A Novel Framework for Dynamic Software Architecture Recovery of Web Applications

Ahmed Raad Raheem[1], Research Scholar, Dept. of Computer Science & Engineering, Acharya Nagarjuna University, Guntur.
Dr.Shaheda Akthar[2], Registrar$_{FAC}$ Dr.Abdul Haq Urdu University, Kurnool.

Abstract
Very often, the developed web applications are poorly structured and poorly documented that makes maintenance of such systems problematic. This paper presents an approach to recover the architecture of such systems to make maintenance more manageable. The approach extracts the structure of web applications and shows the interaction between their various components. The recovery process uses a set of extractors to analyze the source code of web applications. The data thus derived is further analyzed so that the architectural diagrams get more simplified, instead of being complex. Developers can use the extracted architecture to gain a better understanding of the web applications and assist in their maintenance.
Keywords: Reverse Engineering, Web Engineering, Architecture, Software Architecture Recovery

1. Introduction
Software Architecture modeling and representation is very important in the software development process. In essence, by using Software Architecture, we are allowed to get a high level view that proves to be quite beneficial in the entire software life cycle. The system parts are reused as components in Component-based software architecture. A software system Architecture is clearly referred to as "the basic organization in which system parts are enclosed in relation to other components as well as the ecosystem and the principles guiding its design and evolution". Further, it represents a process in which software is designed at a higher level with all the different sub-systems distinctly characterized along with their associations. In fact, it is pertinent to develop understanding about all the significant stages of software lifecycle viz., maintenance, evolution and reuse, though architectural information remains elusive in case of a number of systems and is tough to obtain. In this backdrop, it is expected of the software managers to try and recover the core architecture from the source code through some extra effort. When a set of methods representing architectural information are extracted from base levels of a software system like source code, it is known as Software architecture recovery. While generating architectural elements, the abstraction process frequently involves clustering source code entities (such as files, classes, functions etc.) into subsystems according to a set of criteria that can be application dependent or not. Component-based software architecture as a high level abstraction of a system uses the following architectural elements: components which describe functional computing, connectors which describe interactions and configuration which represents the topology of connections between components.

1.1. Architecture representation/properties
Architecture has different stakeholders with different concerns. Software developers are supported by Architecture representations to illustrate access in explicit manners, besides managing the software system architecture. Architecture representation consists of structural and non-structural information about software architecture. Structural information includes components and connectors describing the configuration of a system and non structural information are architectural properties. Architectural properties include example, safety patterns, communications patterns, behavioral patterns, structural patterns and creational patterns. The recognition of different types of similar patterns is very important knowledge for understanding the existing legacy systems and architecture recovery. The user understands the conceptual and concrete architecture of the system through architectural documents, design patterns, source code and architectural properties. The architecture properties cannot be ignored during the recovery of different architecture artifacts.

1.2 **Related Approaches**

There are different approaches in reverse engineering, which can be attempted at different levels of abstractions. These approaches are related to our work. In general, the components are recovered by applying structural recovery techniques. Murphy's Reflection model allows the user to test the high level conceptual model of a system against the existing high level relations between the components of the system. On the basis of type of information generated, the classification of recovery approaches are done as shown below:

- Data Flow-based approaches
- Knowledge-based approaches
- Design pattern-based approaches
- Program slicing-based approaches
- Formal method-based approaches
- Program comprehension-based approaches
- Domain-based approaches
- Clustering-based approaches
- Concept analysis approaches
- Machine learning approaches
- Metrics-based approaches
- Structural-based approaches

In architecture, modeling language is used to codify specific architectural styles, while a recognizer tools tries to identify instances of such styles. The program slicing is the basic mechanism used. In contrast to these approaches, we use commercially available reverse engineering tools, along with application domain knowledge introduced by a human engineer to complete the automatically recovered software views by manually generated ones.

