and modified components as reported in section 4.3.

4.3. Experiment II: Application Evolution Characteristics

The objective of this experiment is to observe how the various applications have evolved. The Table 2 gives the evolution characteristics for the five applications.

Table 2: Application Evolution Characteristics

	Type	Inserts	Deletes	Same	Changed
TBRegistry	Page	0	0	1	8
	Region	7	3	21	8
	Item	1	0	50	0
OPD	Page	10	0	10	17
	Region	31	1	62	53
	Item	89	21	183	29
IPD	Page	22	0	11	24
	Region	143	7	42	46
	Item	376	7	206	12
Vivek Prabha	Page	0	0	29	12
	Region	0	0	103	28
	Item	0	0	242	8
TBTMS	Page	0	0	34	10
	Region	7	3	136	51
	Item	7	0	311	6

The five applications analyzed can be placed into two categories: 1) *evolved significantly* (TBRegistry, OPD, IPD) have evolved significantly, and 2) *evolved* *moderatley* (Vivek Prabha and TBTMS). Also, the deletion of pages is absent for all applications.

An interesting evolution trend in (page, page-region, region-item) hierarchy is visible in Figure 8 to 10. For significantly evolving applications, page evolution is dominated by modifications followed by insertions. For moderately evolving applications, page evolution is dominated by identical pages, followed by modifications.

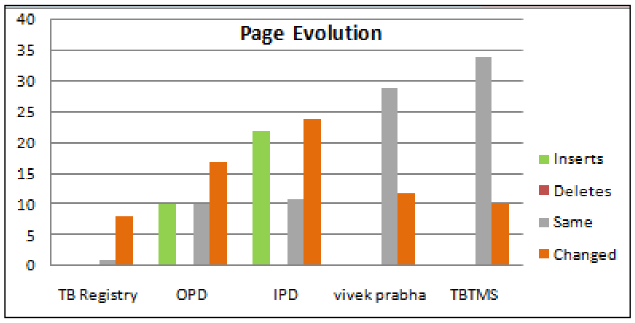


Figure 8: Page Evolution Characteristics

In region evolution, for significantly evolving applications there is no trend. However, for moderately evolving applications, the region evolution trend is similar to the page evolution, i.e., identical regions count is higher followed by modified region counts.

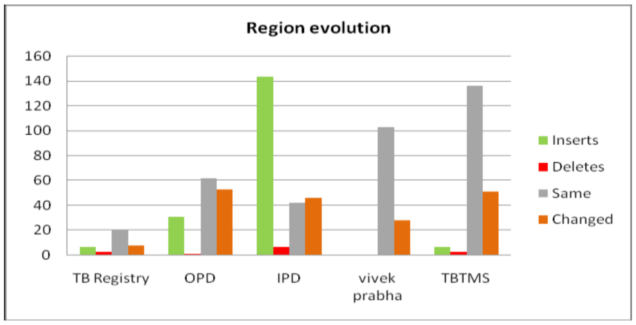


Figure 9: Region Evolution Characteristics

The item evolution, the counts of identical items are in general higher than counts of modified items. Also, insertion counts are high for two of the significantly evolving applications.

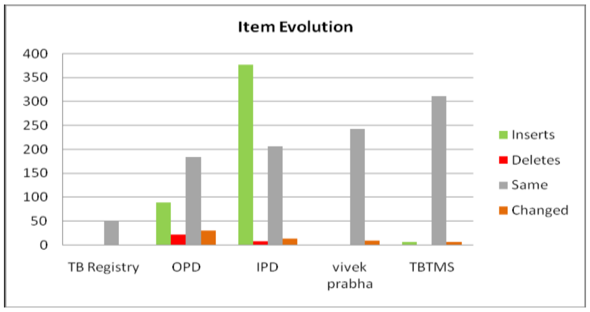


Figure 10: Item Evolution Characteristics

The evolution trends across the three levels (pages, regions, items), intuitively makes sense as at lowest level, we expect more insertions and hardly any changes, whereas at highest level, we expect mostly same, followed by some changes, and hardly any insertions or deletions.

4.4 Experiment III: Application Evolution Characteristics: Layout vs. Content Changes

This was a variation of the Experiment II, with the objective of finding out how many changes can be attributed to layout as opposed to content (PLC is page layout change, PCC is page content change; similarly, RLC and RCC are region layout and content changes). Figure 11 shows the layout vs. content changes for the five applications.

