OPAALS PROJECT

Contract n° IST-034824

WP10: Sustainable Research Community Building in the Open Knowledge Space

Del 10.17 – DSSR Final Release
+
Wiki Engine as a Component

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: D10.17

Due date: M42

Delivery Date: Sept 2010

Short Description:

This deliverable is made up of two parts:

(a) Semantic Search Component has been developed and integrated into the Flypeer architecture.

(b) A Wiki engine was integrated into the web technology component stack and its utility was demonstrated.

Author: Himanshu Govil, Kaipa Kartik, Ankit Jain, Prabhakar TV

Partners contributed:

Made available to: SUAS, TUT, OPAALS Consortium

Versioning		
Version	**Date**	**Name, organization**

Quality check

Internal Reviewers: Paul Krause, Amir Reza Razavi

Dependencies:

Achievements*	The semantic search was integrated into the Flypeer architecture. Hypertext as a basic datatype was made available to the web-developer by integrating a wiki engine into the web technology component stack. This greatly empowers the web developer making it easy to develop more complex applications.
Work Packages	Work Package 10.23
Partners	All the computer Science Partners
Domains	Computer Science
Targets	Web application developers
Publications*	Not yet
PhD Students*	No PhD students
Outstanding features*	This contribution proposes a new architecture for building web applications by using a wiki engine in the component stack significantly reducing the development time for some applications
Disciplinary domains of authors*	Himanshu Govil, Kartik Kaipa, Ankit JainPrabhakar TV, Dept. Of CSE, IIT Kanpur

The information marked with an asterisk (*) is provided in order to address Recommendation n. 4 from the Year 2 review report

By making use of Wiki Query Language or Wiki Scripting Language people can build lot of web applications, depending on their needs, in a very easy and convenient way. We have built four sample applications to demonstrate its utility.

Distributed Semantic Search

Achievements

Semantic search is enhancing a search operation with an ontology. Distributed semantic search is when there are several repositories on multiples nodes, searching on each of these nodes and presenting the collated search results. Lucene was used to do the basic search and this is essentially extending Lucene over a peer-to-peer network. And when the query is enhanced before performing the search it becomes distributed semantic search.

Outstanding Features

The semantic search component has been developed and tested along with other partners – in the visualisation service and the Flypeer network. The query enhancer can take any user specified ontology or use word net.

Table of Contents

Introduction

The Distributed Semantic Search is an application that facilitates search over local as well as peer repositories. At its core, it uses Lucene[1] indexes and JXTA. Lucene is defined as a syntactic indexer and JXTA is a communication protocol. This work involves extending Lucene in peer-to-peer network.

In the distributed search scenario, each node is an independent document repository. Documents are indexed locally and searched using Lucene. User can specify multiple directories for indexing. This application runs on individual peers in a network and uses JXTA protocol to automatically discover other peers within its subnet. While searching, the application searches local index, broadcasts query to all connected peers and then collects results back. Next it merges all results and represents to user.

Application Architecture

Following figure represents the architecture of the application.

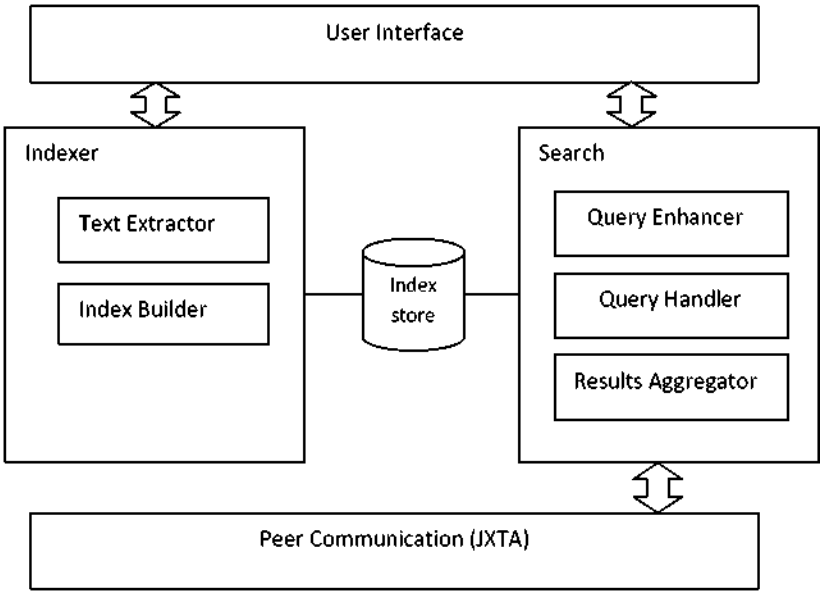


Fig 1. Architecure of Single Node

Indexer: This is responsible for building index from the shared documents. It is composed of following subcomponents

Text extractor – uses Lucene-Tika package to retrieve file contents. The various file formats supported are txt, pdf, html, doc, docx, ppt, pptx, mp3, xml, odp, rtf, etc.

Index Builder – uses Lucene package. It builds a reverse-index for fast searching. The indexer uses local file system to store index.

Peer Communication (Network Layer): This component facilities communication with peer nodes. When a search is performed, the search query is broadcasted to all peer nodes

connected in the network and later results are collected back. This component uses JXTA peer-to-peer protocol for communicating with peers. The JXTA protocol uses XML based messages and it is platform independent. This component is also responsible for peer discovery. Currently the application supports peers in its subnet only.
Search: This component is responsible for Responsible for local repository search. It interacts with local index and returns results. It is composed of following subcomponents:
Query enhancer - enhances search query semantically. Currently this component uses domain ontology or Wordnet to find synset (synonyms) and does query expansion. Further details are available in the section #.
Results Aggregator – merges results collected from peers and produces an aggregated list. It uses match-score (returned by Lucene) of documents to build a merged list of results.
Query Handler – performs search on local index. It also handles remote queries and sends results back to the requesting peer.

The service returns search results serialized in RSS 2.0/XML format as well as JSON for quick consumption on web-pages.

User Interface: The user interface layer is responsible for interacting with user to read input keywords and display results.

Network architecture

The Fig 2 shows network architecture for the application. Each node is autonomous with its own documents repository and search service. Nodes communicate following JXTA protocol. When search is performed on a node, the request is propagated to all connected nodes in the network.

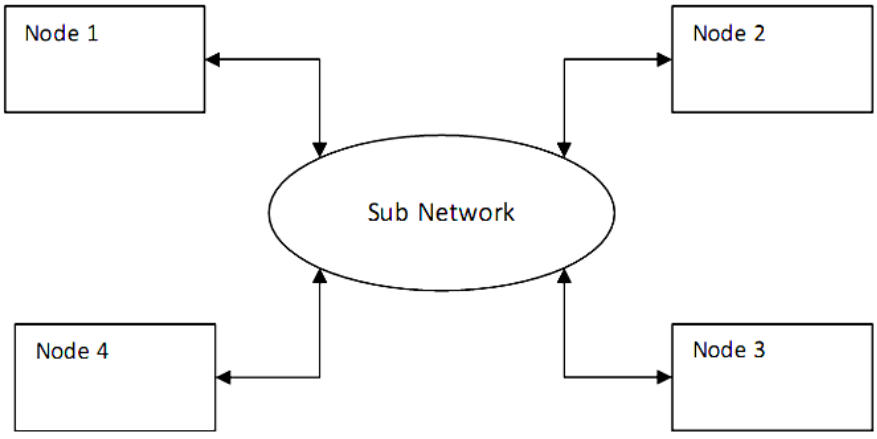


Fig 2: Network architecture for distributed search

Implementation Details

As mentioned earlier, each OKS node is autonomous and runs a local instance of the service. Documents are indexed locally on these nodes and each node is capable of serving remote queries.

The Query Enhancer component shown in the Fig 1 does semantic expansion of user queries. Currently there are two implementations:

1. A Wordnet [2] ontology for English language
2. RDF/XML [3] implementation of any domain specific ontology

Wordnet is a lexical database of English language words. The Wordnet integration is a simplistic implementation where user query is expanded with synonym words.

RDF/XML is a W3C recommended specification for representing knowledge. The knowledge is stored in the form of triples. For example, the XML in Appendix 1 is an RDF/XML serialization of software architecture ontology. The RDF/XML implementation in Query Enhancer component takes a domain ontology serialized in RDF/XML. The component makes use of following three predicate to expand query:

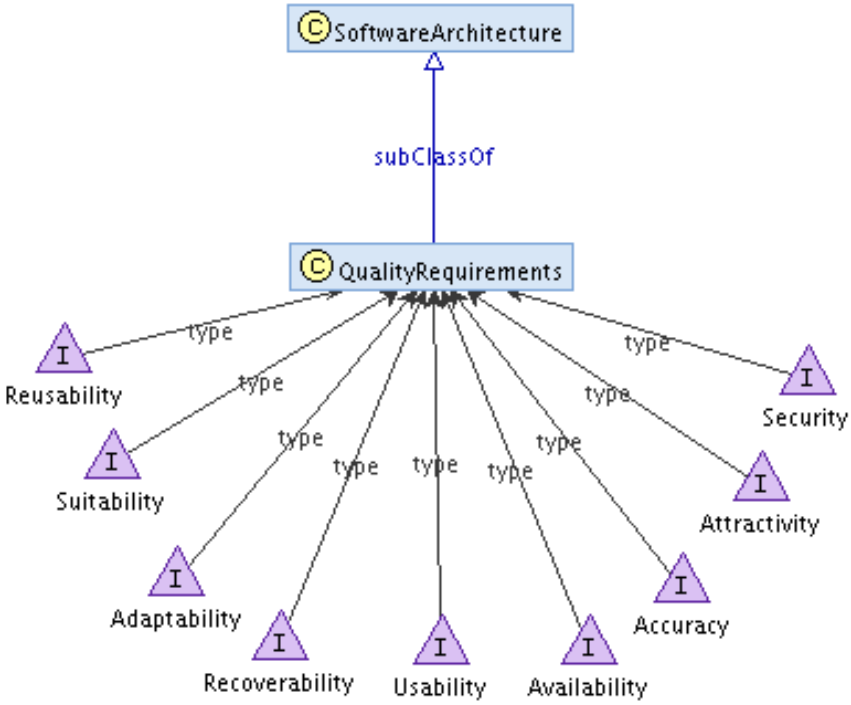


Fig 3: Software Architecture – Quality Requirements

a. rdf:type: The `rdf:type` is used to indicate that a resource is an instance of a class.
b. rdfs:subClassOf: The `rdfs:subClassOf` predicate is used to indicate that all the instances of one class are instances of another.
c. owl:sameAs: The `owl:sameAs` indicates that two terms represent the same resource, i.e. synonyms.

Consider the ontology in Fig 3. The concepts Reusability, Suitability, Adaptability, etc. are the instances of QualityRequirements, and the class QualityRequirements is a sub-class of SoftwareArchitecure. When a search for “quality requirements” is requested, the search query is expanded to include all of its instances. Thus documents with occurrences of “reusability”, “suitability”, “adaptability”, etc. are also returned. Similarly for the ontology shown in Fig 4, when a search is requested for “modifiability tactics” the documents containing terms “localize changes”, “prevention of ripple effect” and “defer binding time” are also returned.

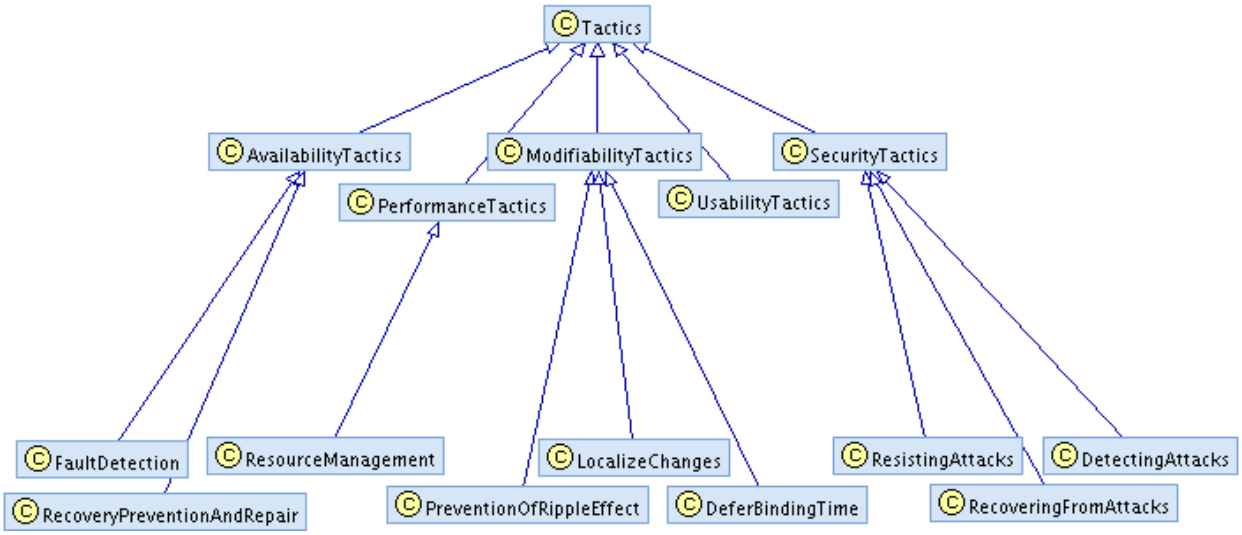


Fig 4: Software Architecture – Tactics

The inclusion of additional search terms may introduce additional documents to be listed in the search results. As these documents were not part of original search term, they may appear irrelevant to user's expectation. In order to solve this noise problem we have boasted the ranking of original search terms. Documents containing original search term appear higher in the results.

Integration with other services:

A rendezvous peer (Flypeer node) has been setup at IIT Kanpur and connected with other rendezvous peers hosted in the Opaals eco-system. In order to integrate semantic search with Flypeer, we created WSDL description of the service, and implemented a wrapper class. The WSDL contains definitions of input and output parameters of the service. The class implements FlypeerBusinessService interface and on receiving service event it returns search results. Once the service is published in Flypeer infrastructure it can be consumed by any other node in the network.

We have showed the integration of search service with following two other components of OKS.

1. Visualization service: The search service provides an HTTP endpoint for consumption by other services. The endpoint exposes two data formats for consumption, RSS/XML and JSON. The *Component-Based Visualization System* (from *www.tut.fi*) has showed visualization of search results by consuming RSS/XML endpoint.
2. Flypeer service: A simple wrapper class has been written to expose the search service on Flypeer infrastructure. Once the service is published in Flypeer infrastructure it can be consumed by any other node in Flypeer.

Future Enhancements

Intelligent query routing: Broadcasting the query to all peers suffers from scalability issue. The application must use intelligent query routing. One of the approaches is using expertise based routing where each peer initially advertises its expertise.

Extending the communication layer to be able to connect to peers outside of sub-network.

Refererences

[1]] http://lucene.apache.org/java/docs/
[2] http://wordnet.princeton.edu/
[3] http://www.w3.org/TR/REC-rdf-syntax/