1.3. General framework for Architecture Recovery

The architecture recovery Framework integrates the architecture recovery tools to support the architecture recovery process. Primarily, block-line diagrams represent architectural information, though such information remains largely inbuilt. They are often concealed in diverse styles. The source codes are manifest in varied forms, while design documentation is also done in diverse ways. The extraction of architectural information is done by using different techniques and tools. Fig. 1 sketches an overview of the framework for an architecture recovery. The source code, design documentation and domain information are usually imputed in the recovery process. It also uses the artifacts endowed by pattern-based, clustering techniques, along with available expertise. Finally, results are represented in different formats and styles. When design documentation and domain knowledge are recovered, they add supplementary data to the on hand abstractions that include data flow diagrams. This process helps in generating added software views as seen in cases of state transition diagrams, component diagrams as well as architecture descriptions.

The source code and required artifacts can be extracted with the help of reverse engineering tools. Reverse engineering tools perform static analysis on the code and extract information like call graphs, cross-reference tables, data flow diagrams, quality metrics, and hierarchies in classes, relationship and other useful information. Reverse engineering tools provide a higher level of abstraction since irrelevant information from the specific view is excluded. After analyzing and verifying the outcomes yielded by reverse engineering tools with some known source knowledge like documents, source codes and comments, the user data is integrated into the tool to create various lexical manifestations. RE tool generates different views which can be used to recover the architecture of the system software, which can be aligned with a bottom-up approach for taking artifacts as inputs in order to spawn reviews.

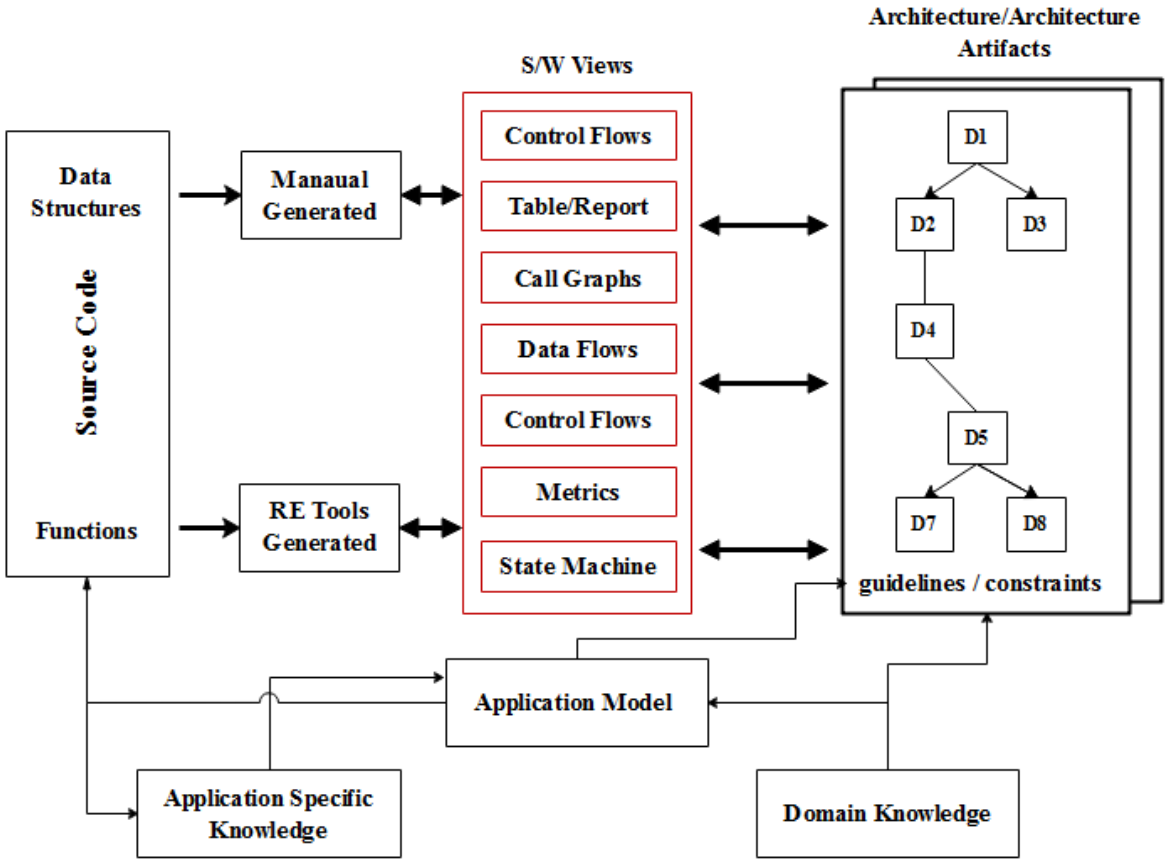

Fig. 1 General framework for Architecture Recovery

The process for Architecture Recovery:

- Study the different architecture recovery approaches (such as Domain-based, Design pattern-based, clustering-based etc).
- Develop architecture conceptual model and formed architecture hypothesis on the system and its structure.
- Analyze, verify and refine the architecture hypothesis against the software system under study.
- Generate different software views and architecture styles of the examined system.
- Iterative use of Architecture Recovery Process.
- Use reverse engineering tools.
- Conducted a case study on the code of five different programming languages software.
- Used existing documents for gaining understanding of system structure and its components.

1.4. Recovery Process

The choice of the right architecture recovery process is critical to extracting artifacts out of the legacy system architecture, which is defined as per the nature of a system. The Recovery Process adopted in our study consists of following phases:

- Architecture concepts
- Legacy architecture analysis
- Extraction
- Abstraction
- Evaluation
- Presentation.

In the first step, we built the hypothesis about the architecture of the existing system. In the second phase, we analyzed the hypothesis developed in the first phase with the help of tools. The next phase extracts the different artifacts from the system by using extraction techniques and Reverse Engineering tools. The Abstraction process produces architecture styles and views at different level of abstractions. In the evaluation stage, the results are evaluated and compared with existing sources of information. Lastly, after recovery, the architecture gets manifest in distinct formats, styles as well as UML notations.

2. Software Architecture Recovery for Web Applications

The web browser is found everywhere and the simple interfaces have opened the door for the development of many new distributed applications. At the start, web pages look like static HTML pages that are simply joined together that makes open access to information easily possible. Subsequently, new technologies such as Java and Java Script Pages were introduced. They have facilitated the development of large-scale applications, which harness the power of the web. Unfortunately, the documentation associated with a web application does not commonly exist and if it does, it is rarely complete or up-to-date. With a very short development cycle, software-engineering principles are rarely applied to the development of web applications. The original developers of a maintained web application are often no longer part of the organization. This lack of documentation and system experts increases the cost and time needed to understand and maintain large web applications. Reverse engineering and software visualization have been proposed as techniques to improve the understanding of large traditional non-web applications. In this paper, we describe an approach to assist developers in understanding their web applications. We describe a method to parse and extract relations between the various components of a web application. The extracted components and relations are visualized by using a specialized viewer. This paper considers this visualization to characterize the software architecture of the system. With the advent of the Internet, a new type of application has emerged: web applications, that use the Internet's infrastructure, are being developed and maintained every day. The latest data suggest that the web accounts for thirty percent share among all software applications in a majority of industries, which is likely to exponentially multiply with the penetration of web and increase in user base. The study of web applications is a fairly new field but there have been significant research contributions from different groups worldwide. We focus mainly on two areas of research that are most related to our research on the architecture recovery and visualization of web applications: web engineering and the modelling of web applications. Multiple components get joined in a unified entity to offer functionality to a web application. The following components exist in web applications: web browsers; web servers; web pages; application servers; application pages; databases; distributed objects such as CORBA, EJB and COM; and multimedia web objects such as images, videos, and etc.