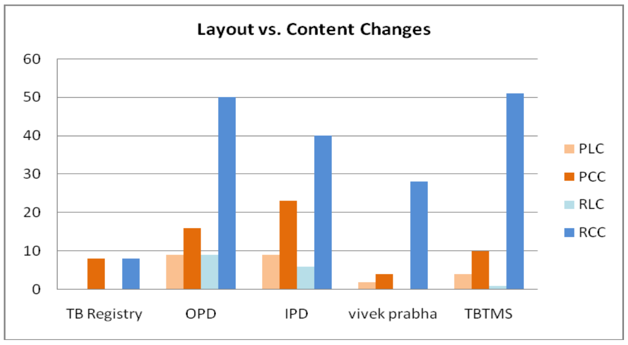


Figure 11: Layout vs. Content Changes

As expected, the content change (shown by darker shades) dominates both page and region components. Also, the layout changes when present are proportionately higher at page level as opposed to region levels.

5. Discussion

In this section, we discuss the pros and cons of the APEX specific assumptions made during analysis and implementation and outline solutions in context of more generic web development environments where such assumptions are not necessarily valid. We also discuss performance, goodness of similarity measure, threshold selection, alternate presentation techniques, and advantages over traditional source code control systems.

5.1 Performance Analysis

While comparing two applications along the three hierarchies, as we drill down from page to page-regions to region-items, the growth ratio is roughly 1:3:6 for the analyzed applications in Section 4. If we derive equivalence by unique identifier or name such as for pages and region-items, we have an overall complexity of *O(nlogn)* but it grows to $O(n^2)$ if a similarity measure has to be used such as for regions.

Performance is also dependent on the extent of evolution. If the application has evolved significantly, the insertions dominate and the number of same plus modified regions is less, thus reducing the region similarity computations. However, for moderately evolved application, the count of same plus modified is dominant increasing the region similarity overhead.

In general, for region similarity, it would be useful to cluster regions into groups using a heuristic so we only need to perform pair-wise matching within the groups.

Another bottleneck is the process diff calculation, which although has to be performed *O(n)* times, where n is the number of pages (or regions), could be significant due to large unit cost. This can be avoided by comparing message digests to determine identical components.

5.2 Goodness of Similarity Measure and Threshold Selection

Selection of a threshold value for a similarity measure is critical to the success of matching components based upon the measure. A similarity measure of 0 represents dissimilar components, whereas measure of 1 representing identical components. Thus, we need to pick a threshold so that (threshold, 1) represents similar components. If the threshold is too high, it tends to categorize the elements as different, but if too low, it can categorize dissimilar components as similar.

For the region similarity function of section 2.2, we intuitively say that two regions are similar if at least half of the content of their properties is same. Thus, the two region sources and the two region names are considered a match if their respective similarity score crosses a value of 0.5. The threshold for the region item and the column counts is also calculated on similar lines as follows:

Let *a, b, c* be number of items in first, second, and common as described in Section 2.2. In order to calculate the threshold, we consider the number of common items must be greater than half the total number of items, which implies that the no. of items changed (no. of items deleted from old app. + no. of items added in new app.) < common no. of items.

i.e. $(a+b-2c)<c$, which gives $\dfrac{2c}{a+b}>\dfrac{2}{3}$

Thus the threshold for each property is as follows:

- *Region Source Threshold* is 0.5 (RST)
- *Region Name Threshold* is 0.5 (RNT)
- *Region Items Threshold* is 0.66 (RIT)

(Also for Report Regions, with columns treated as items)

It's obvious that all regions do not have all the properties. E.g., the HTML regions may not have any columns in them. Similarly a report region may not have any items in it. So, depending on the number of properties compared we vary the threshold and the threshold is calculated as the weighted average of thresholds of the properties compared.

5.3 Establishing Page Equivalence by Name

In our analysis, we assume that pages in the two versions can be matched based on their unique and fixed page identification numbers. For other environments (ASP.NET or PHP), we could consider the filename to be analogous to the page ids. However, this may not always hold since page ids and filenames can be changed, though, such changes are relatively rare.

Under such circumstances the first step is to establish page equivalence between the two versions of the application. We suggest using a combination of heuristic measures as applied in section 2.2 for deriving region equivalence. Comparing the page items and computing an overall region match score will be a good classifier. Layout information can also be considered, but, pages in a web application can have very similar layouts.

5.4 Pre-computing Changes all at Once vs. Lazy Computation on Demand

A page can be marked modified if a change is found in the highest level of the model and view hierarchy computations. This information is sufficient to render the page level diff trees. The lower level calculations can be done on a demand basis reducing the initial processing time. However, for pages which remain unmodified the hierarchies have to be built up to the leaves to make sure no changes have occurred.

In conventional source code control software each file is assigned a version number and/or a timestamp. If such information is available for the application files, it can be used to compute the page level diff trees and inter page diff processing for all pages can be deferred.

5.5 Current Presentation Paradigm and Alternatives

EAST application presents a symmetric visualization of the changes between the two applications. The differences are presented with respect to each other in a two column format. In some cases it may be useful to present a diff with respect to the older application only. It may be sufficient to highlight only the changes made in the new application as annotations in the application summary [14] of the old application. Such alternatives can be implemented as per requirement while keeping the core diff computation unchanged. However, depicting branching transition diffs and layout changes in such a scheme (singe column) would be challenging.