RDF/XML of Software Architecture ontology

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:softArch=http://www.owl-ontologies.com/unnamed.owl# >
  <rdf:Description rdf:about="&softArch;QualityRequirements">
    <rdfs:label>quality requirements</rdfs:label>
    <rdfs:subClassOf rdf:resource="&softArch;SoftwareArchitecture"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="&softArch;Installability">
    <rdfs:label>installability</rdfs:label>
    <rdf:type rdf:resource="&softArch;QualityRequirements"/>
  </rdf:Description>
  <rdf:Description rdf:about="&softArch;Reusability">
    <rdfs:label>reusability</rdfs:label>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdf:type rdf:resource="&softArch;QualityRequirements"/>
  </rdf:Description>
  <rdf:Description rdf:about="&softArch;Suitability">
    <rdfs:label>suitability</rdfs:label>
    <rdf:type rdf:resource="&softArch;QualityRequirements"/>
  </rdf:Description>
  <rdf:Description rdf:about="&softArch;Recoverability">
    <rdfs:label>recoverability</rdfs:label>
    <rdf:type rdf:resource="&softArch;QualityRequirements"/>
  </rdf:Description>
  <rdf:Description rdf:about="&softArch;Usability">
    <rdfs:label>usability</rdfs:label>
    <rdf:type rdf:resource="&softArch;QualityRequirements"/>
  </rdf:Description>
  <rdf:Description rdf:about="&softArch;Accuracy">
    <rdfs:label>quality requirements</rdfs:label>
    <rdf:type rdf:resource="&softArch;QualityRequirements"/>
  </rdf:Description>
</rdf:RDF>
```

Wiki Engine as a Component in the Web Technology Stack

Achievements

The usage of World Wide Web (WWW) and Internet has increased tremendously in the past decade. WWW has changed the way humans now communicate. Initially when WWW was built, it was an application running on the internet. But now web has itself become a platform and numerous applications have been built making use of web-technology stack which run over the WWW. Online shopping portals and online automated sytems, for e.g. Amazon and online railway ticket booking, are two hugely popular applications built using the same. Wiki is another such application exemplified by Wikipedia which has become one of the biggest knowledge repositories. Wiki has proven to be an excellent collaboration tool and lot of companies and organisations are using it for various projects and team management.

Typically a web application is developed using an application server and a database server like the LAMP stack where Apache is used as web server, MySQL engine is used as the database engine and PHP is used as server-side programming language.

In this work, we present a design on how to integrate a Wiki into the current architecture and demonstrate its feasibility. We have developed a query language named Wiki Query Language which can be used along with a existing scripting language JSP to extract out required data from Wiki. We have also built a server side scripting language named Wiki Scripting Language (WSL) using which developers can make use of Wiki for building web applications. It is similar to PHP when it comes to usage. WSL code can be embedded inside HTML just like PHP. WSL is dynamically typed and can execute WQL queries making use of which one can extract required information from the Wiki engine. We have also built a command like application known as Wiki Shell using which people can write queries to fetch information from Wiki Engine.

Outstanding features

A wiki which was initially an application has now been pushed onto the application stack. The query language itself has an easy to learn syntax which makes it easy to develop applications for a variety of domains.

We have provided a number of ways to integrate WQL into an application including an easy to use installer for windows.

The language at its basic allows one to integrate and collate information from a wiki.
For instance we have built an application for collaboration among team members. The actual usage is simple. All a team member has to do is edit a wiki page that has been assigned to him. The power comes from the fact that using WQL one can integrate information across the wiki pages into a single entity. For example this makes it possible to list on a single page the updates of all the team members. Any update by a team member on the wiki page is automatically reflected on the central page. Another page simply picks out contacts from each user's wiki page and aggregates them.
The possibilities are endless.

Another feature that we have included is the fact that using our query language one can modify wiki pages as well. This gives it many added advantages. For instance if one would like to create similar templates for some sort of objects say students then using Wiki Query language it is possible to do the same with very little effort.

WQL opens up a whole new field of what can be done using a Wiki.

1. Introduction & Motivation

1.1 World Wide Web

World Wide Web, which is commonly known as WWW, is a collection of hypertext where the documents are interconnected and can be accessed through the Internet. Since it now supports graphics, sound, animation and videos as well, one can say that WWW is a distributed hypermedia system [1].

Three key features of the WWW are as follows -

• The Address System: This is based on Universal Resource Identifiers

• A network protocol

• A hypertext markup language [2]

The World Wide Web was developed with an intention to create a pool of human knowledge which people across time and space could share. WWW used to be a simple application running over the internet. However, as time progressed lots of advancements took place and Web has now become a platform. In the last decade the percentage of people using internet has grown drastically. Numerous applications have been built now using web as a platform. Social Networking sites, Wikis, Online

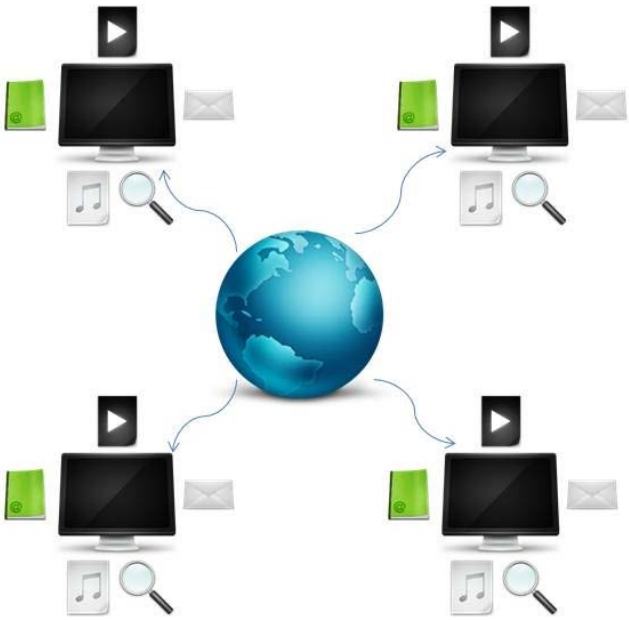


Figure 1.1: Representation of World Wide Web

Shopping portals, e-governance sites etc. are examples of web applications. Web has changed the way humans now live and collaborate. Almost everything is available on the web now.

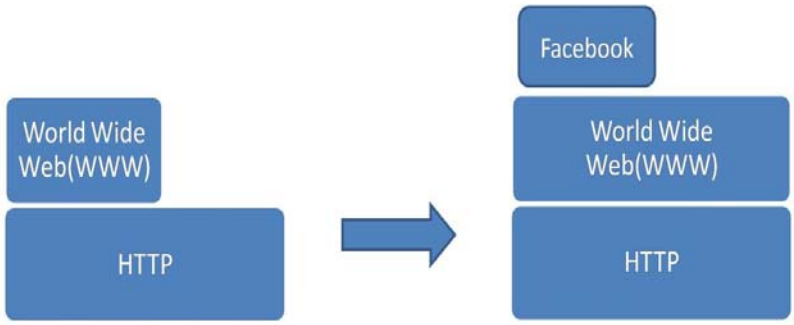


Figure 1.2: Evolution of WWW

1.2 Web Applications

A web application is an application that is accessed over the Internet or the intranet. Web applications can range from simple static web pages to sophisticated applications like online drawing tool. Web based applications have become tremendously popular mainly due to the convenience of using web browser as a client.

Any web application can be easily understood by breaking it into logical chunks called tiers. At the highest level of abstraction, web applications can be organized as three-tiered applications (as shown in figure 1.3).

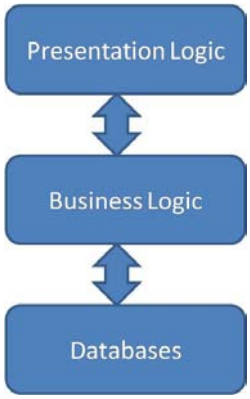


Figure 1.3: High Level Logical View of Web Application

Components in Presentation Logic tier only interacts with components in Business Logic tier. The components in the Business Logic tier contains all the knowledge required to modify the data components that are contained in the Database tier [3].

• Presentation Logic: Presentation logic simply refers to user interface. In this case it sits inside a web browser.

• Business Logic: This tier is also known as Application Logic. It consists of engines using dynamic web content technology like PHP, JSP, ASP, Java etc.

• Databases: The Database tier contains all the data components that are used to provide persistent storage for the application data. Relational Database Engines are the most preferred choice for this tier.

1.2.1 Architecture

Architecture can be best defined as the algorithmic counter-part for large programs or applications. The system architecture defines how the pieces of the application inter- act with each other, and what functionality each piece is responsible for performing. Architecture of web applications can be best defined by client-server model [3]. The runtime image of a web application can be depicted in the way as shown in figure 1.4 [4].

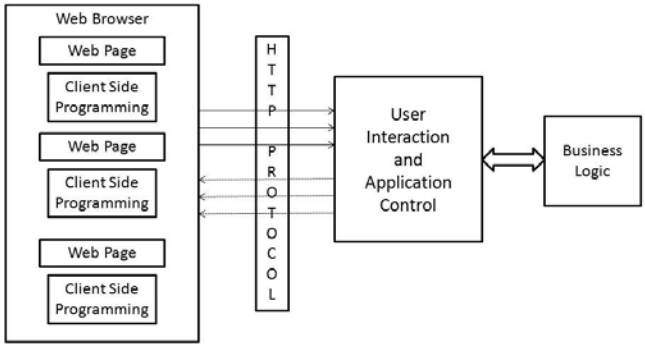


Figure 1.4: Segmentation of Components

1.2.2 Technology Stack

There are various ways in which people build web applications. Following are the essential building blocks required for developing a web application -

• Web Server: The role of web server is to deliver the web pages, as and when client requests, using the HTTP protocol over the World Wide Web. Apache and IIS are two most widely used web servers [5].

• Server-side Logic: Server-side scripting is a technology using which dynamic web pages are generated by running a script on the web server. It is primarily used to build database-driven websites. Some server-side scripting languages are

- PHP, JSP, ASP, Python, Ruby etc.

• Client-side Logic: Instead of executing scripts on web server, developers can execute scripts client-side and it is done by the web browser. Java-script is an example of client-side scripting language.

• Database Engine: It is a system whose overall purpose is to store information and to allow users to retrieve and update that information on demand. Mostly a Relational Database Management System(RDBMS) is used to build web ap- plications. Database is accessed through a query language. Structured Query Language (SQL) is used for Relational Database Management Systems.

1.3 Motivation

Database engines are an integral part of any web application.They provide persistence and sometimes business logic. The data is stored in the form of tables and some functionality is added as triggers. Designing databases for complex web applications is not an easy task and it requires lot of effort to design it so that there is no redundancy and database provides all functionality. Though Relational Database Engines provides
Lots of functionality but they have certain drawbacks as well. Following are few of them

• These days many web applications have huge databases with a large number of tables. In such cases it becomes hard to visualize the database. Finding the useful information hidden in the database becomes difficult and in such cases the database becomes a data-dump [6]. It also becomes difficult for a new person to understand the database design and make changes in it. Sometimes it so happens that person who designed the database for a web application is unavailable and for a new person to make changes in it becomes a challenging task.

• Newbies find it hard to build database-driven websites because learning curve of database designing is steep and a person should have some experience first to design an excellent and scalable database.

• There are cases and scenarios where the nature of information is very dynamic and it keeps changing from time to time. In such cases, most of the times it has been seen that websites do not get updated the moment new information arrives as people avoid updating database tables or static HTML pages from time to time.

• In most of the cases, information providers are not web application experts and they do not know how to insert or update information in the database or to edit a HTML page. Hence, the need for database administrator arises whose only job is to update information. This is clearly a waste of resources and money.

1.4 Our Work and Contribution

Wiki is a web application that allows users to create and edit interlinked web pages in an easy and collaborative manner. Wikis are becoming increasingly popular. This is quite evident from the number of clicks Wikipedia has everyday. During the period

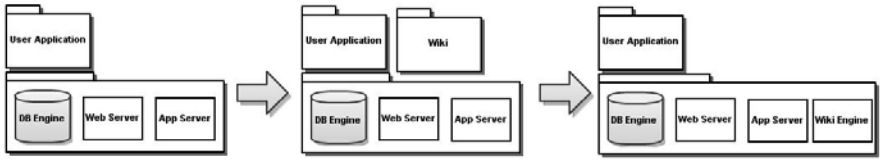


Figure 1.5: Wiki as a platform

from 2001 to early 2007, about 1.6 million articles were written on Wikipedia [7]. Wiki software and platforms have gained attention as a technology that promises to change forms

of work, education and commerce [8]. Wikis have become a popular format for sharing knowledge among people with common interests in academia and professional domains [9]. Wiki has also been proven as an excellent platform for cross-cultural collaboration [10].

In this work we have pushed Wiki engine into the web-technology stack just like sometime back database engine was pushed. So now Wiki will also become a platform(as shown in figure 1.5) and developers will be able to make use of it for building complex web applications. Making use of Wiki one can make lots of web applications in a very easy and convenient manner.

Like Structured Query Language(SQL) is used to manage Relational Database Management Systems, similarly we have built a Wiki Query Language(WQL) using which one can extract out required information from the Wiki. We have integrated WQL with JSP(Java Server Pages). So developers can execute WQL queries from JSP just like they do now for SQL queries. Moreover, we have also built a shell like environment for WQL using which users can write queries and can get the requisite information. Other than WQL, we have also built Wiki Scripting Language(WSL) which is a stand alone scripting language and can be used to generate dynamic web pages by extracting information from Wiki Engine. Wiki can be queried and the required infor- mation can be extracted out by making use of provided functions. It is similar to PHP when it comes to syntax. Just like we can embed PHP code within HTML, the same can be done with WSL as well.

In this work we have tried to overcome all the issues mentioned in section 1.3. One does not need to learn anything new to edit a Wiki. It is like editing normal text. Thus editing a Wiki page is easier compared to editing a table in RDBMS or editing some text in HTML pages. Therefore, information providers can directly edit Wikipages themselves and the web-pages will also get updated automatically. Wiki provides another layer of abstraction over database engine and hence is easy to visualize. This will eliminate the need for having a database administrator for maintaing web applications.

We have built four sample applications to demonstrate the utility of WQL and WSL. In two of them, we have used WQL embedded in JSP and in the other two WSL has been used. The language itself has been developed using JAVA and has a syntax which is very intuitive and easy to learn. For demonstration purposes, we have used Mediawiki as our Wiki engine. The reason behind using mediawiki is that it is the most widely used Wiki and Wikipedia also uses the mediawiki engine.

1.5 Organization of Report

• Chapter-2 describes the architecture of WQL and WSL and the architecture of web applications which are built using WQL and WSL. In this chapter we have also discussed how we have integrated WQL with JSP and the implementation details.
• Chapter-3 describes the syntax of WQL and WSL and semantics of each and every construct.
• Chapter-4 discusses the four sample applications which we have built.
• Chapter-5 contains conclusions and future work.

2. Architecture & Implementation Details

2.1 Wiki

As has been mentioned in Section 1.4, Wiki is a web application that allows users to create and edit interlinked web pages in an easy and collaborative manner. Following the interlinks users can navigate from one page to another. A Wiki can be visualised as shown in figure 2.1.

The power of Wiki lies in simple usage and less technical hurdles. To use a Wiki, clients (end users) require no additional training or software. They can view and edit a Wiki page inside a web browser. They need not learn Hypertext markup language. Editing of Wiki pages is done using extremely simple rules which are very easy to remember as well. Typically every Wiki engine has the following features -

• Edit: Wiki allows its users to edit Wiki pages.

• Inter-links: Each Wiki page can be linked to other Wiki pages.

• Recent Changes: Wiki keeps a log of recent changes made to Wiki pages.

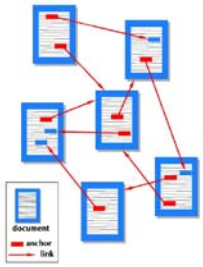


Figure 2.1: Visualisation of Wiki

• Version: Wiki also keeps previous versions of a Wiki page so that users can roll back to the previous version if they want to.

• Search: Wiki allows users to search for a Wiki page. As each Wiki page has a unique title, Wiki allows searching for Wiki pages.

2.1.1 Architecture of a Wiki Engine

The architecture of Wiki is similar to that of a database driven website. Wiki software, also known as Wiki engine, is a server side script whose purpose is to render Wiki pages which can be viewed inside a web browser. The content of a Wiki page is written in simple text which is then stored in a database. The Wiki text also has some rules which specify how the text should be rendered inside the browser. When a Wiki page is accessed, client's browser sends request to the server. Server extracts data from database engine which is simple text in Wiki format. Wiki script then translates this text into HTML and the web server sends it back to the browser and this is how users view Wiki pages.

Let us look at the case of mediawiki engine [11]. In mediawiki, PHP is used as a

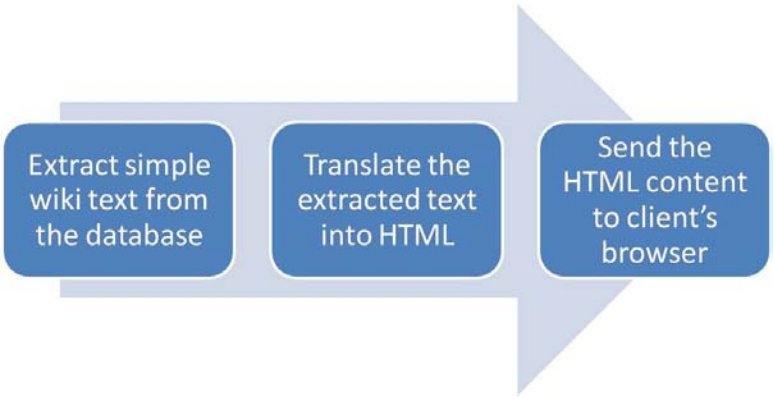


Figure 2.2: Work Flow of Wiki Engine

server-side scripting language and MySQL as the database engine. PHP reads sim- ple text from the MySQL engine and then checks this text line by line and replaces formatting rules of Wiki text by HTML constructs.

2.1.2 Structure of a Wiki page

Since we want to query Wiki pages, we need to find some sort of structure in them. It is not so easy to identify structure in Wiki pages as anyone and everyone can edit it in a way they like it and hence cannot be constrained in a schema. Now without the knowledge of schema, one cannot write a database query [12]. In some cases it so happens that structure in a data exists but it is not directly evident. It has to be extracted out from the data itself. This type of data is known as semi-structured data or unstructured data [13]. Wiki pages are also semi-structured or unstructured.

We looked at lots of Wiki pages and came to the conclusion that any Wiki page consists of three things - Text, Sections and Links.

• Text: Text refers to the information a Wiki page contains.
• Section: The information of a Wiki page can be divided into different categories.

These categories have been termed as Sections. One way of categorizing is by headings a Wiki page contains. Each heading can be considered as a different section.

• Links: A very important property of Wiki is that Wiki pages are connected to each other. Hence a Wiki page also contains links to other Wiki pages so that a user can navigate from one page to another.

Semi-structured data can be best represented in a graph-like or tree-like structure [12]. Hence, it is a good idea to also represent Wiki page in a tree-like or graph-like structure. We can treat Wiki page as a Extensible Markup Language (XML) document [14] as XML is a widely used way to represent semi-structured data and XML docu- ments also form a tree structure. Since now a Wiki page can be treated as a XML object, we have made a Document Tree Definition(DTD) [15] for the same. Each Wiki page contains a root element. Now this root element can have text or sections or links. Each section, in turn, again may contain text or sections or links. Each section has an

attribute named title. Thus we see that structure of Wiki is recursive in nature as sections can also have sections. The DTD for Wiki pages is as follows -

```
<!DOCTYPE root
 [
   <!ELEMENT root (section+)>
   <!ELEMENT section (#PCDATA | link | section )*>
   <!ELEMENT link (#PCDATA)>
   <!ATTLIST root
        title CDATA #REQUIRED>
   <!ATTLIST section
        title CDATA #REQUIRED>
   <!ATTLIST link
        target CDATA #REQUIRED>
```

Let us now have a look at a sample Wiki page and its corresponding XML Repre- sentation. Figure 2.3 shows a sample Wiki page -

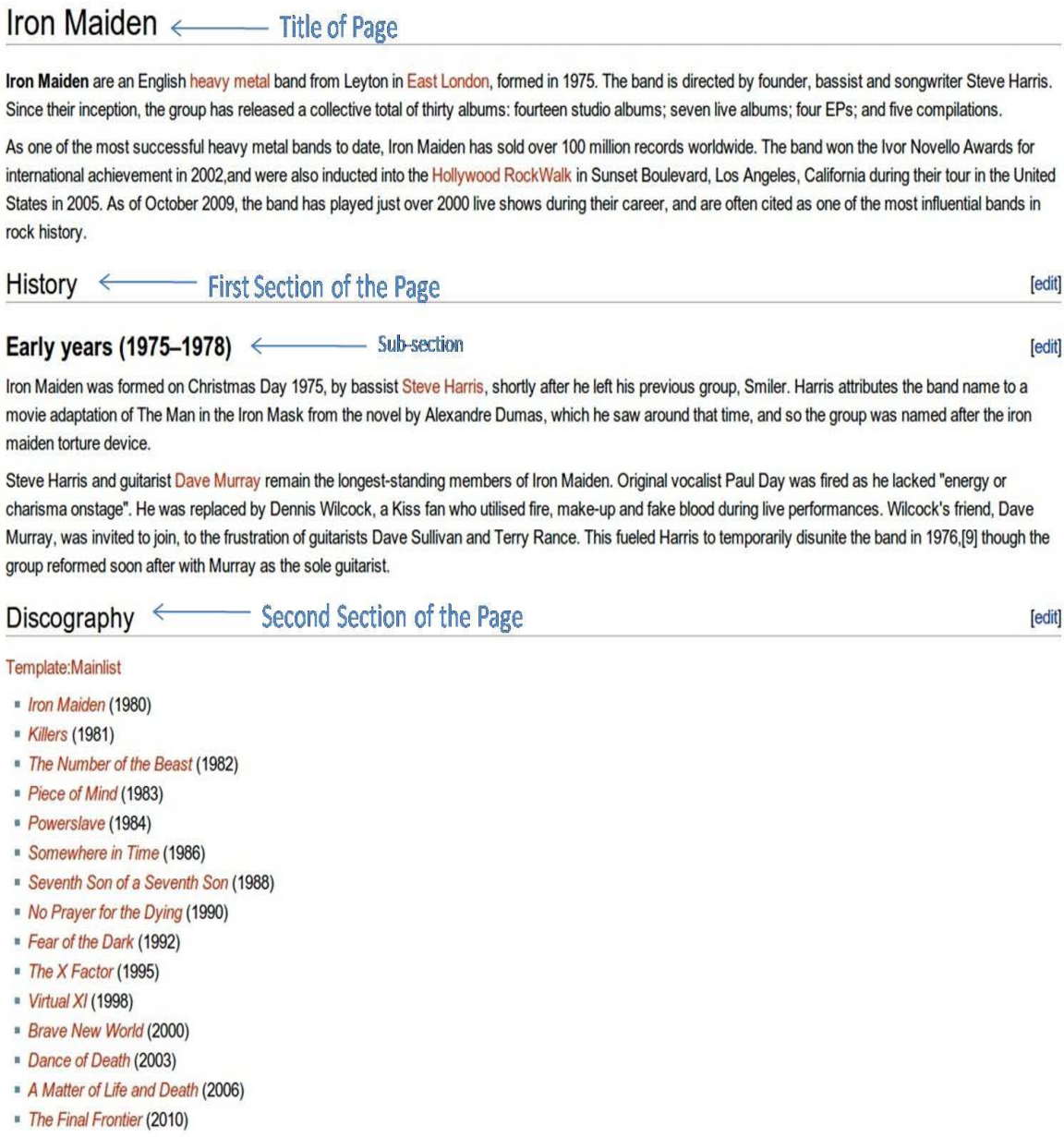


Figure 2.3: Sample Wiki Page

Figure 2.4 is the XML representation of Wiki page shown in figure 2.3 -