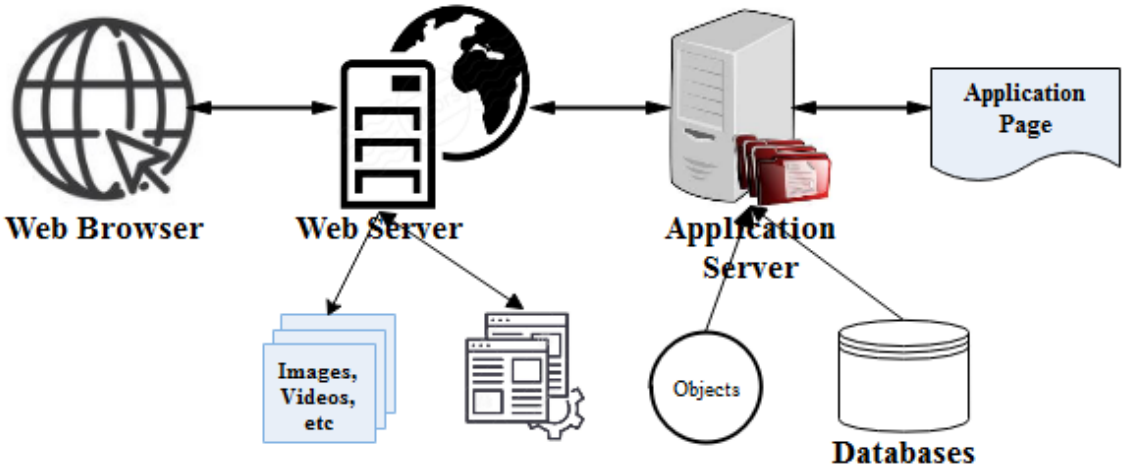

Fig 2 Various components of a Web Application.

The user of the application employs the web browser as the interface to gain access to the functionality of the web application, wherein the browser provides the interface to a user who clicks the links and fills form fields after which the user actions are transmitted to the web server by the browser. The requests are sent by using the HTTP protocol. Upon receiving the request, the web server determines whether it can fulfil the request directly or if the application server must be invoked. The application server and the web server may reside on the same machine or on different machines, who may request for one and the same application as well. In the web server, HTML pages and multimedia images, videos, or audio files are hosted before being forwarded to the application server. Later, the application page is processed by

application server before returning HTML pages to web server. Finally, the web server returns the requested page to the web browser, which displays it to the user. A database is an essential part of a web application as it is the primary communication interface between the many components of the web application. The visualization tools for web applications focus primarily on displaying the hyperlinks between the static pages. As the web was originally developed as a document-sharing platform, these tools approach the problem of visualizing and maintaining web application as a document maintenance problem rather than a software engineering problem.

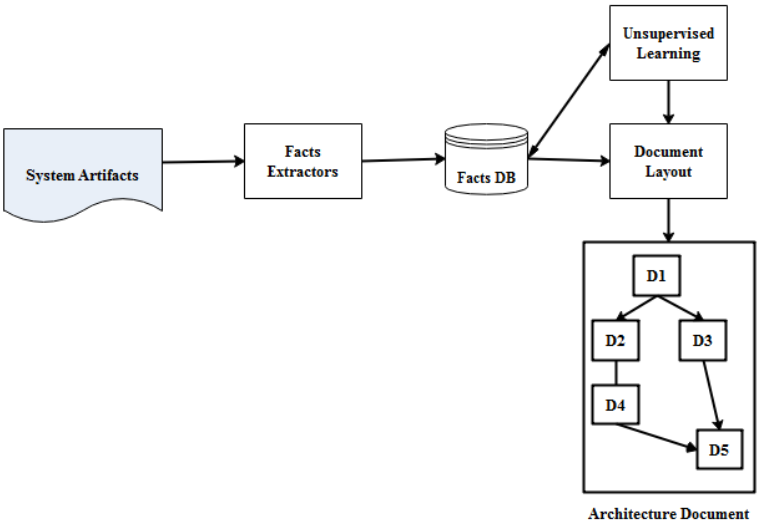

Fig 3. Block diagram

We use a semi-automated process to recover the architecture of web applications. The process uses extractors to analyze the source code of the application. The output of tools is combined with input from a system expert to produce architecture diagrams. Figure 3 shows an overview of the PBS environment. Firstly, the artifacts of the software system (such as source code) are processed using specialized extractors. The extractors automatically generate facts about the software system based on these artifacts. The facts could be detailed such as: function “f” uses variable “a” or at a higher level such as: file “f1” uses file “f2”. The level of analysis done on an extractor determined the quality of extracted data as well as the likely analysis of the recovered facts. Specifically, architecture level analysis does not require a fact at the function level, rather directly lift them to a higher level. The generated facts are stored in table. The schema permits many tools to operate independently on the extracted facts and reduces the coupling between the fact extracting and the fact consuming tools such as the visualizer. Furthermore, the use of schema permits the integration of different types of facts to produce a single architecture document that contains all the extracted facts. Once the facts have been produced, a “first-cut” attempt to visualize them would lead to an architecture view.