5.6 Advantages over Source Code Control Systems

Conventional RCS and CVS systems [19, 20] used today exploit file comparison operations to detect changes and maintain versions of the software. The framework outlined in this paper maps the detected changes into the model, view, controller and database schema components of a web application. The changes in different components are managed separately in hierarchies where each node represents the container component of its child nodes. This abstraction allows better modelling and visualization of the evolution in a web application. In [21], we explore the feasibility of developing a system like EAST for other web development environments.

6. Conclusions and Future Work

The paper presented a scheme for analyzing and modeling synopsis of an evolving database-centric web application. Assuming a Model-View-Controller architecture for web application, the analysis accounted for content changes as well as layout changes. The presentation of these changes in a two-column format allowed us to visually see the changes with drill-down capabilities along the page, page-region, region-items hierarchy. The scheme also took into account the changes with respect to schema dependencies.

This scheme was used to implement EAST evolving application synopsis tool, which was used to analyze evolution of already deployed applications (in successive releases). The tool was able to capture and represent both content as well as layout related changes fairly accurately. The drill-down capability allowed us to examine the next level of changes in an easy manner.

We plan to use the tool for studying changes between released versions of the application. In future, we plan to enhance the tool to handle the case where the dependency on referenced schema object remains the same for the two versions of the application but the referenced schema objects evolve. Also, we plan to extend this scheme to work for environments (e.g PHP web application) where the application directly generates the HTML code (as opposed to via a tool). The lack of readily available application metadata makes this task challenging.

7. Acknowledgments

We thank developers at Sarada Research Labs, Bangalore, for providing us versions of various applications for our experimental study and for providing valuable feedback.

References

[1] T.Koppelaars, *Building Robust Applications in a Database-Centric Way*. http://web.inter.nl.net/users/T.Koppelaars/J2EE_DB_CENTRIC.doc

[2] J. Ploski, W. Hasselbring, J. Rehwinkel, S. Schwierz: *Introducing Version Control to Database-Centric Applications in a Small Enterprise*. IEEE Software 24(1): 38-44 (2007).

[3] Oracle Application Express, http://www.oracle.com/technology/products/database/application_express

[4] J. W. Hunt and M. D. McIlroy. *An Algorithm for Differential File Comparison*. Computing Science Technical Report, Bell Laboratories 41, 1976.

[5] S. Horwitz. *Identifying the semantic and textual differences between two versions of a program*. ACM SIGPLAN Notices 25(6):234-245. June 1989.

[6] W. Yang. *Identifying Syntactic Differences Between Two Programs*. Software--Practice and Experience, Vol. 21(7):739-755, July 1991.

[7] J. F. Roddick. *A Survey of Schema Versioning Issues for Database Systems*. Information and Software Technology, 37(7):383–393, 1996.

[8] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, J. Widom: *Change Detection in Hierarchically Structured Information*. SIGMOD Conference 1996: 493-504.

[9] Y. Wang, D. J. DeWitt, J. Cai: X-*Diff: An Effective Change Detection Algorithm for XML Documents*. ICDE 2003: 519-530.

[10] V. Levenshtein. *Binary codes capable of correcting deletions, insertions, and reversals*. Sov. Phys. Dokl., 6:707–710, 1966.

[11] G. Cobena, S. Abiteboul, and A. Marian. *Detecting changes in XML documents*. In Proc. of ICDE, 2002.

[12] Google Diff Match Patch, http://code.google.com/p/google-diff-match-patch/

[13] G.E. Krasner and S.T. Pope, *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*, tech. report, ParcPlace Systems, Mountain View, Calif., 1988.

[14] S. Sadhwi, A. Biswas, J. Srinivasan. *An Application Synopsis Tool for Database Applications developed using Oracle Application Express,* India Software Engg. Conf., Pages: 113-118, Mysore, India, 2010.

[15] Oracle Database 10g Express Edition, http://www.oracle.com/technetwork/database/express-edition/overview/index.html

[16] Oracle PL/SQL Programming Language http://www.oracle.com/technetwork/database/features/plsql/overview/index-101230.html

[17] J. Madhavan, P.A. Bernstein, E. Rahm. *Generic Schema Matching using Cupid*. VLDB 2001: 49-58.

[18] Oracle SQL Developer, http://www.oracle.com/technetwork/developer-tools/sql-developer/

[19] W. F. Tichy, *RCS-A System for Version Control*, Software-Practice & Experience, 15 (7): 637-654, July 1985.

[20] D. Grune. *Concurrent Versions System, a method for independent cooperation*, IR 113, Vrije Universiteit, Amsterdam, 1986.

[21] S. Karnati, S. V. Madhava Krishna, A. Biswas, J. Srinivasan. *Tracking changes in Evolving Web Applications*, Submitted to ISEC 2011.