```
−<root title="Iron Maiden">
  Iron Maiden are an English
  <link target="http://en.wikipedia.org/wiki/Heavy_metal_music"> heavy metal</link>
  band from Leyton in
  <link target="http://en.wikipedia.org/wiki/East_London,_England">East London</link>
  , formed in 1975. The band is directed by founder, bassist and songwriter Steve Harris. Since their inception, the group has released a collective total of thirty albums: fourteen studio albums, seven live albums, four EPs and five compilations. As one of the most successful heavy metal bands to date, Iron Maiden has sold over 100 million records worldwide. The band won the Ivor Novello Awards for international achievement in 2002,and were also inducted into the
  <link target="http://en.wikipedia.org/wiki/Guitar_Center#Hollywood.27s_RockWalk">Hollywood RockWalk</link>
  in Sunset Boulevard, Los Angeles, California during their tour in the United States in 2005. As of October 2009, the band has played just over 2000 live shows during their career, and are often cited as one of the most influential bands in rock history.
 −<section title="History">
  −<section title="Early Years">
    Iron Maiden was formed on Christmas Day 1975, by bassist
    <link target="http://en.wikipedia.org/wiki/Steve_Harris_(musician)">Steve Harris</link>
    , shortly after he left his previous group, Smiler. Harris attributes the band name to a movie adaptation of The Man in the Iron Mask from the novel by Alexandre Dumas, which he saw around that time, and so the group was named after the iron maiden torture device. Steve Harris and guitarist
    <link target="http://en.wikipedia.org/wiki/Dave_Murray_(musician)">Dave Murray</link>
    remain the longest-standing members of Iron Maiden. Original vocalist Paul Day was fired as he lacked "energy or charisma onstage". He was replaced by Dennis Wilcock, a Kiss fan who utilised fire, make-up and fake blood during live performances. Wilcock's friend, Dave Murray, was invited to join, to the frustration of guitarists Dave Sullivan and Terry Rance. This fueled Harris to temporarily disunite the band in 1976,[9] though the group reformed soon after with Murray as the sole guitarist.
   </section>
  </section>
 −<section title="Discography">
    <link target="http://en.wikipedia.org/wiki/Iron_Maiden_(album)"> Iron Maiden (1980)</link>
    <link target="http://en.wikipedia.org/wiki/Killers_(album)">Killers (1981)</link>
    <link target="http://en.wikipedia.org/wiki/The_Number_of_the_Beast_(album)">The Number of the Beast (1982)</link>
    <link target="http://en.wikipedia.org/wiki/Piece_of_Mind">Piece of Mind (1983)</link>
    <link target="http://en.wikipedia.org/wiki/Powerslave">Powerslave (1984)</link>
    <link target="http://en.wikipedia.org/wiki/Somewhere_in_Time_(album)">Somewhere in Time (1986)</link>
    <link target="http://en.wikipedia.org/wiki/Seventh_Son_of_a_Seventh_Son">Seventh Son of a Seventh Son (1988)</link>
    <link target="http://en.wikipedia.org/wiki/No_Prayer_for_the_Dying">No Prayer for the Dying (1990)</link>
    <link target="http://en.wikipedia.org/wiki/Fear_of_the_Dark_(album)">Fear of the Dark (1992)</link>
    <link target="http://en.wikipedia.org/wiki/The_X_Factor_(album)">The X Factor (1995)</link>
    <link target="http://en.wikipedia.org/wiki/Virtual_XI">Virtual XI (1998)</link>
    <link target="http://en.wikipedia.org/wiki/Brave_New_World_(Iron_Maiden_album)">Brave New World (2000)</link>
    <link target="http://en.wikipedia.org/wiki/Dance_of_Death_(album)">Dance of Death (2003)</link>
    <link target="http://en.wikipedia.org/wiki/A_Matter_of_Life_and_Death_(album)">A Matter of Life and Death (2006)</link>
    <link target="http://en.wikipedia.org/wiki/The_Final_Frontier">The Final Frontier (2010)</link>
  </section>
</root>
```

Figure 2.4: XML Representation

2.2 Wiki engine as a component in Technology Stack

This section discusses how and where Wiki fits into the technology stack and what will be the architecture for web applications which are running over Wiki platform.

2.2.1 Architecture - Web Applications

The architecture for web applications running over Wiki engine is still client-server model but with a one change. Now a Wiki Engine also sits along with a Database Engine. Wiki Engine communicates with application server and transfers required data to it which is then sent to the client's browser. Figure 2.5 demonstrates this architecture

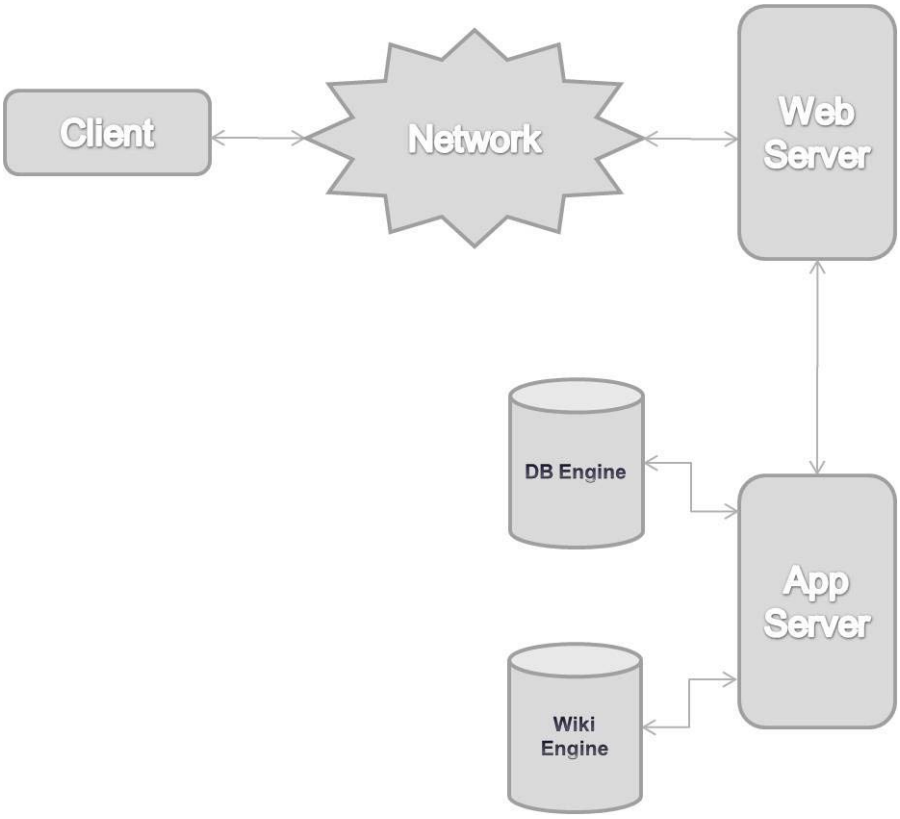


Figure 2.5: Architecture for web applications running over Wiki

Architecture of applications is same irrespective of whether they have been built using JSP + WQL or only WSL. However, the work flow is a bit different in both the cases. Let us discuss the workflow in both the cases.

2.2.2 JSP + WQL - Workflow

The purpose of WQL is to allow users to create dynamic web pages with data stored in a Wiki engine. To make use of Wiki engine, web applications have to be built using a server side scripting language - JSP, and Wiki Query Language (WQL). Queries written in WQL can be embedded in JSP in the same way SQL queries are embedded now. So essentially the work flow is like this -

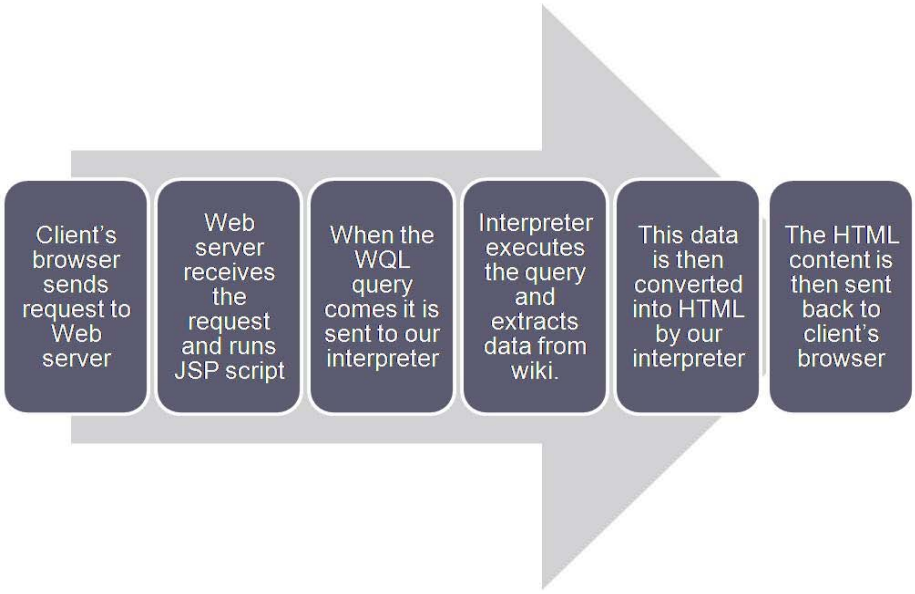


Figure 2.6: Workflow of web applications built using JSP & WQL

• Client sends a request for a web page to web server.

• Web server reads the request and runs the server side script (JSP)

• The script then executes the code and when the WQL query comes it sends it to the WQL Interpreter.

• WQL Interpreter then returns the required data, first extracting from Wiki and then converting the same into HTML, to the demon executing the JSP script.

• Web server sends back the generated HTML content to the client which is dis- played in his/her browser.

2.2.3 WSL - Workflow

WSL is a server-side scripting language using which users can create web applications by extracting data from the Wiki. In this section we have discussed workflow of applications which are built using WSL. Figure 2.7 shows this workflow -

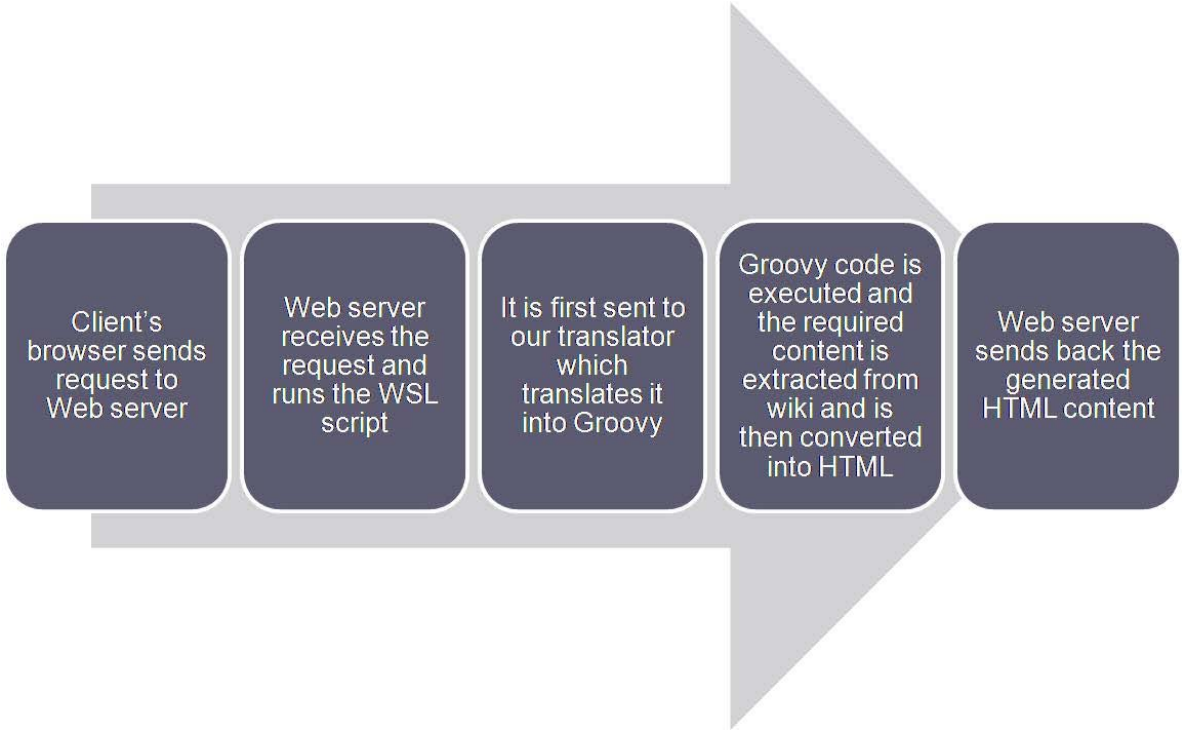


Figure 2.7: Workflow of web applications built using WSL

• Client sends a request for a web page to web server.

• Web server reads the request and runs the WSL script

• The code is sent to our translator which translates it into Groovy

• Groovy compiler then executes the code and the data is then extracted from Wiki. This is then converted into HTML.

• Web server sends back the generated HTML content to the client which is finally displayed in his/her browser.

2.3 Extracting data from Wiki

As has been discussed in Section 2.1 Wiki is a network of hypertext documents inter- linked with each other. We have treated this network as a graph where each Wiki page is a vertex and the links act as edges and each vertex, in turn, has a tree-like structure. A better way to visualize it is to treat it as a map of city. Now initially user is at some location in the map

which is the start location and the required information is at some other location which is the final location. To extract the required information, the user needs to go from the start location to the final location. This path/route, from start to final location, is specified by writing queries.
Referring to the Document Tree Definition(DTD) described in section 2.1.2 for Wiki pages, information available on a Wiki page is divided into various sections. So, every time a user will want to either fetch the complete Wiki page or some sections of a Wiki page. To store and return the data from a Wiki page, a suitable data-structure is required. The data-structure used by us is a List of Nodes. Now the question arises what a Node is? The following section discusses Node in detail.

2.3.1 Node

A Node is an object which it used to store a Wiki page or a section of a Wiki page. Whenever a Wiki page or a section of page is requested, we parse it and store it in a Node. It has following three fields -

• Data: It contains the text of an entire Wiki page or a section of Wiki page.
• List of Sections: It is a simple list which contains the names of all the sections (or subsections) a Wiki page (or a section of Wiki page) contains.
• List of Links: It is also a list containing names and addresses of all the interlinks that a Wiki page (or a section of Wiki page) has. Implementation of Node has been done in Java. We have written a Node class in Java and whenever a Node is required to store data, an instance of Node class is created.

```
public abstract class Node // NODE Class

{   public List<Links> links; // List of Links
    public List<String> sections; // List of Sections String
    text; // Data field of Node Hashtable<String, String>
    hashlinks; Hashtable<String, Integer> sectionno;
    public List<String> allsections;
    String description;
    String base;
    public String getText()
    {       return text;
    }
    public List<Links> getLinks()
    {       return links;
    }
    public List<String> getSections()
    {       return sections;
    }
    public String getTitle()
    {       return title;
    }
```

```
}
public class Links
{     private String text;
      private String link;

      public Links(String link, String text)
      {       this.text = text;
              this.link = link;
      }
      public String getLink()
      {       return link;
      }
      public String getText()
      {        return text;
      }
}
```

Figure 2.8 shows the visualisation of Node -

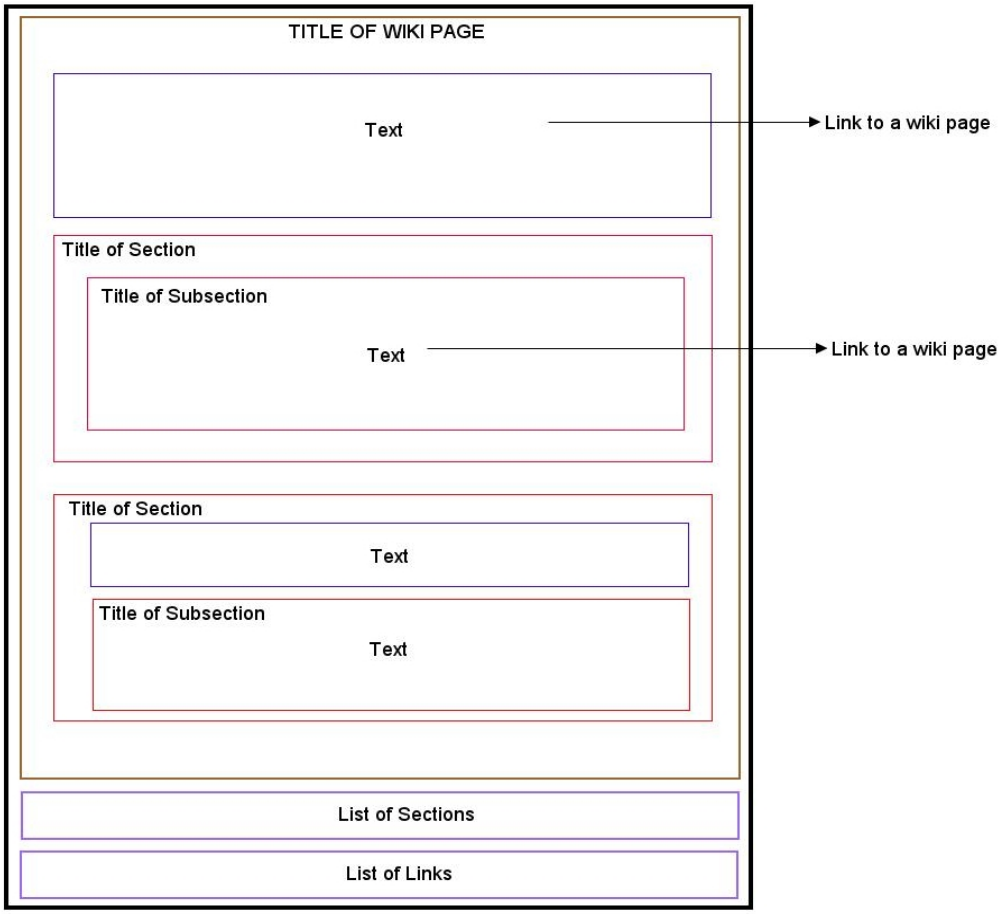


Figure 2.8: Visualisation of Node

2.3.2 Algorithm

We have built an interpreter which takes a query string as an input and returns a list of Nodes containing the required data. We have named it Wiki Interpreter. Before discussing the algorithm let us look at a couple of underlying assumptions that we have made while implementing it. One of them is that the title of every Wiki page is unique. No two Wiki pages can have the same title and most of the popular Wiki engines do not allow users to create two Wiki pages with same title. Another one is that a Wiki engine has a search feature which allows users to search for a Wiki page. User only specifies the title of the Wiki page that he wants to see. The Wiki interpreter then makes use of the Wiki search feature and fetches the required page.

Using Wiki Query Language, developers can construct queries by wrapping actions around one another. The list will keep on changing and at the end the developer will have the required information. Let us now look at how all this works. Initially, Wiki interpreter receives query string either from a server side scripting language(JSP or WSL). After receiving it, the interpreter parses the string to know what needs to be done. Every time an action is performed a list of Nodes is created and the subsequent actions are then performed on the entire list. Now, when Wiki interpreter fetches the data from Wiki it is in Wiki text format and therefore, the list will also have the data in Wiki format only. However, for web application we need to return data in HTML format so that it can be displayed inside a web browser. So for that we have written a parser which converts Wiki text into HTML. When the final list is generated after executing the entire query, it is passed to this Wiki-to-HTML parser and the text contained in the list gets converted into HTML. This list is then returned to the daemon executing server-side script(JSP or WSL). Figure 2.9 explains this algorithm.

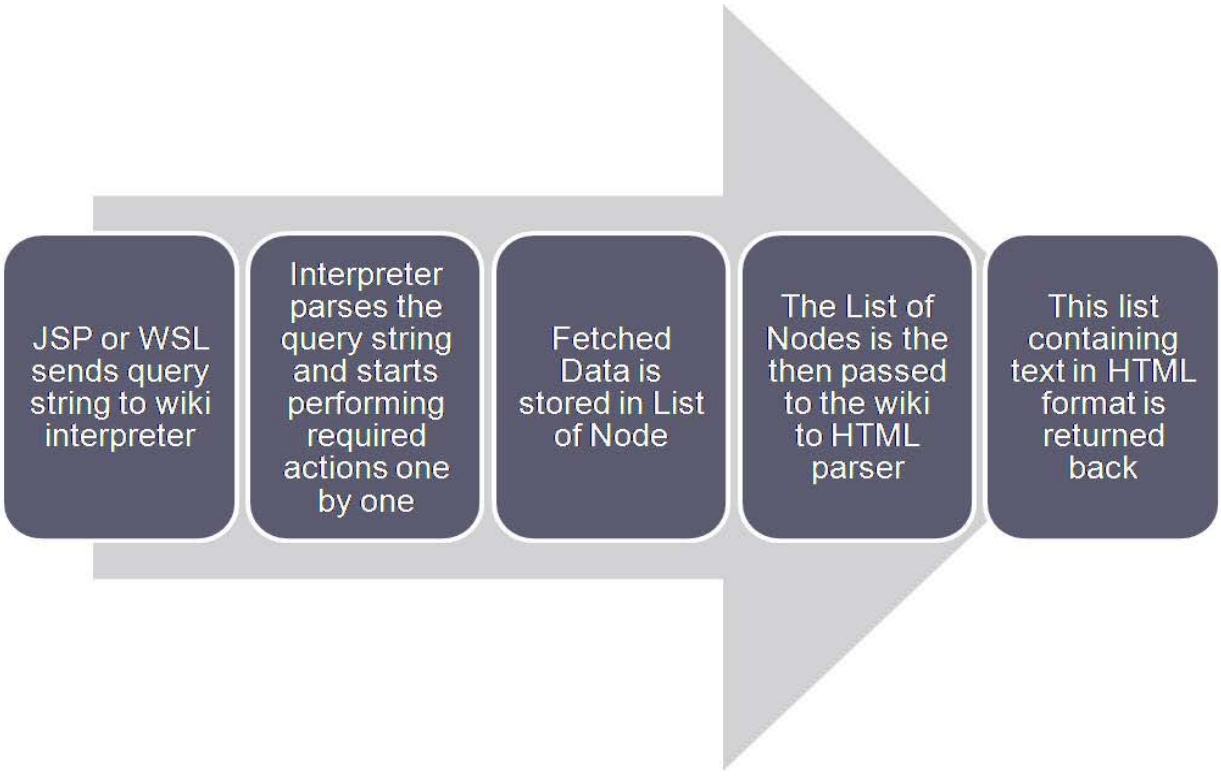


Figure 2.9: Algorithm to query a Wiki engine

2.4 Language Features

Following are the features, WQL and WSL provides to their users -

• Easy to get it up and running: We are providing an installer for WSL, which will install the complete stack on user's machine. Users only need to have JRE installed on their machines apriori and they are ready to go.

• Shallow Learning Curve: The syntax of WQL/WSL is fairly intuitive. Queries are constructed by wrapping actions one round another. Moreover, writing programs in WSL is similar to writing programs in PHP. Therefore not much learning is required to start building applications in WSL/WQL.

One very important design pattern that we have incorporated in our language is the Decorator Pattern. The Decorator pattern is an object-oriented programming design feature using which one can add new functionalities to existing objects dynamically. This is done by implementing a new decorator class, also known as "wrapper" class, which wraps the original class. This gives lot of flexibility as one can change what decorator does at run time. One can wrap a component with any number of decorators. Java.io package is largely based on Decorator Pattern [16]. Decorator Pattern provides following key features -

• Composition: Composition is an integral part of our language. Users can write large complex queries by wrapping small queries one around another.

• Modularity: A system is said to be modular if it can be broken into components. Our implementation is also modular and hence is easier to understand in terms of different components.

• Scalability: Because of Decorator Pattern new functionalities can be added by writing new code rather than making changes in the existing code.

2.5 Implementation Details

The implementation has been done in Java and Groovy. We chose object oriented programming language because we wanted to incorporate the decorator pattern in our design. We have used Another Tool for Language Representation (ANTLR) for parsing query string. ANTLR is a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing actions in a variety of target languages. From a formal grammar, ANTLR generates a program that determines whether sentences conform to that language. By adding code snippets to the grammar, the recognizer becomes a translator or interpreter [17]. So
we wrote the grammar for our language in ANTLR and it automatically generated the lexer and parser for us in Java.
We have made full use of object-oriented programming and therefore, our code is divided into various useful classes. Figure 2.10 represents class diagram of our implementation.

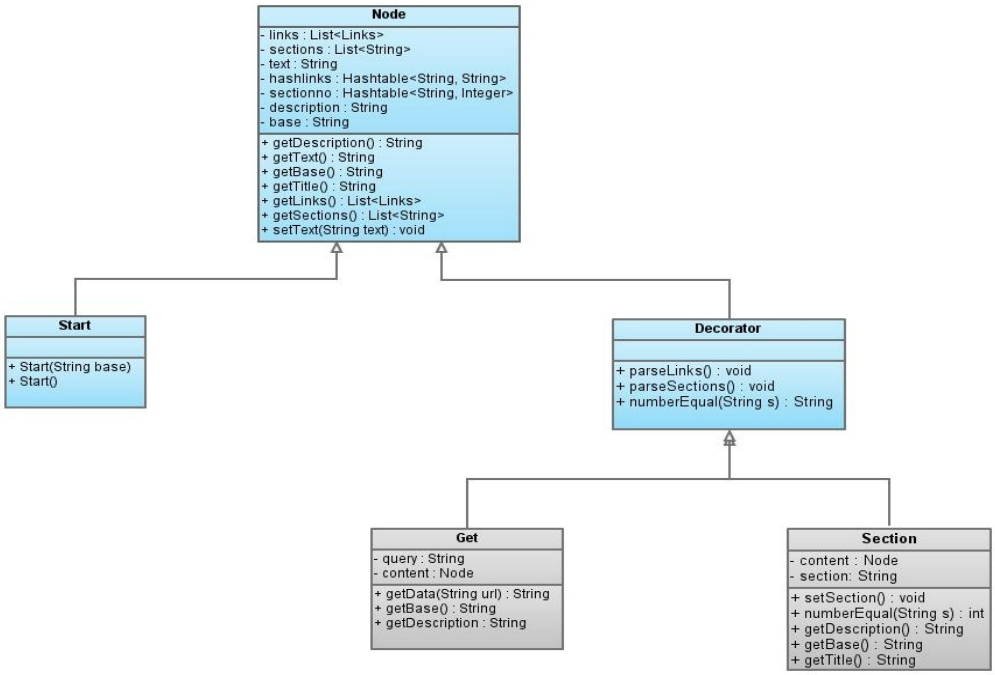


Figure 2.10: Class Diagram

Other than the classes shown in figure 2.10, we have written some other helper classes as well which are as follows -

• Wiki.java: This is Wiki text to HTML parser class and converts given text in Wiki format into HTML code.

• ModifyWiki.java: This class is used to implement functions which are used for editing a Wiki.

• WikiURL.java: Its sole purpose is to locate pages which can be used by Decorator class.

• wikiDetails.java: This page is used to get details of a Wiki page such as starttime, edittime etc.

• Links.java: This class was written to define a ne object type which has been used in Node class.

The grammar that we have written in ANTLR is as follows -

```
  prog returns [List<Node> ret]
  e=stat (NEWLINE (e=stat)?)*
stat    returns [List<Node> ret]
page
| ID '=' e=page
| NEWLINE
page  returns [List<Node> ret]
GETLINKS csi '.' e=page
GETLINKS f=range '.' e=page
```

```
GETLINKS css '.' e=page
GETSECTION css '.' e=page
GETSECTION csi '.'  e=page
GETSECTION f=range '.' e=page
GET css csi?
'(' e=page ')'
|
I
D
|
CREATEPAGE title=STRING content=STRING
|
EDITPAGE title=STRING content=STRING
|
EDITPAGE e=page content=STRING
|
APPENDPAGE title=STRING content=STRING
|
APPENDPAGE e=page content=STRING
|
EDITSECTION title=STRING number=sectionno[$title.text] content=STRING
|
EDITSECTION e=page content=STRING
|
APPENDSECTION title=STRING number=sectionno[$title.text] content=STRING
|
APPENDSECTION e=page content=STRING
|
ADDSECTION title=STRING sectiontitle=STRING content=STRING
|
ADDSECTION e=page sectiontitle=STRING content=STRING
|
ADDSECTIONAFTER title=STRING number=sectionno[$title.text]
sectiontitle=STRING content=STRING
|
ADDSECTIONAFTER e=page sectiontitle=STRING content=STRING
|