2.1. Fact Extraction

In traditional architecture recovery, an extractor parses the source files of the software system and emits facts about the system. Many of the traditional software systems are developed in a single programming language and all the source code of the system is available. For a large number of traditional software systems, only one language specific extractor is needed. Extractors range from lightweight extractors that search for specific patterns of interest in the source code and emit the relevant facts to more detailed parsers. Such parsers may be modified compilers that emit facts about the source code instead of producing assembly code or binaries. Fig 4 shows an overview of the various extractors and their input and the type of facts generated by them. We used five types of extractors: an HTML extractor, a Server Script extractor, and a DB Access extractor. Each extractor is responsible for examining a component or a section of a component. Each extractor generates facts that conform to the CLS schema for web applications. Once all the facts are emitted, the clustering information is combined and all the data are processed by a layout tool. The output of the layout tool can be viewed and analyzed using a visualizer. The directory structure of the

web applications and the source code directory are crawled by a shell script. The script determines the type of the component and invokes the corresponding extractor. For example, if the script determines that a file is a Java script file, then the java script Extractor is invoked. Each extractor generates a set of facts and stores its results in a file with the same name as the input file and the name of the extractor as the suffix. THE FACTS file is combined with the clustering information that is generated using the directory structure and user input.

2.2. HTML Extractor

The HTML Extractor is responsible for processing HTML files. An HTML page is an ASP/JSP page that contains no code segments and is processed only by the HTML Extractor. An ASP/JSP page contains various sections. Each section is parsed and analyzed by the appropriate extractor. An ASP/JSP page contains

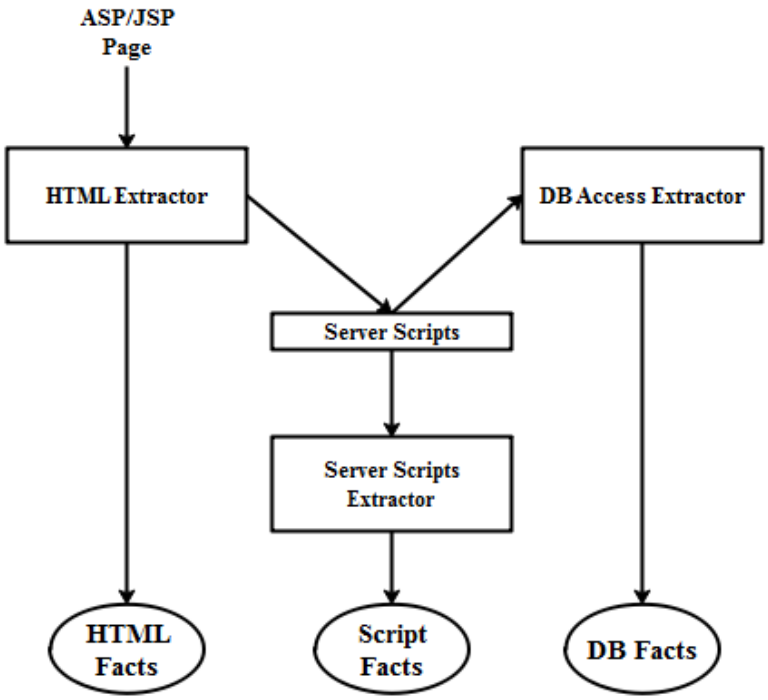

Fig 4. Conceptual Architecture of the Fact Extractors

1. HTML sections are texts with links to other online content. The HTML sections are sent to the requesting browser, without modification by the server.
2. The Server scripts are executed on the server and the result of the execution is sent back to the requesting browser.
3. The Client scripts are interpreted inside of the user's browser. The Client scripts are sent to the requesting browser without modifications by the server.