ADDSECTIONBEFORE title=STRING number=sectionno[$title.text]
 sectiontitle=STRING content=STRING
|
ADDSECTIONBEFORE e=page sectiontitle=STRING content=STRING
;
range returns [int lower, int upper]
```

```
: '['       a=INT? ':' b=INT? ']'
| 'all'
;
sectionno [String title]
returns[int value]:
INT
|
STRI
NG
;
css   returns [String arg]
: '[' e=STRING    (',' e=STRING )* ']'
|
e=STRING
;
csi returns [List<Integer> args]
: '[' e=INT      (',' e=INT )*']'
|
e=INT
;
GETLIN
KS
: 'getlinks' | 'getlink';
GETSECTION
: 'getsections' | 'getsection';
GET : 'get' | 'Get'| 'getpages' | 'getpage';
CREATEPAGE
: 'createPage' | 'createpage';
APPENDPAGE
: 'appendpage' | 'appendPage';
EDITPAGE: 'editpage' | 'editPage';
EDITSECTION
: 'editsection' | 'editSection';
APPENDSECTION
: 'appendsection'|'appendSection';
ADDSECTION
: 'addSection' | 'addsection';
ADDSECTIONAFTER
: 'addsectionafter' | 'addSectionAfter'
;
ADDSECTIONBE
FORE
: 'addsectionbefore' | 'addSectionBefore'
;
```

```
OR: 'or'
| 'OR'
;
AND : 'AND' | 'and';
ID  : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
    ;
INT : '0'..'9'+
    ;
NEWLINE:'\r'? '\n' | ';' ;
STRING
    :  '"' ( ESC_SEQ | ~('\\'|'"') )* '"'
    ;
COMM
ENT
    :   '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
    |   '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
    ;
WS :    ( ' '
        | '\t'
        | '\r'
        | '\n'
      ) {$channel=HIDDEN;}
;
fragment
HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;
fragment
ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
    |   UNICODE_ESC
    |   OCTAL_ESC
    ;
fragment
OCTAL_
ESC
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;
fragment
UNICODE_
ESC
    :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT ;
```

3. Syntax and Semantics

This chapter discusses the syntax of WQL and WSL. First we will discuss the syntax of WQL and after that we will look at WSL.

3.1 WQL - Syntax

Similar to SQL, WQL can also be divided into the Data Manipulation Language(DML) and the Data Definition Language(DDL). DML part of WQL includes -

- getpage
- getlinks
- appendpage
- appendsection

DDL part of WQL includes -

- createpage
- editpage
- editsection
- addsection
- addsectionafter
- addsectionbefore

Before discussing all the above constructs in detail, let us first have a look at some important terms that we have used in our grammar -

• css: css refers to comma separated strings and is explained in figure 3.1.

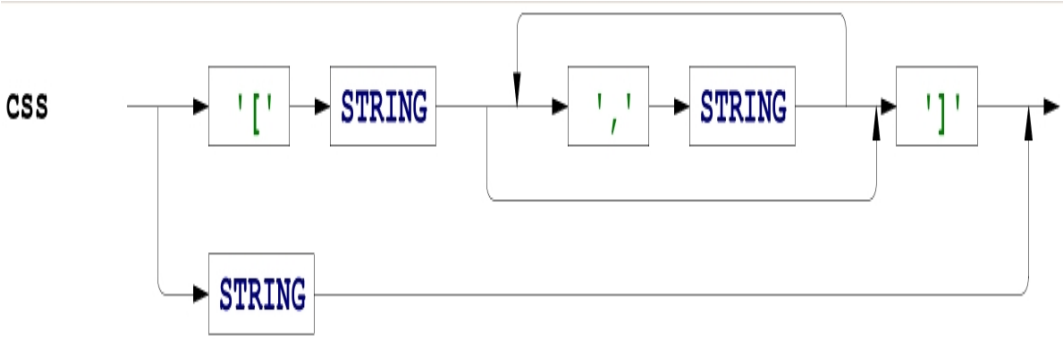


Figure 3.1: css

• csi: csi refers to comma separated integers and is explained in figure 3.2.

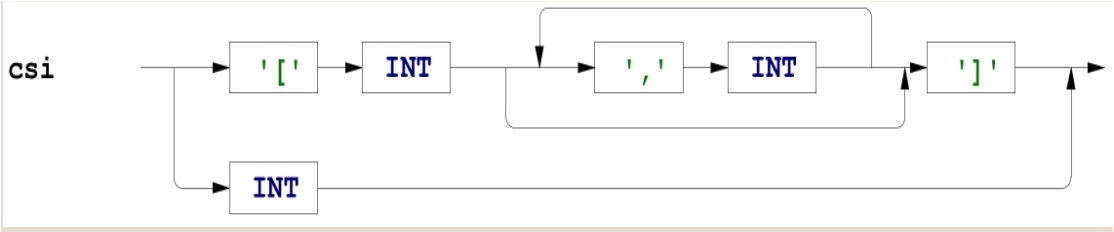


Figure 3.2: csi

• range: range refers to a range of integers and is explained in figure 3.3.

• seectionno: It is explained in figure 3.4.

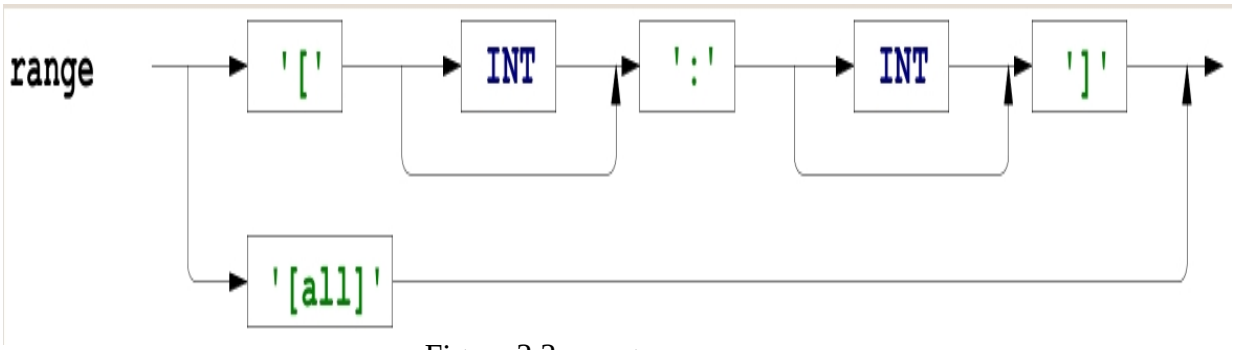


Figure 3.3: range

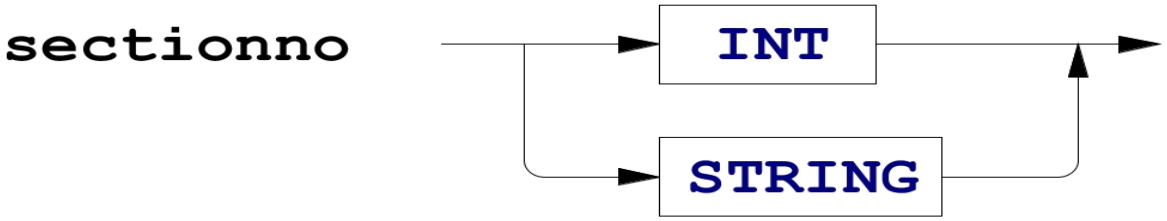


Figure 3.4: sectionno

3.1.1 Getpage

getpage “title ofpage”
getpage [“title 1”, “title 2”,.....,“title n”]

Table 3.1: getpage

getpage downloads the Wiki page whose title is same as what has been mentioned in the query. Since title of every page is unique, there is no ambiguity in deciding which page to fetch. This is done by making use of search feature of Wiki engine. getpage construct returns a single Node or a list of Nodes depending on whether single or multiple pages are downloaded.

When a getpage query is executed the required Wiki page is downloaded and its

text, which is in Wiki syntax, is put in the data field of Node object. This Wiki text is then parsed and the names of all the sections are put in the list of sections of Node object and similarly names and addresses of links are placed in the list of links of Node object. getpage is always the first step of every query. getpage can also be viewed in terms of XML. So, effectively, getpage returns the root elements of all the XML documents which users specify to download.

As has been discussed in section 2.1, one important feature that Wiki provides is that Wiki engine keeps versions of pages which allows users to roll back to the previous version. In our language we have also provided a feature to allow users to download previous versions of a Wiki page. This can be done by using the getpage construct in the following manner -

getpage (“title of page”)(integer)
getpage [“title 1”, “title 2”,....., “title n”][integer 1, integer 2,....., integer n]

Table 3.2: getpage with version support

The integer in the above query specifies which version user wants to download. Integer value equal to 1 refers to the current version of Wiki page.

3.1.2 Operator

’.’ Operator is our composition operator using which users can compose queries. get- secion and getlink need a list of Nodes upon which they can operate. So, before using any of these two, users need to do a getpage first. Doing getpage will create a list of Nodes, after which getsection and getlink can be wrapped around using ’.’ operator.

3.1.3 Getsection

Figure 3.5 shows how getsection should be used -

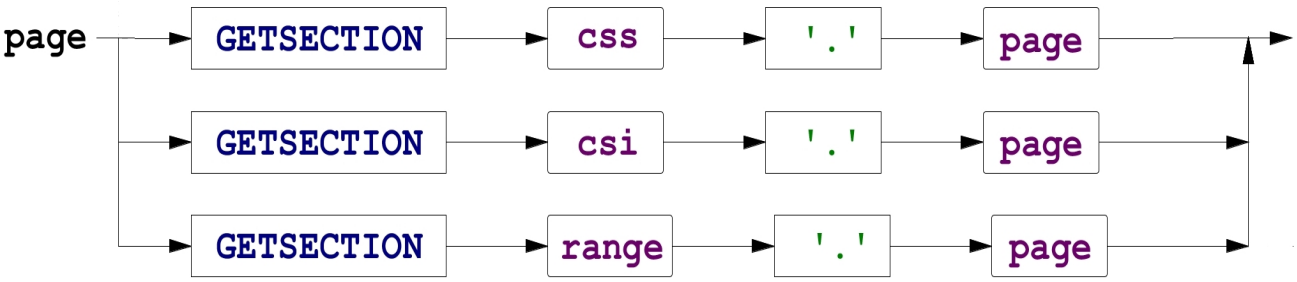


Figure 3.5: getsection

(getsection “title of section”) . <list of nodes>
(getsection [“title 1”, “title 2”,.....,“title n”]) . <list of nodes>
(getsection all) . <list of nodes>
(getsection 4) . <list of nodes>
(getsection [2,7,9]) . <list of nodes>
(getsection [3:9]) . <list of nodes>

Table 3.3: Examples of getsection

The function of getsection is to fetch a section or multiple sections from a given list of Nodes.getsection always operates upon a given a list of Nodes and fetches the desired section. Let us assume that there is a list with only one Node. When getsection query is executed, it looks for the section, whose title is same as what is mentioned in the query, in the Node. If it is there, it replaces the text field of the Node object with Wiki text of desired section. Similarly, list of sections and list of Nodes are also updated. So, essentially the Node object now contains section element of the XML document which was initially present in the Node. If multiple sections are downloaded, a Node for each section is created. If the list contains more than one Node, the same action is carried out for every Node object. Using getsection user moves down by one level in the wiki-tree. At the end, the updated list of Nodes is returned.

3.1.4 getlinks

Figure 3.6 shows how getlinks should be used -

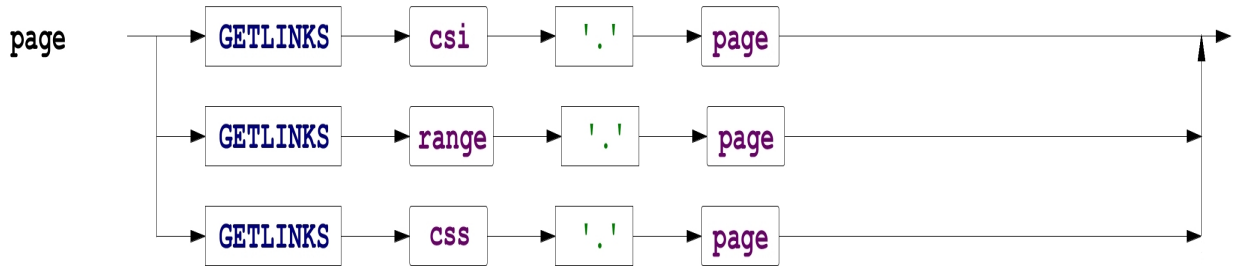


Figure 3.6: getlinks

(getlinks 4) . <list of nodes>
(getlinks [2,7,9]) . <list of nodes>
(getlinks [3:9]) . <list of nodes>
(getlinks all) . <list of nodes>
(getlinks “author”) . <list of nodes>
(getlinks [“author”, “Cape Town”,.....,“Mahatama Gandhi”]) . <list of nodes>

Table 3.4: Examples of getlinks

getlinks is used to fetch the page where a link points to. Using getlinks user moves from one vertex to another in the graph. getlinks fetches the entire Wiki page and hence creates a list of Nodes with each Node having the root element of XML representation of Wiki page.

3.1.5 createpage

createpage is used to create a new Wiki page and add it to the Wiki database. The main utility of createpage comes when you want to create pages with a fixed template.

In such a case you can create a form which users can use to create new Wiki pages and in that form and in that form the template can be fixed.

createpage “title of page” “text in wiki format”

Table 3.5: createpage

3.1.6 editpage

editpage “title of page” “text in wiki format”
editpage <list of nodes> “text in wiki format”

Table 3.6: editpage

editpage is used to edit a Wiki page. It replaces the original text present in the Wiki page with the one specified inside the query. editpage also takes a list of Nodes as an input, so if a user wants to make the same changes in a number of Wiki pages, he/she can do that as well.

3.1.7 appendpage

appendpage is used to add data at the end of one or more Wiki pages. Table 3.7 shows the syntax of appendpage.

appendpage “title of page” “text in wiki format”
appendpage <list of nodes> “text in wiki format”

Table 3.7: appendpage

3.1.8 editsection

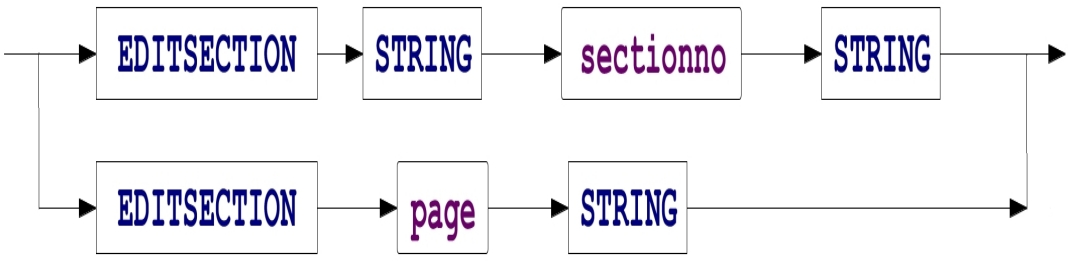


Figure 3.7: editsection

editsection “title of page” “title of section” “text you want to add”
editsection “title of page” 3 “text you want to add”
editsection <list of nodes> “text you want to add”

Table 3.8: editsection

editsection is used to edit a section of a Wiki page. It replaces the original text present in the section with the one specified inside the query. Moreover, editsection can also take a list of Nodes as an input, so if a user wants to make same changes in a number of sections, he/she can do that.

3.1.9 appendsection

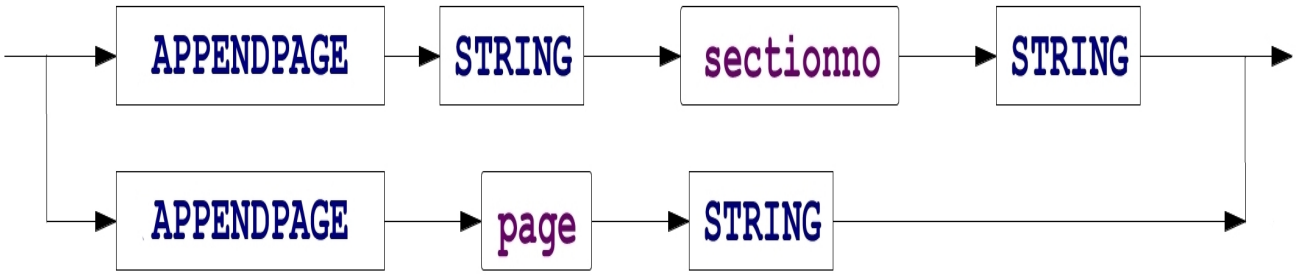


Figure 3.8: appendsection

appendsection “title of page” “title of section” “text you want to add”
appendsection “title of page” 3 “text you want to add”
appendsection <list of nodes> “text you want to add”

Table 3.9: appendsection

appendsection is used to add some text at the end of a section. Again like editsection, appendsection also takes a list of nodes as an input.

3.1.10 addsection

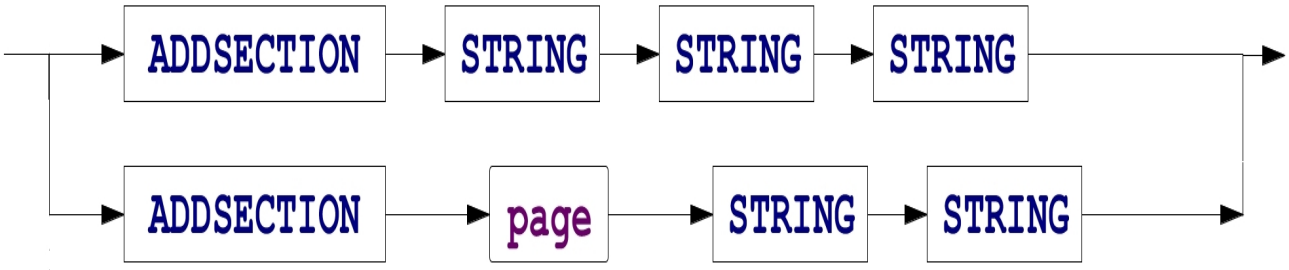


Figure 3.9: addsection

addsection “title of page” “title of section” “text you want to add”
addsection <list of nodes> “text you want to add”

Table 3.10: addsection

addsection is used to add a new section in a Wiki page. The new section is added at the end of a Wiki page. Using addsection one can add the same section in more than one Wiki page.

3.1.11 addsectionafter

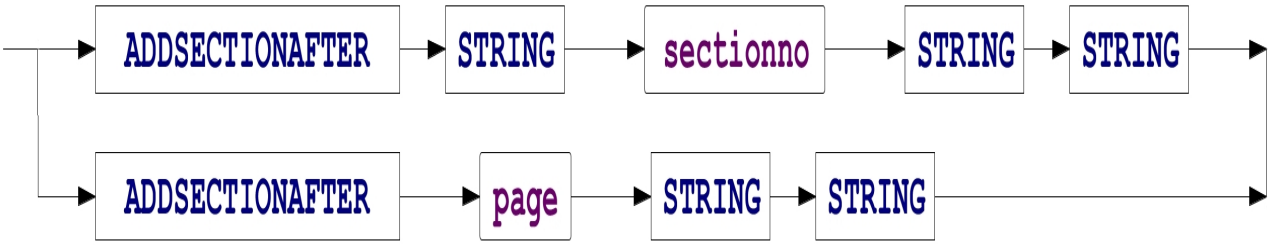


Figure 3.10: addsectionafter

addsectionafter “title of page” “title of existing section” “title of new section”
“text you want to add”
addsectionafter “title of page” 3 “title of new section” “text you want to add”
addsectionafter <list of nodes> “text you want to add”

Table 3.11: addsectionafter

addsectionafter is used to add a new section after an existing section in a Wiki page.

3.1.12 addsectionbefore

addsectionbefore is used to add a new section before an existing section in a Wiki page.

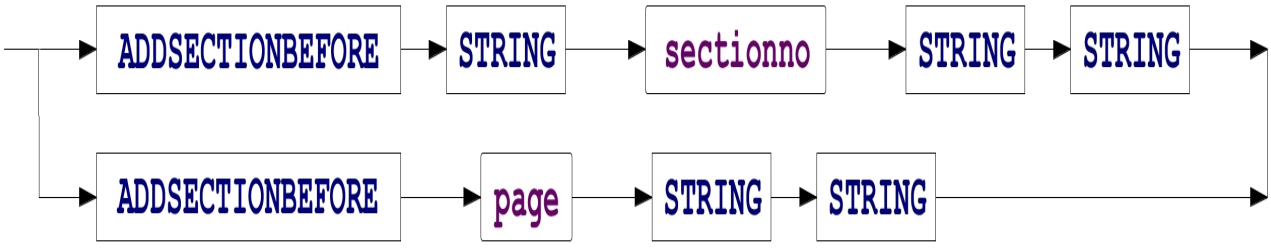


Figure 3.11: addsectionbefore

addsectionbefore “title of page” “title of existing section” “title of new section”
“text you want to add”
addsectionbefore “title of page” 3 “title of new section” “text you want to add”
addsectionbefore <list of nodes> “text you want to add”

Table 3.12: addsectionbefore

3.1.13 = Operator

'=' is an assignment operator. Users can assign queries to a variable and can then use this variable to construct other queries. For e.g. -
f = getpage "Taj Mahal"
g = (getsection "History"). (f) OR g = (getsection "History"). f
This is now same as (getsection "History"). (getpage "TajMahal")

3.1.14 listsections()

listsections(Node object)

Table 3.13: listsections

listsections() returns the list of sections field of a Node object. It can be used in scenarios where user wants to play with the names of sections a Wiki page contains.

3.1.15 listlinks()

listlinks(Node object)

Table 3.14: listlinks

listlinks() takes as arguement a Node and returns the list of links field of that Node. The links themselves are stored as an array containing two elements with the first one being the address of location where the link points to and the second one being the text.

3.1.16 setbase()

setbase() is used to tell the location of Wiki engine to the Wiki interpreter. For e.g. - setbase("URL pointing to location of wiki that will be used")
User should first set the location of the Wiki engine using setbase() before executing any query.

3.2 WSL - Syntax

Wiki Scripting Language(WSL) is a server-side scripting language which can communicate with Wiki interpreter and execute WQL queries. WSL code can be interspersed within the HTML code between <wsl> and </wsl> tags. The extenstion of a file containing wsl code is ".wsl". Moreover, within wsl tags user can write any Groovy code. A typical wsl file will look as follows -

```
<html>
<head>This is a test page</head>
<body>
```

```
<h1>Adminstrator</h1>
<wsl>
// ADD WSL CODE HERE
</wsl>
</body>
</html>
```

3.2.1 query()

query() function is used to execute WQL queries. The query string, in WQL format, can be passed as an argument which is then sent to the Wiki interpreter and the required data is fetched. For e.g. -

query(“““ getpage “CSE Department” ”””)

The triple quotes ensure that the user doe not need to escape the ” characters everytime they are used inside the query.

3.2.2 Data-types

WSL is a dynamically typed language. So the result of a query, which is generally a list of Nodes, can be assigned to any variable. A user, therefore, has to simply write -

v = query(“““ getpage “CSE Department” ”””) //v will contain a list of Nodes.
Since it is a dynamically typed language, following piece of code will work.

```
<wsl>
a = "Hello"
a = 5
</wsl>
```

3.2.3 echo()

echo() function is used to print something on the standard output. A string or a variable can be passed as an argument. It works as follows -

• echo(“Show this on display”)
• echo(v)

When we use echo () for a list of Nodes, it prints out only the data field of Nodes.

3.2.4 Other Useful Constructs

As we have already mentioned inside <wsl> tags user can write Groovy code as well, therefore, user can make use of conditional statements, loops, string operations, arrays and

other such complex constructs to build web applications. Form handling can also be handled by writing code in Groovy.

3.3 Sample WSL Code

A sample .wsl file looks as follows -

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CSE Department</title>
</head>
<body>
<wsl> setbase("http://172.27.22.35/mediawiki/index.php");
a = query(""" getpage "CSE Department" """)
echo(a)
</wsl>
<br>
<hr>
<footer>
<font size=2>This book has been made using Wiki Query Language</font>
</footer>
</body>
</html>
```

4. Sample Applications

This chapter discusses about the four sample applications which we have made to demonstrate how Wiki can be used as a component in the technology stack to make web applications. First two applications have been built using WQL along with JSP and the other two have been built using WSL.

4.1 Departmental Website

We have built a website for our department using JSP and WQL. Sometimes it happens that people do not update their webpages from time to time and webpages contain outdated information.

4.1.1 Wiki Design

The design of Wiki is simple in this case. Each faculty member has a Wiki page of its own which contains general information about the faculty. XML representation of one such page is as follows -

```
<root title = "Tvp Home">
<section title = "Designation">
      SOME TEXT HERE
</section>
<section title = "Contact Info">
      SOME TEXT HERE
</section>
<section title = "Area of Interest">
      SOME TEXT HERE
</section>
<section title = "Teaching">
      SOME TEXT HERE
</section>
</root>
```

The above page is shown in Figure 4.1. Other than the above mentioned Wiki page each faculty has another Wiki page which contains the list of publications by the faculty member. There are two index pages which lists the names of professors and contains links to their individual Wiki pages.

4.1.2 Website

The departmental website contains page for each faculty member which is generated by extracting data from the Wiki page of faculty member. If faculty wants to make some changes, he/she only needs to edit his/her Wiki page and the web-page will get modified automatically. It is shown in Figure 4.3. This page contains link to a web- page which displays the publications of a faculty member which is again generated by extracting data from the Wiki page and is shown in Figure 4.4

Other than these pages there is another page with title Recent Research Papers. This page contains list of publications by all the faculty members in the past three years. This page is generated by extracting data from the Wiki pages of all the faculty members. This page is shown in Figure 4.5 and the query for the same is (getsection ‘‘2010’’) . (getlinks all) .(getpage ‘‘Department publications’’).

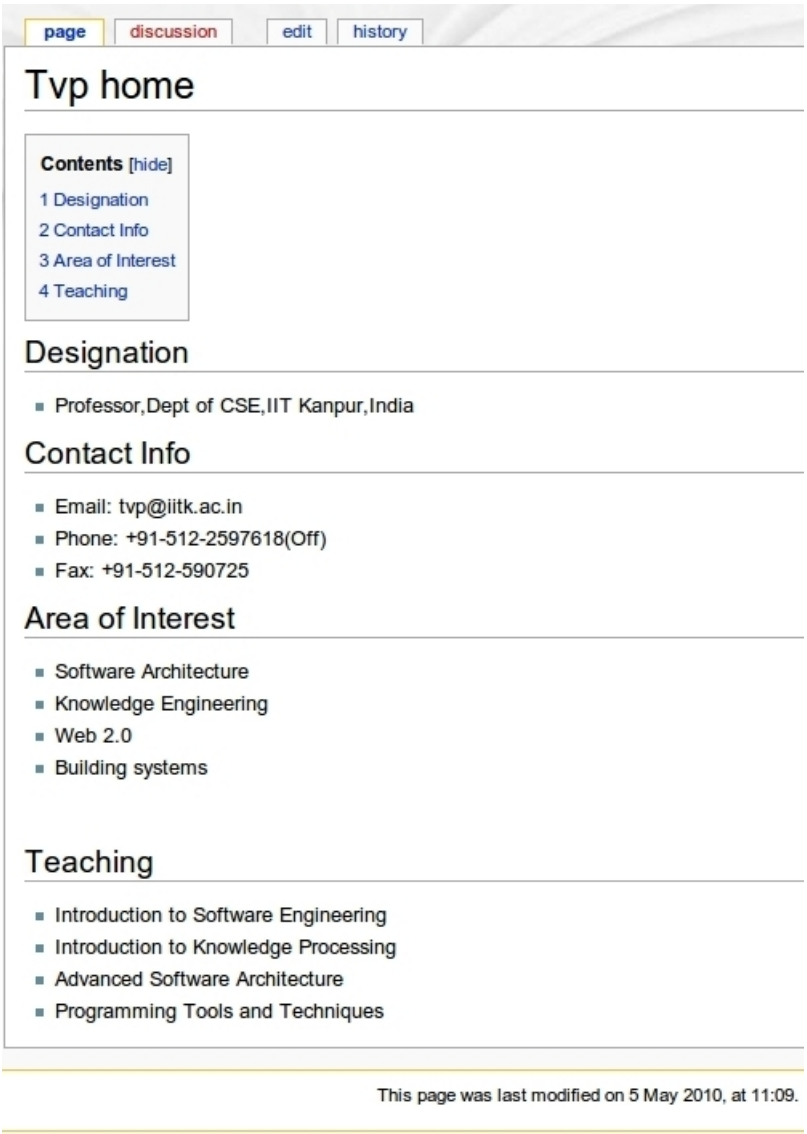


Figure 4.1: Wiki Page for faculty member

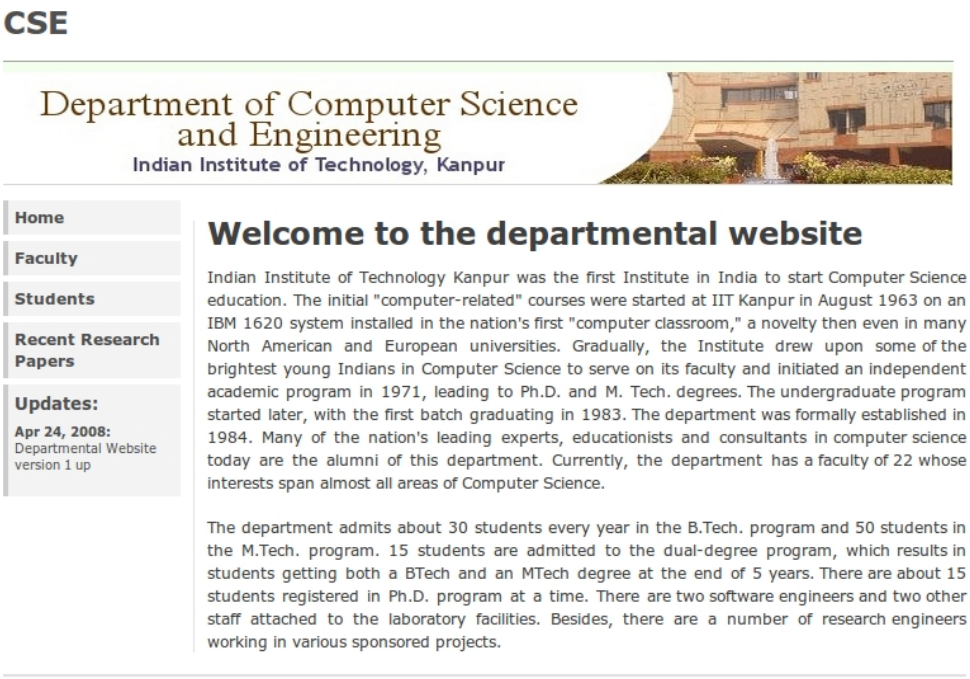


Figure 4.2: Index Page of Departmental Website

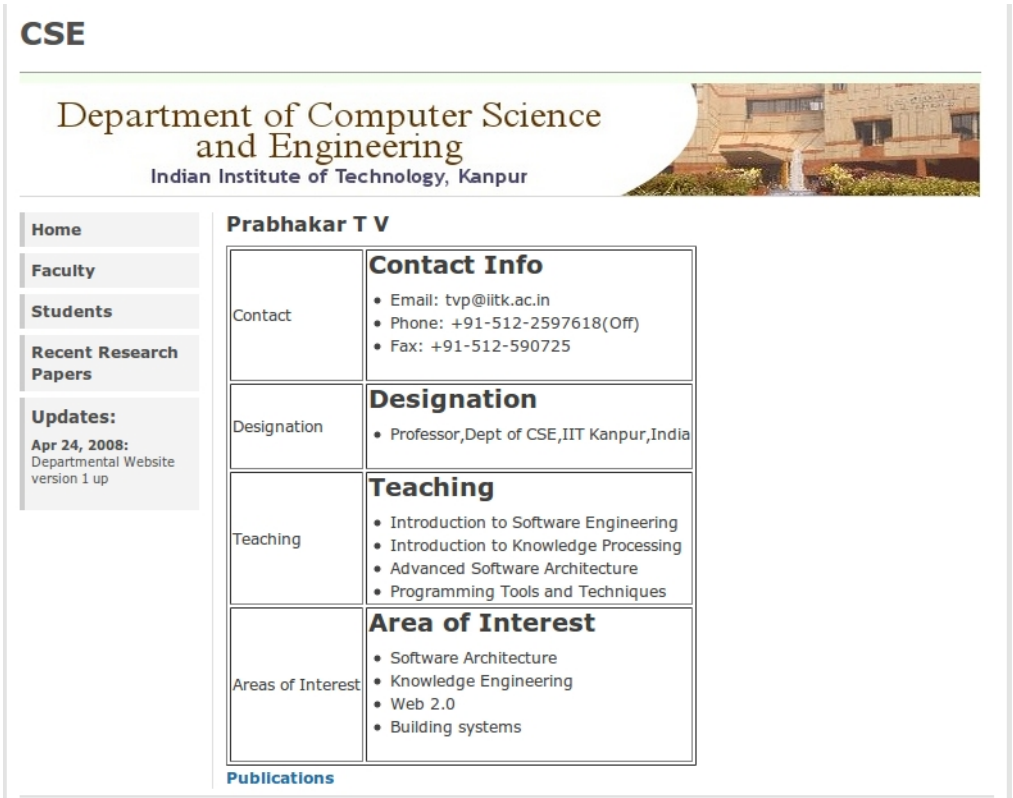


Figure 4.3: Page of a faculty member

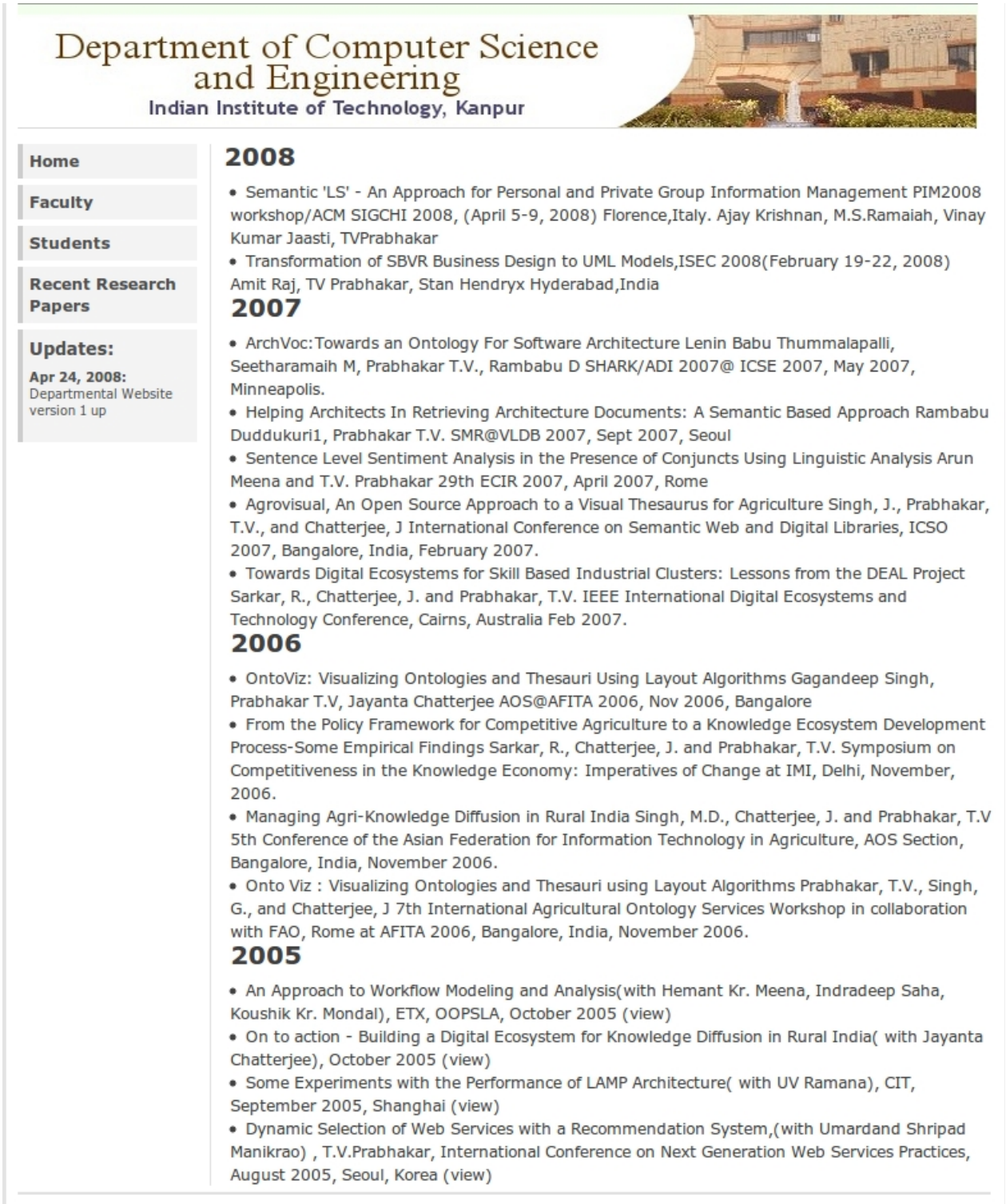


Figure 4.4: Page containing publications by a faculty member

Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Home
Faculty
Students
Recent Research Papers

Updates:
Apr 24, 2008: Departmental Website version 1 up

Recent Research Papers

2010

• Anil Seth, *Global Reachability in Bounded Phase Multi-Stack Pushdown Systems*, To appear in proc. 22nd International Conference on Computer Aided Verification (CAV 2010), Edinburgh, UK, July 15-19, 2010.

• Finding Top-k Similar Pairs of Objects Annotated with Terms from an Ontology. Arnab Bhattacharya, Abhishek Bhowmick, Ambuj K. Singh. International Conference on Scientific and Statistical Database Management (SSDBM), 2010, to appear, Heidelberg, Germany.
• Most Significant Substring Mining based on Chi-square Measure. Sourav Dutta, Arnab Bhattacharya. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2010, to appear, Hyderabad, India.
• Querying Spatial Patterns. Vishwakarma Singh, Arnab Bhattacharya, Ambuj K. Singh. International Conference on Extending Database Technology (EDBT), 2010, pages 418-429, Lausanne, Switzerland.

2009

• Amitabha Mukerjee, Using attentive focus to discover action ontologies from perception. Fifth International Workshop on Neural-Symbolic Learning and Reasoning NeSy09, Jul 11, 2009
• Amitabha Mukerjee, Discovering symbols from interactions easier than explaining interactions via symbols? Workshop on Logic and the Simulation of Reasoning, IJCAI-09, Pasadena CA Jul 12, 2009
• Amitabha Mukerjee and Madan Dabbeeru, Symbol emergence in design, Fifth International Workshop on Neural-Symbolic Learning and Reasoning NeSy09, Jul 11, 2009
• Amitabha Mukerjee and Madan Dabbeeru, The birth of symbols in design. 21th International Conference on Design Theory and Methodology, San Diego August 31-Sept 2, 2009.
• Madan M. Dabbeeru, Amitabha Mukerjee, Product Platform selection in Lower-Dimensional Manifold Spaces, DETC conference August 31-Sept 2, 2009, San Diego.

• Anil Seth, *Games on Higher Order Multi-stack Pushdown Systems*, In proc. third international workshop on Reachability Problems 2009 (RP09), Palaiseau, France, September 23-25, 2009. LNCS, vol. 5797, pp. 203-216.
• Anil Seth, *Games on multi-stack pushdown systems*, In Proc. Symposium on Logical Foundations of Computer Science, LFCS 2009 , Deerfield Beach, Florida, USA. LNCS, vol. 5407, pp. 395-408.

• Image Management for Biological Data. Arnab Bhattacharya, Vebjorn Ljosa. Book chapter in Encyclopedia of Database Systems edited by M. T. Ozsu and L. Liu. Springer, 2009.
• On Low Distortion Embeddings of Statistical Distance Measures into Low Dimensional Spaces. Arnab Bhattacharya, Purushottam Kar, Manjish Pal. International Conference on Database and Expert Systems Applications (DEXA), 2009, pages 164-172, Linz, Austria.

• Vishwas B. C., Abhishek Gadia, and Mainak Chaudhuri. Implementing a Parallel Matrix Factorization Library on the Cell Broadband Engine. In Scientific Programming special issue on high-performance computing with Cell BE, 17(1-2): 3-29, February 2009.

Figure 4.5: Recent Publications from the department

4.2 Gita Supersite

A website known as Gita Supersite is currently being hosted on a server at IIT Kanpur which contains shlokas of Gita and their translations and commentaries in Hindi, Sanskrti and English by various people. It has been made using PHP and MySQL. We have built the same website using JSP and WQL to see the advantage one gets by using WQL.

The main advantage that we have got by using WQL instead of SQL is that information providers can now directly edit the content themselves without the need of having a database administrator.

4.2.1 Wiki Design

There are total 701 shlokas in Gita which have been divided into 18 chapters and for every shloka we have 27 options(mool shlokas, translations and commentaries). So for every shloka there are 27 different Wiki pages(shown in Figure 4.7). Other than these 18927 Wiki pages, there are few index pages as well which contains links to these Wiki pages. So there is a main page(show in Figure 4.6) which contains links to the index pages of all these options. Then these index pages of every option contains links to different chapters(shown in Figure 4.8) which in turn contains links to the shloka pages(shown in Figure 4.9).

4.2.2 Website

The index page of Gita Supersite contains a form where a user can select what all things he/she wants to view. Also he can specify the chapter number and shloka number. Once user submits the form, the corresponding query string is sent to the Wiki interpreter and the information is fetched from the Wiki engine. The generated web-page is then sent back to the client which is displayed in his/her browser. Figure 4.10 shows the index page and Figure 4.11 shows one sample dynamically generated web-page.