Each section is written, using a different language:

1. HTML sections represented as HTML.
2. Server scripts expressed as VBScript, Perl or JScript.
3. Client scripts expressed as JavaScript or VBScript.

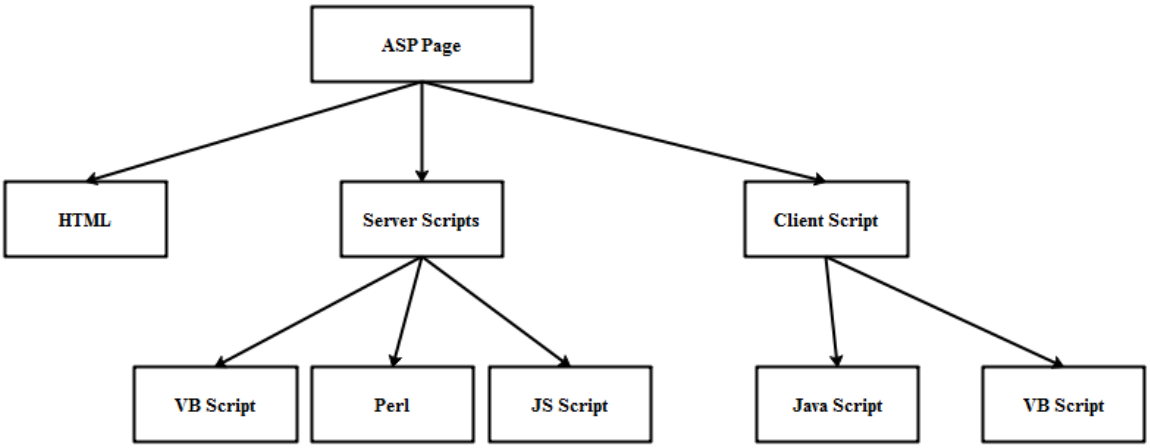

Fig. 5. Internal structure of HTML Extractor

2.3. Database Access Extractor

The Database Access Extractor uses regular expressions to locate data table accesses inside source code statements with the server scripts and the source code of the components as input. The extractor searches for matches that resemble common database access functions and SQL keywords such as SELECT or INSERT. The extractor then employs some heuristics to validate the matches. For example, once the extractor locates the keyword SELECT, it searches for the keyword FROM and determines if a database table is being accessed or if the matches are coincidental and no database table access has occurred on the basis of the distance and the strings between both keywords.

Abstracting and Merging the Extracted Facts

Each extractor emits facts that are language dependent and technology dependent. For example, a VBScript extractor creates outputs indicating that a function is called by another function; whereas the JavaScript extractor emits a fact indicating that the processed file assigns a value to a field in an Enterprise Java Bean (EJB). These facts are technology dependent. Abstractly, both extractors indicate that a processed file is accessing a data field in an object. In one case a file is reading a data field, and in the other case, a file is updating a data field. Once the various extractors have processed the source files of the application, the facts are combined and abstracted to a higher level which is the programming language and technology independent. To handle various kinds of facts that are extracted (and then abstracted), we use a pyramid of schemas as illustrated in Figure 6. The bottom layer of the pyramid has a schema for each source language. The next layer abstracts up to either object-oriented or procedural languages. The next layer simplifies and abstracts up to higher level facts that are common to web applications. Finally, the top layer further simplifies and abstracts to the architectural level. The schemas at these various levels will now be discussed in more detail. Each language extractor has a schema which specifies the various entities it generates and the relations that exist between these entities.