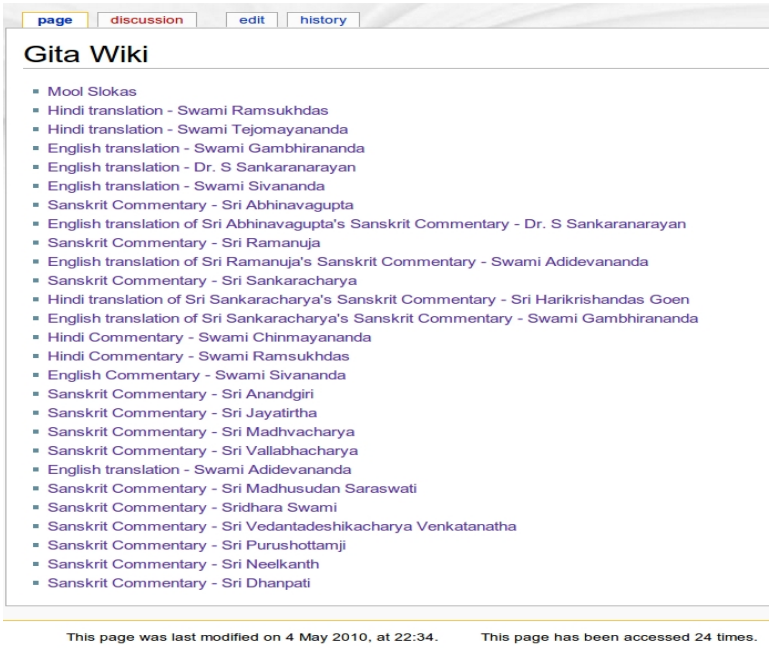


Figure 4.6: Index Page of Gita Wiki

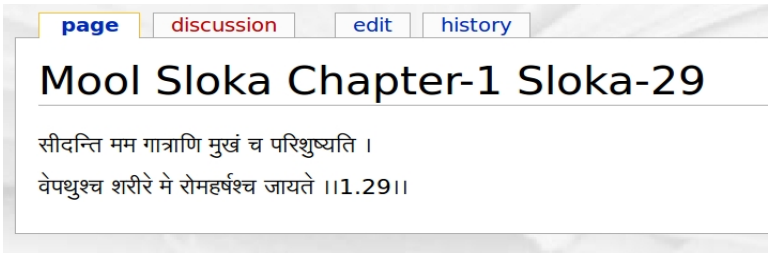


Figure 4.7: Mool Shloka's Wiki Page

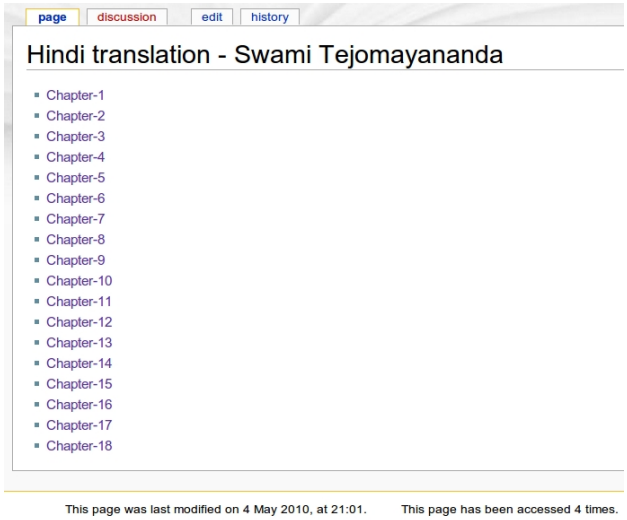


Figure 4.8: Index Page of chapters containing hindi translation

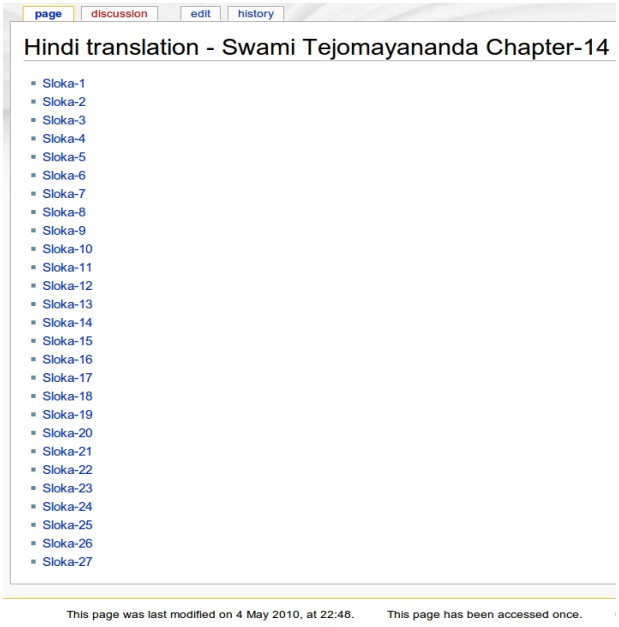


Figure 4.9: Index page of shlokas of one chapter

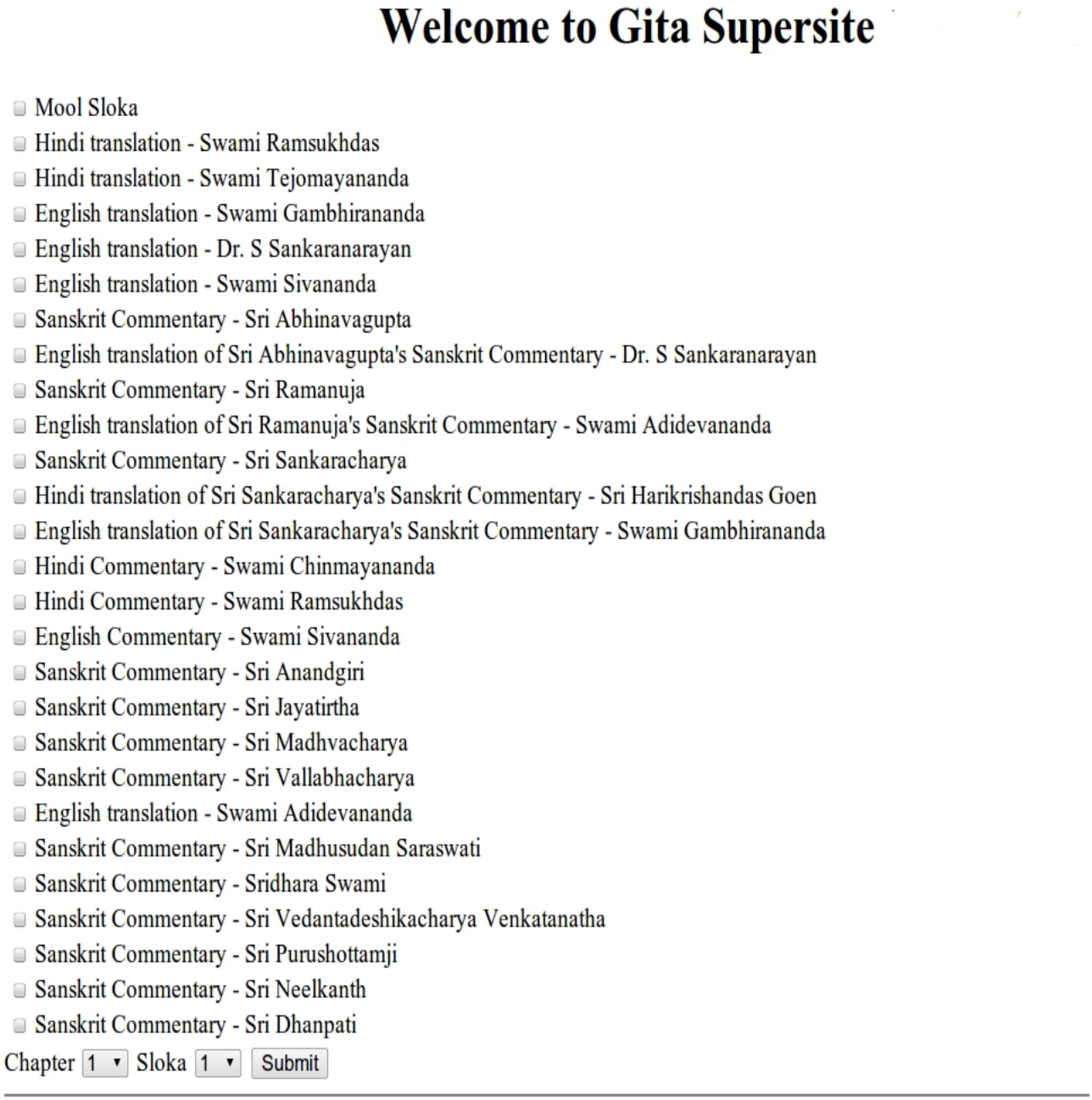


Figure 4.10: Index Page of Gita Supersite

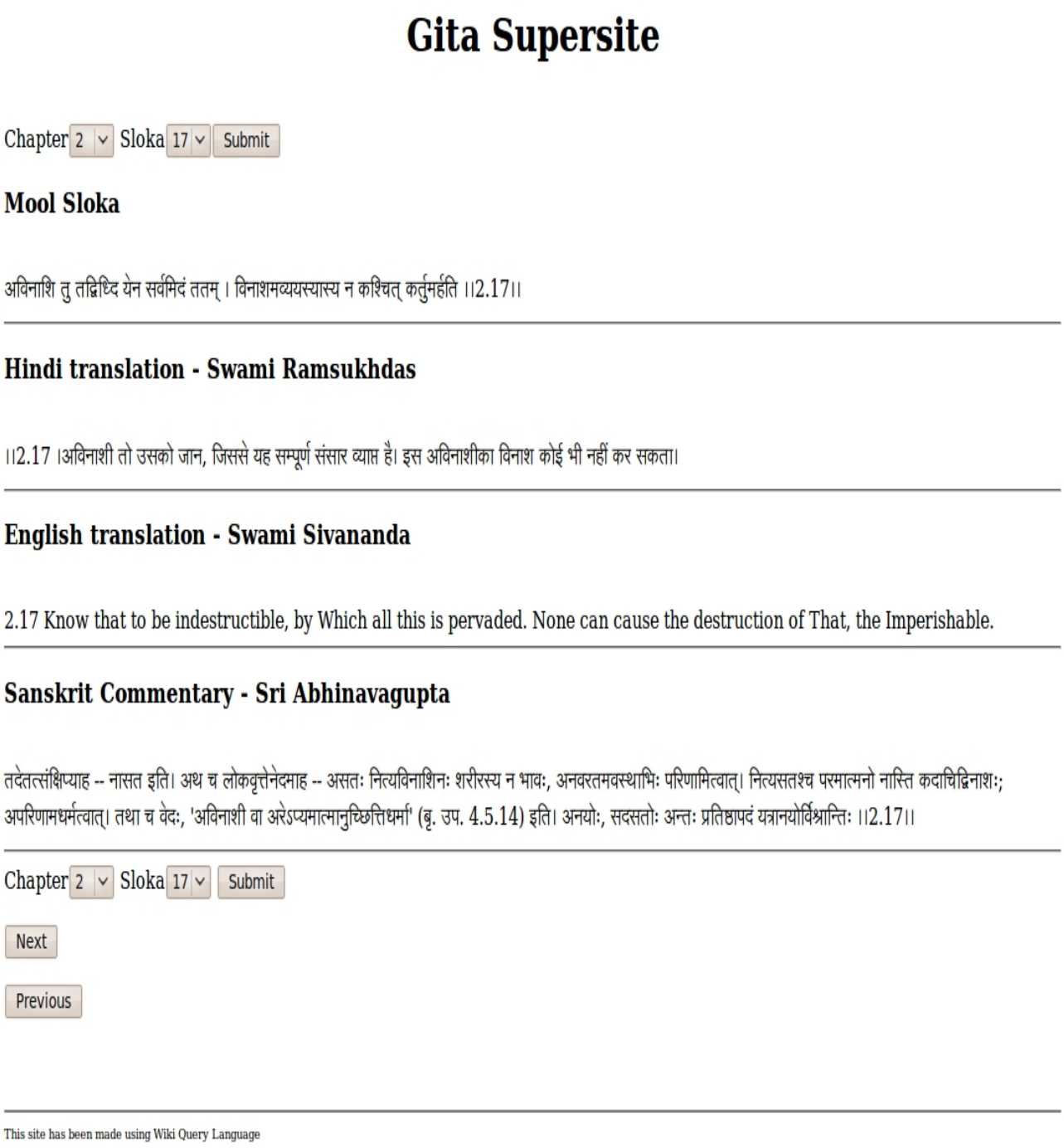


Figure 4.11: Dynamically Generated web-page

4.3 Collaboration Portal for Students' Placement Team

Students' Placement Office is run by students and office staff. Every year a big team of students is made who coordinates with office staff and looks after the placement work. The team consists of around 80 students and is headed by 3 students. Each student has his role outlined and only he is responsible for the job assigned to him. Typically every year Student's Placement Team sends invitations to around 1100 companies and looks after the

placement of around 1000 students. Since the team is quite big and the work huge, it becomes difficult for the three coordinators to keep a tab on each and every thing. Sometimes there is also a communication gap between the team members which leads to not so pleasant scenarios. To overcome these hurdles we have built a collaboration portal using WSL which will help the placement team in better communication among the team members and will help the coordinators in managing the team in an efficient manner.

4.3.1 Wiki Design

There are two types of entities in this system - one is students and the other is companies. Every team member has a Wiki page of their own which contains information about him and his work. Following is the XML representation of the Wiki page of a team member -

```
<root title = "Himanshu Govil">
<section title = "Designation">
    SOME TEXT HERE
</section>
<section title = "Contact Info">
    SOME TEXT HERE
</section>
<section title = "Companies">
    SOME TEXT HERE
</section>
<section title = "Recent Updates">
    SOME TEXT HERE
</section>
<section title = "To Do List">
    SOME TEXT HERE
</section>
</root>
```

In the Wiki page of a student, Companies section contains list of companies which the student is trying to contact. In Recent Updates section, he adds the updates of last week and in To Do List section he maintains the work needed to be done in the coming week. So, team members maintain their Wiki pages and update information as and when something comes up. Figure 4.12 shows one such wiki page.

Other than the Wiki pages of students, there is a Wiki page of each company as well which contains these sections –

- Sector: Sector to which the company belongs to.
- Who is contacting: Name of the team member who is looking after the company.
- Contact persons: Nodal points in the company.
- Updates: All the updates related to the company.

A sample Wiki page of a company is shown in Figure 4.13

page | discussion | edit | history

Himanshu Govil

Contents [hide]

Designation

Member, Core Team

Contact Info

Email: govilh@iitk.ac.in

Gmail: govil.himanshu@gmail.com

Phone: 09839381974

Companies

- Parle
- Britannia
- Colgate Palmolive
- Asian Paints
- Gamble3

Recent Updates

- Parle has agreed to fill the proforma
- Called Britannia this week and they are still not sure about recruiting this year

To Do List

- Call Asian Paints on 15th June
- Call Colgate Palmolive on 15th June
- Call Parle and check for confirmation

Figure 4.12: Wiki page of a team member

Figure 4.3 Collaboration Portal for Students' Placement Team

page | discussion | edit | history

Parle

Contents [hide]

1 Sector
2 Who is contacting
3 Contact Persons
4 Updates

Sector

FMCG

Who is contacting

Himanshu Govil

Contact Persons

- Ranjeet Kumar Mishra: HR Person who looks after campus recruitment.

Email: ranjeet.mishra@pg.com

Phone: 9838XXXXXX

- Anuj Midha: He is an IITK Alumnus. He graduated in 1996 and is currently a Senior Manager in Mumbai Office

Email: anuj.midha@pg.com

Phone: 9838XXXXXX

Updates

- Was able to find the right person on 10th May
- Talked to Ranjeet for the first time on 14th May and sent an invitation mail
- 18th May: He said this time they want to recruit in good numbers and hence are looking forward to visit new campuses
- 24th May: Will discuss with the higher management whether to visit IIT Kanpur or not
- Talked to Ranjeet on 2nd June and he said that they definitely want to visit IIT Kanpur this time
- Parle has agreed to fill the proforma
- Parle coming to campus this year
- Parle has agreed to come to the campus
- Parle coming to campus
- Parle coming to campus
- Parle will conduct off campus recruitment
- Parle coming to campus next year

Figure 4.13: Wiki page of a company

4.3.2 Collaboration Portal

The online portal contains various web-pages which are generated by extracting information from the Wiki. The titles of these web-pages are Company wise updates, Student wise updates, To Do List, Contact Details and Create company. Let us look at each one them in a bit detail -

• Index Page: The index page of the portal is a simple web-page and looks as follows -

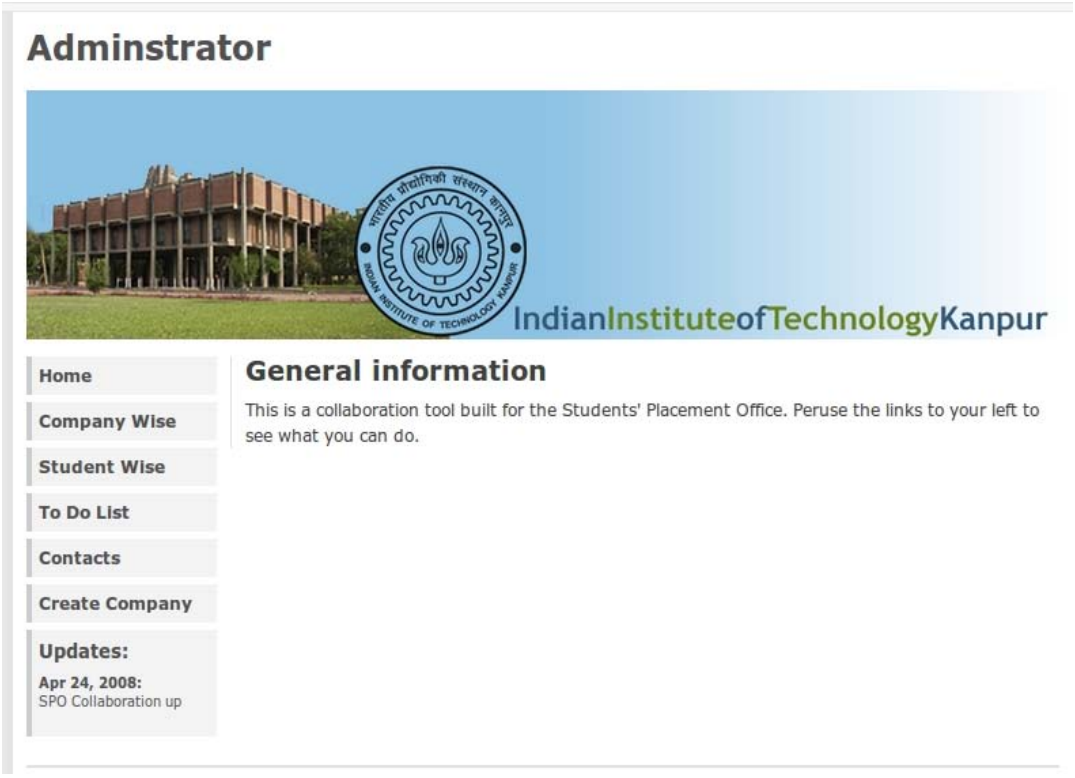


Figure 4.14: Index Page of the portal

• Company wise Updates: This page displays the updates of every company. This is done by extracting the Updates section from the Wiki pages of all the com- panies. The query to do the same is - (getsection "Updates"). (getlinks all).(getpage "Companies"). Figure 4.15 shows this page.

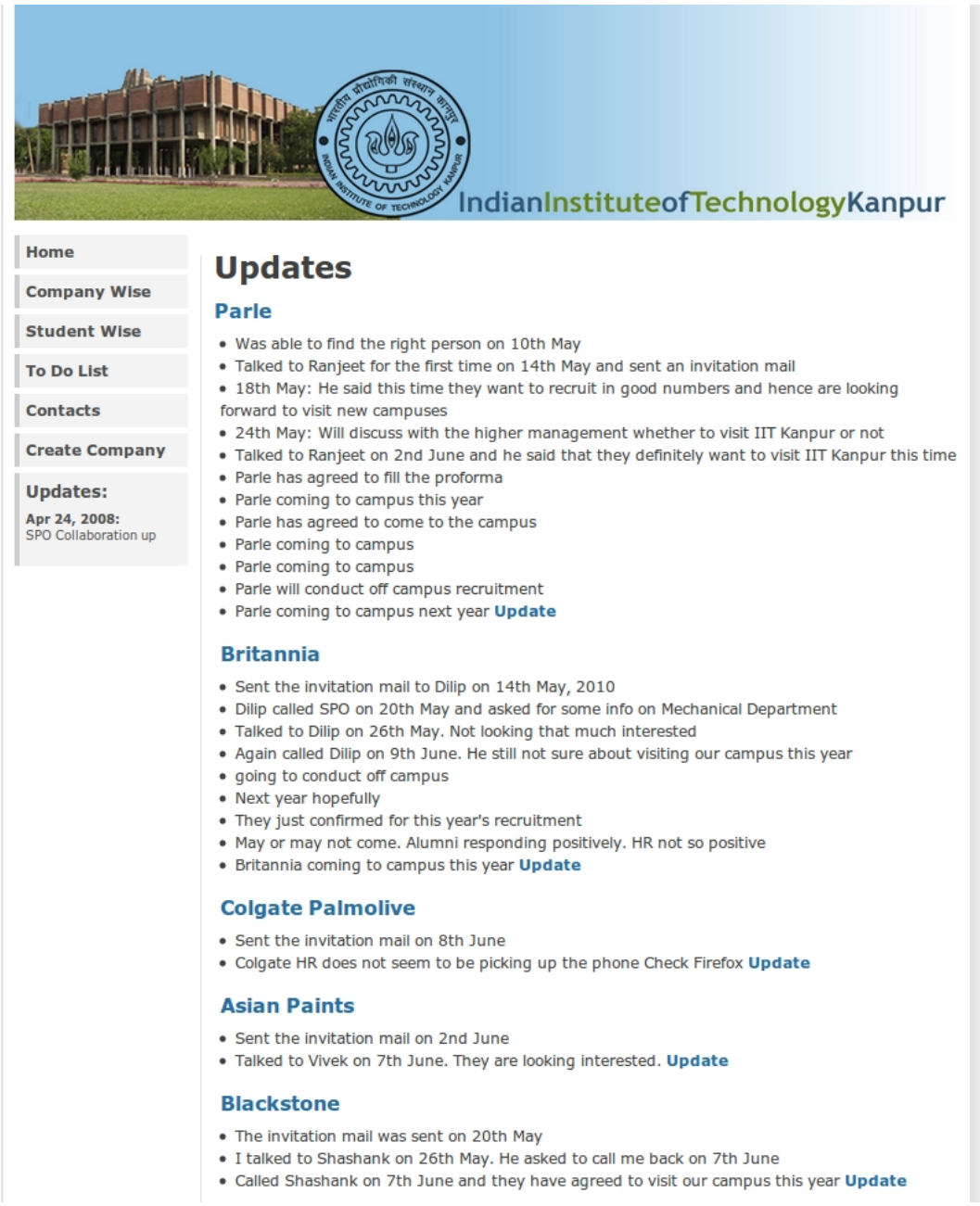


Figure 4.15: Company Wise Updates

This page also contains an “update” button for every company using which users can add new updates, if there is any, from the portal itself. The same will be added to the Wiki as well. This is done by using appendsection construct.

• Student wise updates: This page displays the updates of every team member. This is done by extracting the Updates section from the Wiki pages of all team members. The query to do the same is - (getsection "Updates") .
(getlinks all).(getpage "SPO Team") where SPO Team is a Wiki page containing links to the Wiki pages of team members. Figure 4.16 shows this page.

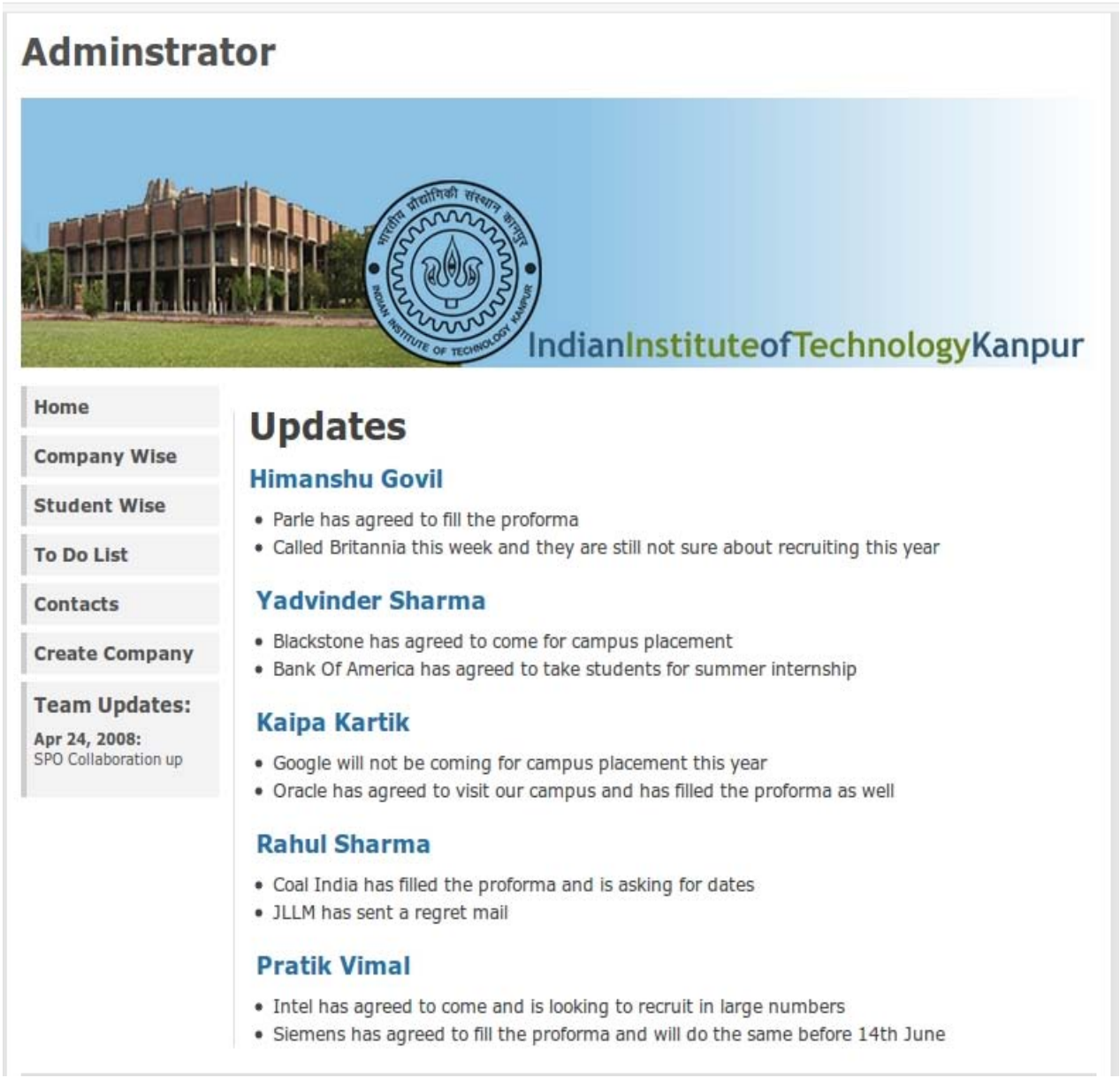


Figure 4.16: Student Wise Updates

• Contacts: This page contains the contact information of all team members. This page is generated by extracting the Contact Info section from the Wiki pages of all team members. The query to do the same is - ((getsection "Contact Info") . (getlinks all) . (getpage "SPO Team")). The page is shown in the following figure -

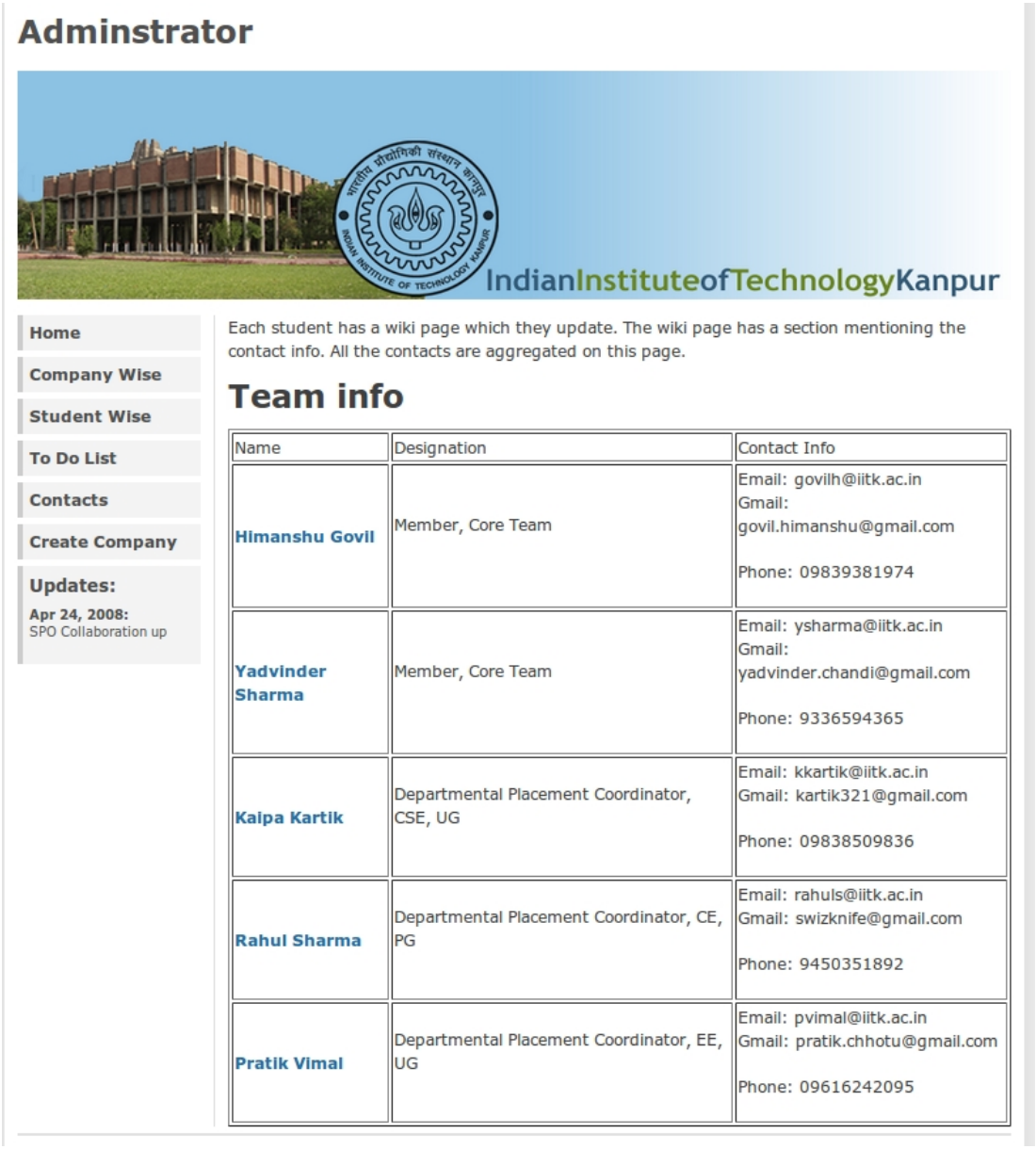


Figure 4.17: Contact Info of Team Memb

• To Do List: This page contains information about the work each team members has to do in the coming page. This page is generated by extracting the To Do List section from the Wiki pages of all team members. The query to do the same is - ((getsection "To Do List") (getlinks all) .(getpage "SPO Team")). The page is shown in the following figure -

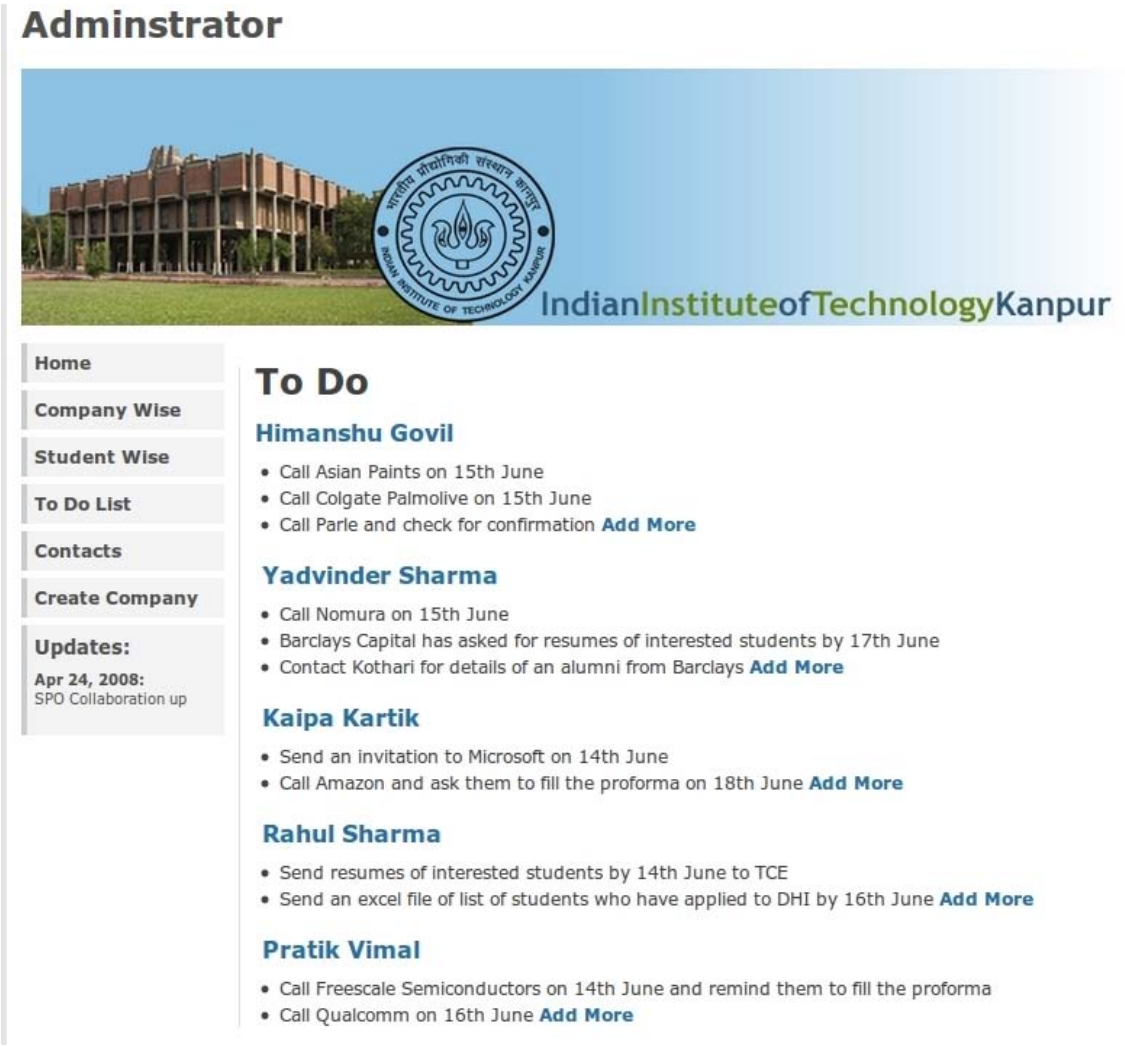


Figure 4.18: To Do List

This page also contains an “Add More” button for each team member using which the coordinators can assign some task to a team member from the porta
itself. The same will the added in the To Do List section of the Wiki page of that team member. This has been implemented by using appendsection construct.

• Create Company: This page is basically a form which allows users to create new Wiki pages. Administrator can use this form to add new companies into the Wiki and can assign the company to a team member as well. Administration will fill the form and will click on the submit button. After this a new Wiki page is created and the Wiki page of the team member also gets updated.

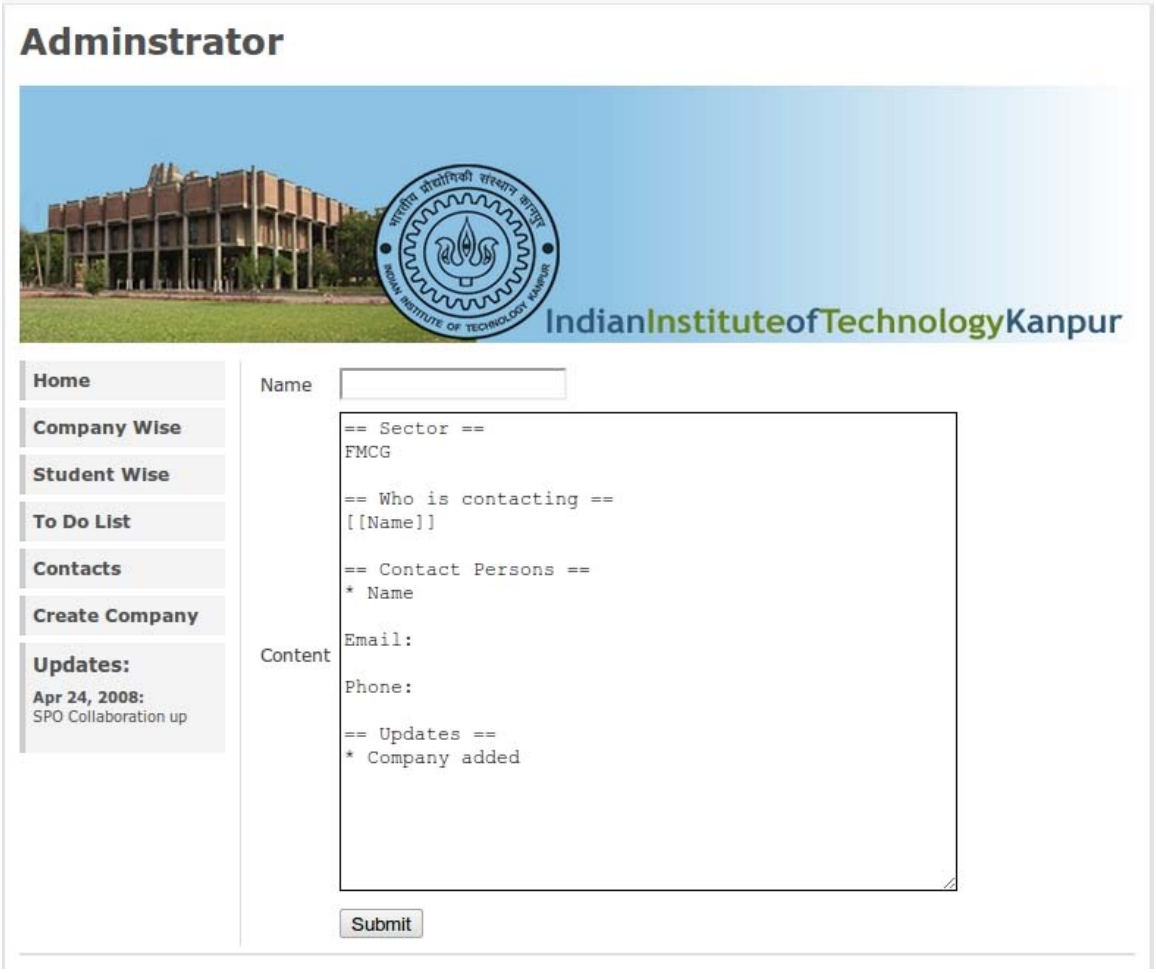


Figure 4.19: Create Company

4.4 Wikibooks

Wikibooks is a simple publishing application which compiles information from different Wiki pages and prints all of it at one page. The main feature this application posseses is that the user can choose which version of the Wiki page he/she wants to publish. Wikibooks can be used in many different scenarios. Imagine a situation where a big team is working on some project and within the team there are 3-4 groups working on different modules. Now each group, within themselves, can collaborate on Wiki and can make a technical

report of their module. At the end all the reports can be combined to make a single report. Now, in this case the team can decide which versions of different reports they want to choose.

We have built an application which publishes a Wiki book on rock bands. Each chapter contains some information about a rock band. Let us look at the Wiki design and the application a bit closely.

4.4.1 Wiki Design

The Wiki design in this one is pretty simple. There is a separate page for each chapter and other than these chapters there is an index page as well which contains links to the individual Wiki pages. Figure 4.20 shows the index page and the Figure 4.21 shows a sample Wiki page for one chapter.

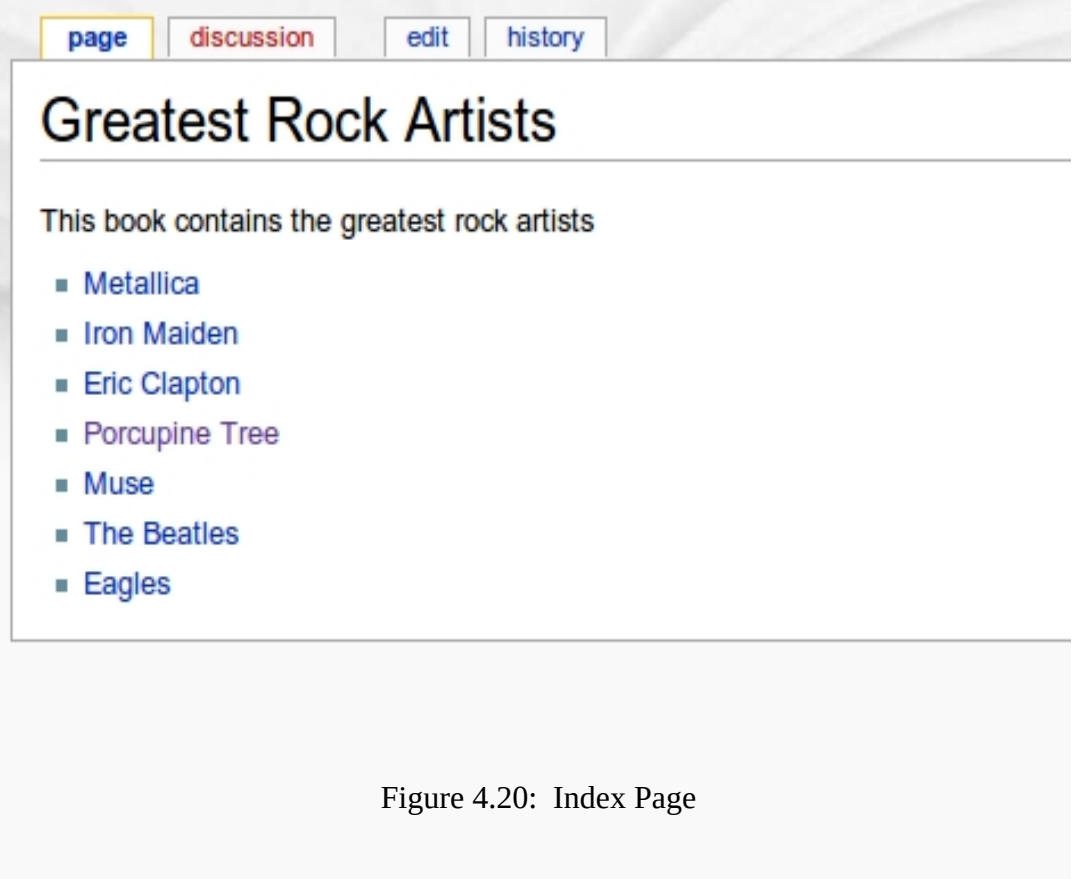


Figure 4.20: Index Page

page | discussion | edit | history

Porcupine Tree

Porcupine Tree are a progressive rock band formed by Steven Wilson in 1987 in Hemel Hempstead, Hertfordshire, England. Their music is difficult to categorize, being associated with both psychedelic rock and progressive rock, yet having been influenced by trance, krautrock and ambient due to Steven Wilson and Richard Barbieri's liking for the Kosmische Musik scene of the early '70s, led by bands such as Tangerine Dream, Neu! and Can[1]. Since the early 2000s, their music has been leaning towards progressive metal.

The band are noted for their multimedia approach, with their live performances including screens displaying a different film projection to each song. This visual element was introduced during the tour for the In Absentia album, when the band started to work with Danish photographer and filmmaker Lasse Hoile, whose involvement has created a distinctive image for the band.

Despite being signed to both Roadrunner and Atlantic labels[2][3], the band has their own record label, Transmission, which they use to launch some independent releases and special editions of their albums. Their 2007 album Fear of a Blank Planet was nominated for a Grammy Award for Best Surround Sound Album[4].

Fear of a Blank Planet (2006 to 2007) [edit]

The band's website announced that new material would be played during the first half of their tours of Europe and the United States. Their new material was much heavier and layered than anything they had previously done, indicating that Porcupine Tree was heading toward an even more metal-oriented sound.

On 8 August 2006, it was announced that Porcupine Tree had signed with Roadrunner Records UK. Wilson commented that "Roadrunner has established itself as one of the world's premier independent labels for rock music, and we couldn't be more enthusiastic about working with them to expand our audience and elevate Porcupine Tree to the next level."[23]

The first Porcupine Tree concert DVD, Arriving Somewhere..., was released on 10 October 2006. It was accompanied by a brief tour in which the group performed 50 minutes of new material for the forthcoming studio album for the first half of the shows. Supporting acts included Swedish band Paatos in Europe (except France and Belgium where they were supported by Oceansize), and ProjeKCt 6 (Robert Fripp and Adrian Belew) in the USA.[13]. In January, 2007, it was revealed the title for the forthcoming album would be Fear of a Blank Planet[24]

With the release of Fear of a Blank Planet on 16 April 2007, Porcupine Tree charted in almost all European countries.[25] and peaking at #59 on the Billboard 200[26] A 92-date tour for 2007, took the band to countries they had never visited, like Finland and Mexico. The tour included appearances in many major music festivals such as the German twin-festivals, Hurricane[27] and Southside[28], and the Download Festival of Donington Park[29]. Later in 2008 when the tour resumed, the band performed their first ever shows in Australia.