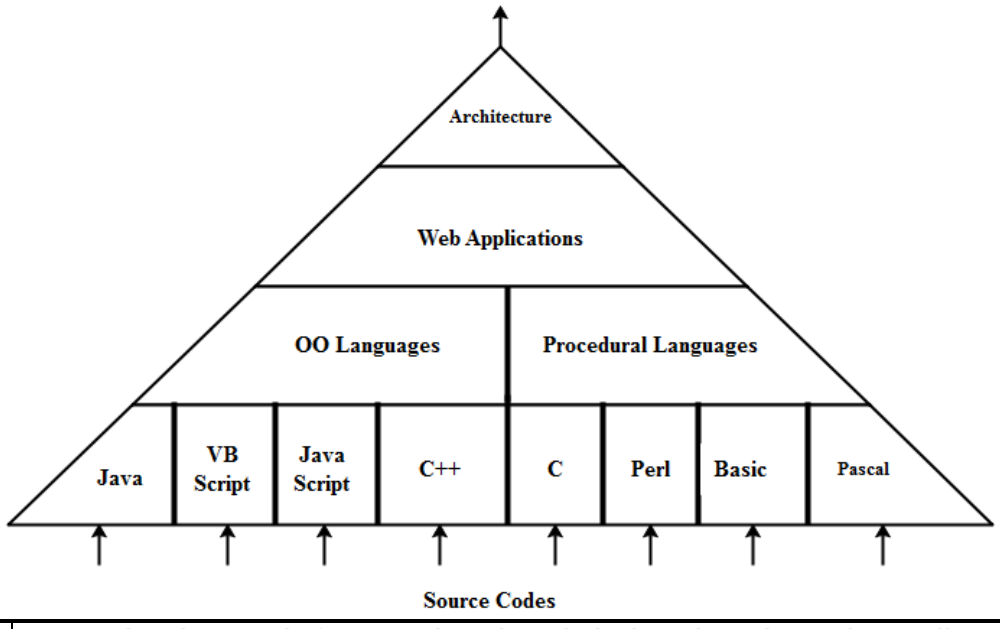

Fig. 6. Pyramid of Schemas

For each extractor in our architecture recovery process, we need to provide a mapping from the schema of the extractor to the object-oriented or procedural schemas. Using the abstracted object oriented schema, we can now study the interaction between components written in different programming languages.

Generating the Architecture Diagrams

In this final phase, the extracted facts along with developer's or architect's input are used to produce the architecture diagrams. Fig.7 shows the steps involved in producing the architecture diagrams. If we directly use the facts at the Web Application Schema level to generate diagrams, we would get excessively complicated diagrams due to the large amount of extracted relations and components. Instead of showing all the extracted relations and artifacts in a single diagram, we decompose the artifacts of the software system into smaller meaningful subsystems. This decomposition reduces the number of artifacts shown in each diagram and improves the readability of the generated diagrams especially for large software systems. A clustering tool reads the facts from the THEFACTS file and proposes decompositions based on heuristics such file naming conventions, development team structure, directory structure, or software metrics. The decomposition information along with the extracted facts is stored back into the THEFACTS file so other tools can access it. An automatic layout tool reads the stored facts and the clustering information to generate diagrams. The layout tool attempts to minimize the line crossing in the generated architecture diagrams is supported to improve these diagrams. By automating as much as is possible in this process, we are able to dramatically reduce the recovery time of large software systems.

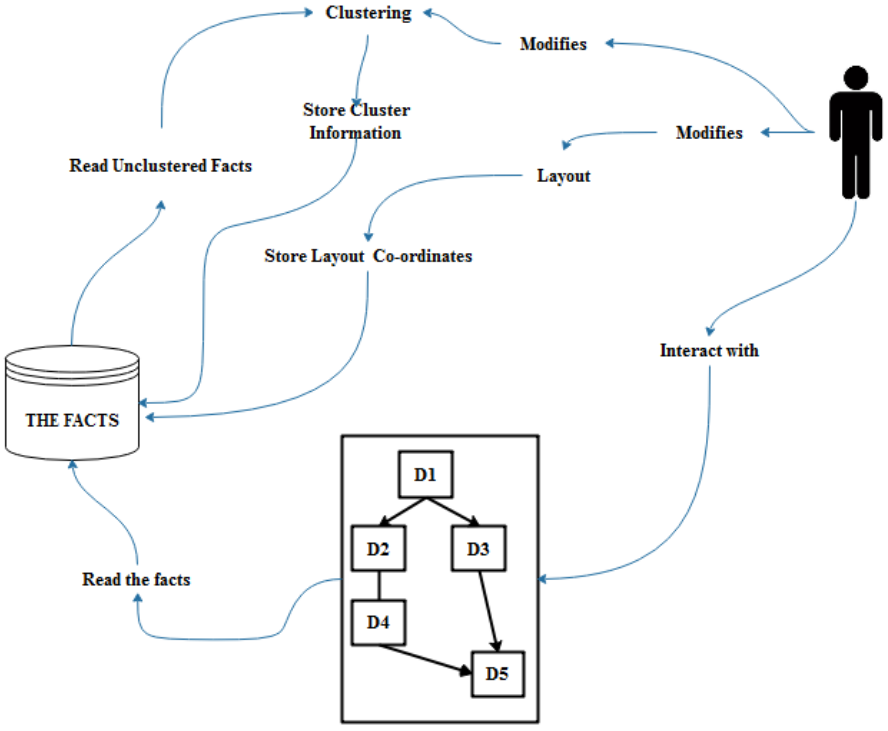

Fig. 7. Generating architecture diagrams from facts

Conclusion

Maintaining web application is not an insignificant task. Developers need tools to assist them in understanding complex web applications. Unfortunately, current tools are implementation focused, while current web applications tend to have little documentation. In this paper, we have shown an approach that can recover the architecture of a web application and show the interactions between its various components. The approach is based on a set of extractors that co-operate to parse the source code of the application and gather data which is later processed and visualized. Developers can use these visualizations to gain a better understanding of their applications before they attempt further modifications to add any new functionality or fix bugs.

References

[1] I. T. Bowman. Architecture Recovery for Object Oriented Systems. Master's thesis, University of Waterloo, 1999.

[2] I. T. Bowman, R. C. Holt, and N. V. Brewster. Linux as a Case Study: Its Extracted Software Architecture. In IEEE 21st International Conference on Software Engineering, Los Angeles, USA, May 1999.

[3] P. J. Finnigan, R. C. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. A. Muller, J. Mylopoulos, S. G. Perelgut, M. Stanley, and K. Wong,The software bookshelf. IBM Systems Journal, 36(4):564–593, 1997. Available online eat http://www.almaden.ibm.com/journal/sj/ 364/finnigan.html

[4] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia. Web Site Reenginnering using RMM. In Proceedings of euroREF: 7th Reengineering Forum, Zurich, Switzerland, March 2000.

[5] A.E.Hatzimanikatis, C.T.Tsalidis, and D.Christodoulakis. Measuring the readability and maintainability of Hyperdocuments. Software Maintenance: Research and Practice, 7, 1995.

[6] Pearl Brereton, David Budgen, and Geo_ Hamilton. Hypertext: The Next Maintenance Mountain. Computer, 31(12), December 1998.

[7] Ivan T. Bowman, Richard C. Holt, and Neil V. Brewster. Linux as a Case Study: Its Extracted Software Architecture. In IEEE 21st International Conference on Software Engineering, Los Angeles, USA, May 1999.

[8] Cornelia Boldyre_. Web Evolution: Theory and Practice, 2000. Available online at <http://www.dur.ac.uk/cornelia.boldyreff/lect-1.ppt>

[9] Grady Booch. The architecture of Web Applications, 2000. Available online at http://www.developer.ibm.com/library/articles/ booch_web.html>

[10] Ilka Philippow, Detlef Streitferdt, Matthias Riebisch. Design Pattern Recovery in Architectures for Supporting Product Line Development and Application,2001.

[11] Kim, H., Boldyreff, C.: A method to recover design patterns using software
product metrics. In Proc. of the 6th International Conference on Software Reuse, ICSR6, 2000.

[12] Ilian Pashov, Matthias Riebisc,Using Feature Modeling for Program Comprehension and Software Architecture Recovery,2002.