The lyrics of the album deal with common behaviour tendencies concerning society (especially youth) in the beginning of the 21st century such as bipolar disorder, attention deficit disorder, drug abuse, alienation[30] and depravation caused by mass media[20]. Richard Barbieri during a Porcupine Tree performance in Kraków, Poland, in 2007

Wilson: "My fear is that the current generation of kids who're being born into this information revolution, growing up with the Internet, cell phones, iPods, this download culture, 'American Idol,' reality TV, prescription drugs, PlayStations — all of these things kind of distract people from what's important about life, which is to develop a sense of curiosity about what's out there."[31]

The concept of the album was inspired mainly by Bret Easton Ellis novel Lunar Park[32] and the title alludes to Public Enemy's album, Fear of a Black Planet, both sharing the particularity of reflecting notorious conflicts affecting society in the world at some time. Wilson notes that whilst race relationship was the main issue among young people when Public Enemy's album was released, in the 21st century it was replaced by a general superficiality, boredom, and introversion[33]. The album features contributions from Rush's Alex Lifeson and King Crimson's Robert Fripp.

A new EP called Nil Recurring was released on 17 September 2007, featuring four unreleased tracks from the Fear of a Blank Planet sessions and including another contribution from Robert Fripp. The second leg of the tour started on 3 October 2007, now promoting new music from the EP. Nil Recurring entered the UK Top 30 Independent Label Albums at #8[34]. The EP was later reissued in 18 February 2008 through Peaceville Records[35].

On 5 November 2007, Fear of a Blank Planet won the "Album of the Year" award for the 2007 Classic Rock magazine awards[36]. In December, 2007, it was nominated for a "Best Surround Sound Album" Grammy though Love by The Beatles won the award[4]. In January, 2008, was voted "Best Album of 2007" by readers of the Dutch Progressive Rock Page[37]. The LP version of Fear of a Blank Planet includes the Nil Recurring EP tracks.

A recording from an 4 October 2007 in-store, mostly acoustic, performance at Park Avenue CDs in Orlando (Florida) was released on 18 February 2008 on CD under the name of We Lost The Skyline[38]. The title is a reference to the lyrics of "The Sky Moves Sideways (Phase One)," which was the opening song on the live set. The album was released on vinyl of 21 March 2008[39]. It was originally intended to be a full-band show, but lack of space in the store determined that only the two guitarists, Steven Wilson and John Wesley, played.

Figure 4.21: Wiki Page for a chapter

4.4.2 Application

The index page of Wikibooks application is a form where a user can select what all chapters and versions of them he/she wants to view. Once user submits the form, the corresponding query string is generated and is then sent to the Wiki interpreter. The required information is fetched from the Wiki engine and is converted into HTML format. The dynamically generated HTML content is then sent back to the client which is displayed in his/her browser. Figure 4.22 shows the index page and Figure 4.23 shows one sample dynamically generated web-page. One feature that we have added is that user can easily reorder the chapters by simply moving chater names using mouse.

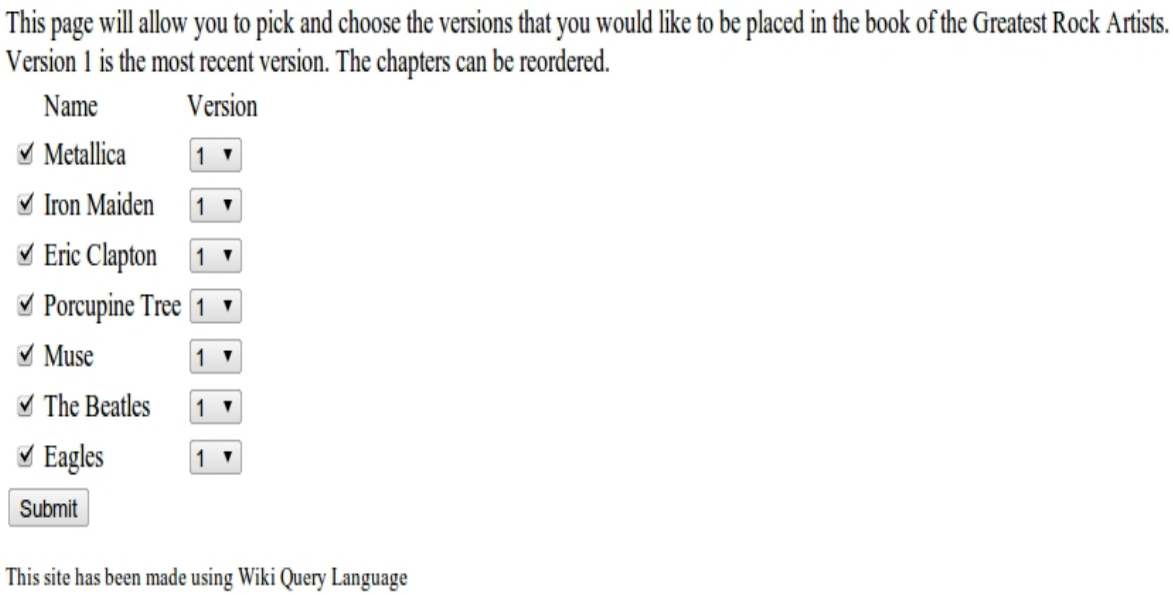


Figure 4.22: Index Page of Application

Chapter 2 Iron Maiden

Iron Maiden are an English heavy metal band from Leyton in east London, formed in 1975. The band are directed by founder, bassist and songwriter Steve Harris. Since their inception, the group has released a collective total of thirty albums: fourteen studio albums; seven live albums; four EPs; and five compilations.

Pioneers of the New Wave of British Heavy Metal, Iron Maiden achieved success during the early 1980s and, after several lineup changes, the band went on to release a series of platinum and gold albums. These include the US platinum-selling landmarks The Number of the Beast in 1982, Piece of Mind in 1983, Powerslave in 1984, the acclaimed live album Live After Death in 1985, Somewhere in Time in 1986, and Seventh Son of a Seventh Son in 1988. Their second most recent studio effort, A Matter of Life and Death, was released in 2006 and peaked at number nine on the Billboard 200 and at number 4 in the UK and is also one of the few rock albums to be certified platinum in India. Their newest album, The Final Frontier, will be released on August 16, 2010.[1]

As one of the most successful heavy metal bands in history, Iron Maiden have sold over 75 million records under EMI and a total of over 100 million records worldwide with almost no radio or television support.[2][3][4][5] The band won the Ivor Novello Award for international achievement in 2002,[6] and were also inducted into the Hollywood RockWalk in Sunset Boulevard, Los Angeles, California during their United States tour in 2005. As of October 2009, the band has played just over 2000 live shows during their career. For the past 30 years, the band has been supported by their famous mascot, "Eddie the Head", who has appeared on almost all of their album and single covers, as well their live shows.

Chapter 3 Eric Clapton

Eric Patrick Clapton, CBE (born 30 March 1945) is an English blues-rock guitarist, singer, songwriter and composer. Clapton is the only person who has been inducted into the Rock and Roll Hall of Fame three times; as a solo performer, as well as a member of rock bands the Yardbirds and Cream. Often viewed by critics and fans alike as one of the most important and influential guitarists of all time,[2] Clapton was ranked fourth in Rolling Stone magazine's list of the "100 Greatest Guitarists of All Time"[3] and #53 on their list of the "Immortals: 100 Greatest Artists of All Time".[4] In 2010, Clapton was ranked #4 on Gibson.comâ€™s Top 50 Guitarists of All Time.[5]

Although Clapton has varied his musical style throughout his career, it has always remained grounded in the blues; despite this focus, he is credited as an innovator in a wide variety of genres. These include blues-rock (with John Mayall & the Bluesbreakers and The Yardbirds) and psychedelic rock (with Cream). Clapton's chart success was not limited to the blues, with chart-toppers in Delta Blues (Me and Mr. Johnson), Adult contemporary ("Tears in Heaven") and reggae (Bob Marley's "I Shot the Sheriff"; he is often credited for bringing reggae and Bob Marley to the mainstream).[6] Two of his most successful recordings were the hit love song "Layla", which he played with the band Derek and the Dominos, and Robert Johnson's "Crossroads", which has been his staple song since his days with Cream.

Chapter 4 Porcupine Tree

Porcupine Tree are a progressive rock band formed by Steven Wilson in 1987 in Hemel Hempstead, Hertfordshire, England. Their music is difficult to categorize, being associated with both psychedelic rock and progressive rock, yet having been influenced by trance, krautrock and ambient due to Steven Wilson and Richard Barbieri's liking for the Kosmische Musik scene of the early '70s, led by bands such as Tangerine Dream, Neu! and Can[1]. Since the early 2000s, their music has been leaning towards progressive metal.

The band are noted for their multimedia approach, with their live performances including screens displaying a different film projection to each song. This visual element was introduced during the tour for the In Absentia album, when the band started to work with Danish photographer and filmmaker Lasse Hoile, whose involvement has created a distinctive image for the band.

Despite being signed to both Roadrunner and Atlantic labels[2][3], the band has their own record label, Transmission, which they use to launch some independent releases and special editions of their albums. Their 2007 album Fear of a Blank Planet was nominated for a Grammy Award for Best Surround Sound Album[4].

Figure 4.23: Dynamically Generated Web-page

4.5 Advantages over the current system

• Easier to edit information: In present scenario if a person wants to make changes in his web-page which is hosted on some server, he first has to connect to that server and download the HTML file. Then he needs to make changes in HTML format and then again upload the latest version of that file on to the server. However, if the website has been built using WQL or WSL and is fetching data from Wiki, user only needs to make changes in his/her Wiki page which can be done using a web browser. Moreover, one does not have to learn HTML to edit a web-page, he/she only needs to edit Wiki text which is plain text with some rules.

• Since editing information available on a web-page is now easier, people will update their web-pages more frequently than they do now.

• No need of a web-administrator/database administrator: The departmental web- site which we have built contains a web-page which displays the recent research papers of all the faculty members at one place. In the current system, every faculty member has a separate page of their own where they have listed out their publications. If the department wants to have a web-page which displays pub- lications of every faculty member without using a Wiki, a web administrator is required whose job will be to assimilate all this information and publish it on a web-page.
When we use SQL adding new features to an already existing system would have been a tougher task. A database administrator who can add columns and tables to the database without breaking any of the existing functionality would have been needed and additional work would have been needed to reflect these changes in the application. But any one can edit a wiki page and all one would need to do is add new sections to bring about new changes. Hence the database administrator is redundant.

• Less time required to build web applications: As of now developers do spend some amount of time in designing the database for web applications. However, if some- one is using WSL or WQL to build one, this time can be saved. Also, newbies can build dynamic web applications without having to learn about Relational Database Management Systems and SQL. Moreover, Wiki already provides fea- tures to edit a Wiki page and to create new Wiki pages. Therefore, developers do not have to create HTTP forms to add entries in database or to update SQL tables and hence less work needs to be done on this front.

5. Walkthrough

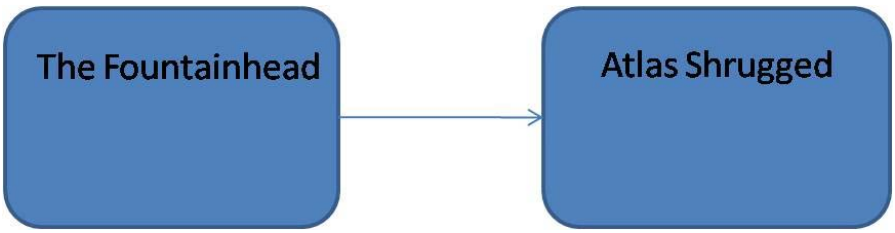
Let us see how the wiki interpreter executes a query step by step.

```
(getsection [“Author” , “Plot”]) .  (getpage [“The Fountainhead”, “Atlas Shrugged])
```

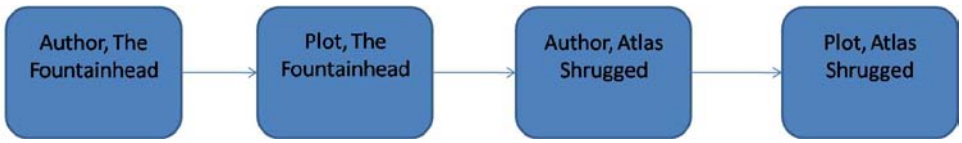
Step-1: Wiki interpreter will receive the query string. It will parse it and start executing from the rightmost parentheses.

Step-2: Now the first action is getpage. So the wiki interpreter will read the title of pages it needs to download. After knowing the titles, it will fetch both the wiki pages.

Step-3: Once the fetching is done, wiki interpreter will process the pages and will create a list of nodes as shown in the following figure



Step-4: Wiki interpreter will now look at the second parentheses and will perform the required action. Now next construct is getsection. So, it will fetch the required sections from each node and will construct a new node for each section. The list will now look like as shown as follows -



Step-5: This list will be returned back to JSP or WSL.

6. How to get started ?

The entire software is available at http://sourceforge.net/projects/wql/

6.1 WQL

• Download Java from http://java.com/en/download/ and install it.

• Download Netbeans IDE from http://netbeans.org/downloads/ and install it on the machine where you want your application to run.

• We have built a plugin for Netbeans which creates a sample project. Download this plugin along with the other libraries(refer to the following image) that we have provided.

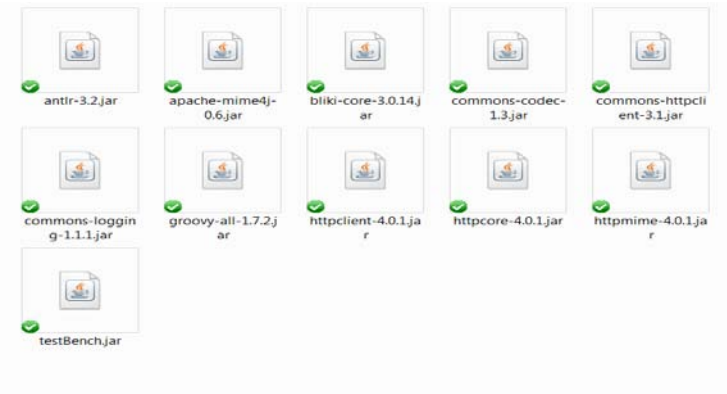


Figure B.1: Jar files present in library

• To install the plugin open Netbeans and go to Tools→ Plugins. A window will get opened and then go to Downloaded tab. The window will look like -

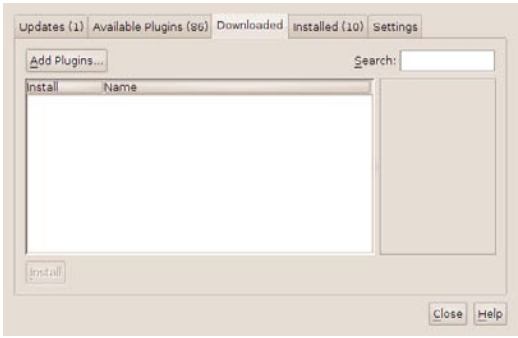


Figure B.2: Plugin Window

• Click on the Add Plugins button and go to the directory where you saved the plugin. Add it and after that click on the Install buttom which is at the bottom. Follow the on-screen instructions and the plugin will get installed.

• To create a WQL application using JSP, go to File→ New Project. A window will be opened once you click on New Project. In the Categories column choose Java Web and in the Project Column choose WQL(refer to Figure B.3). The netbeans project contains the

sample files that are required to get you started. A sample jsp file and and a servlet Sample.java are included for reference.

• Add the jars provided in the libraries.

• You are good to go now.

If you want to integrate WQL with your existing jsp project, all you need to do is to include all the jars that we have provided in a zip file. If you are using an IDE such as Eclipse or Netbeans then adding the jars is a simple matter. However, if you are using the commandline then all one needs to do is to set the CLASSPATH environment variable to include the jars.

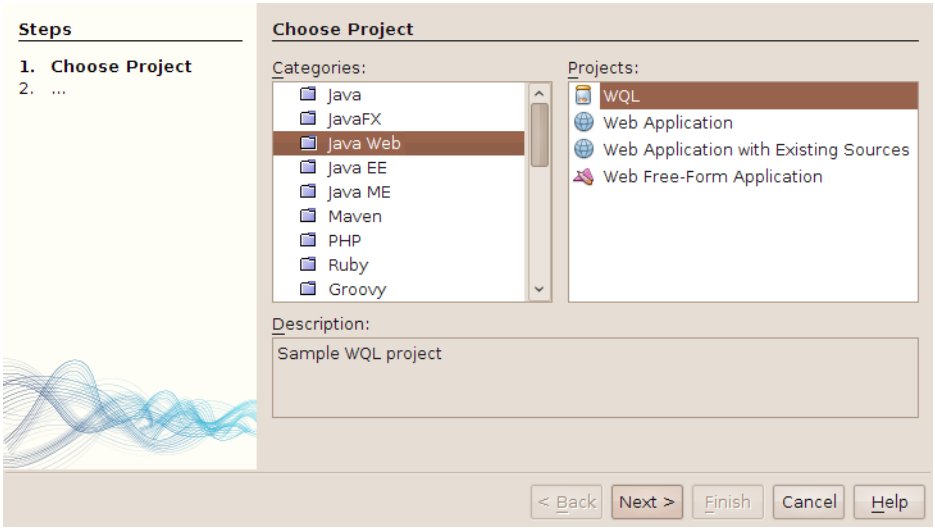


Figure B.3: New Project

6.2 WSL

• Download Java from http://www.java.com/en/download/ and install it.

• We have provided a installer for WSL. This is in essence a modified version of the tomcat installer provided by Apache. The installer takes care of installing the entire stack required for WSL. Figure B.4 to Figure B.9 shows all the steps of installation.

• Open the folder where you have installed WSl(By default this is C:\ProgramFiles\Apache Software Foundation\Tomcat6.0).

• Open the directory webapps in this folder.

• There is a webapp called wsl in this folder. This folder contains *.wsl files to help you get started writing code. This webapp is fully configured and ready to go.

• Most websites have index.html as the start page. The wsl application uses in- dex.wsl as the start page.

• Create as many wsl files as you want in this folder.
• Once the server is started the application can be viewed inside your browser on the address http://localhost:8080/wsl.

• More applications can be created by simply making copies of the wsl folder and renaming it as per your choice.

• You are ready to go.

In case the developer wishes to deploy our project on an existing tomcat installation, the user has to place the sample tomcat project in the webapps folder of tomcat. The rest of the instructions remain the same.

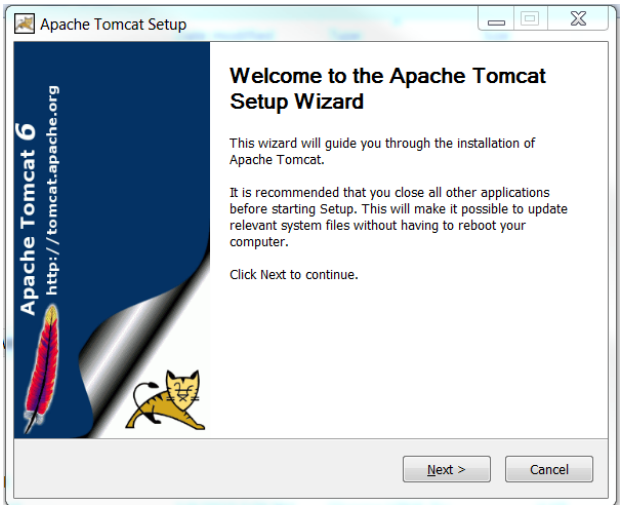


Figure B.4: Step-1

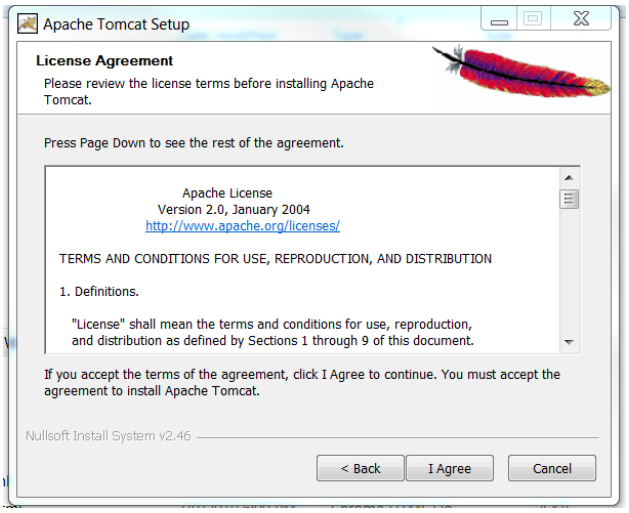


Figure B.5: Step-2

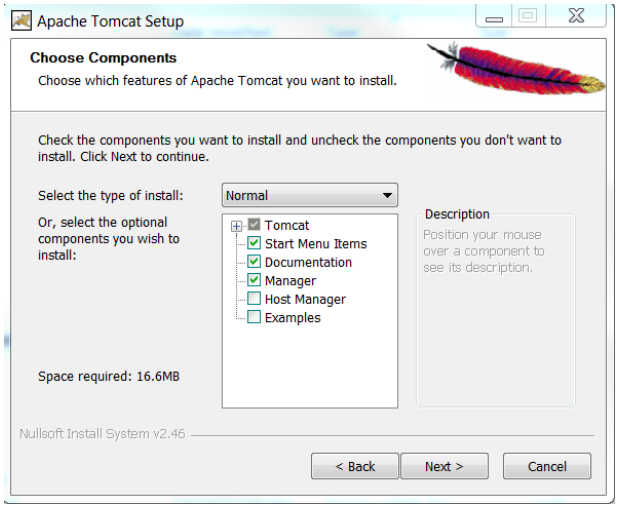


Figure B.6: Step-3

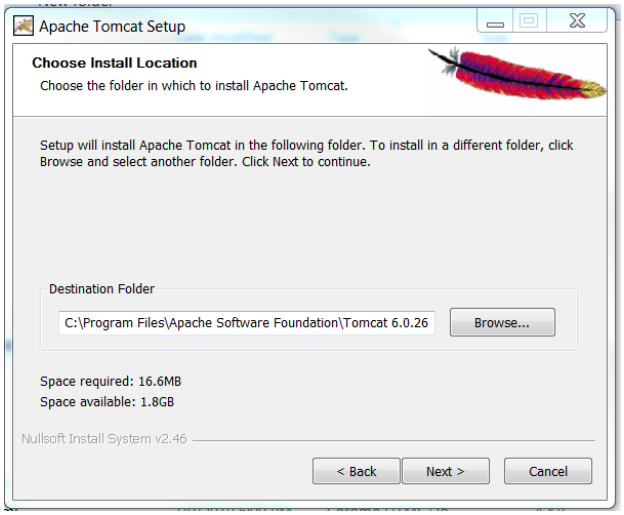


Figure B.7: Step-4

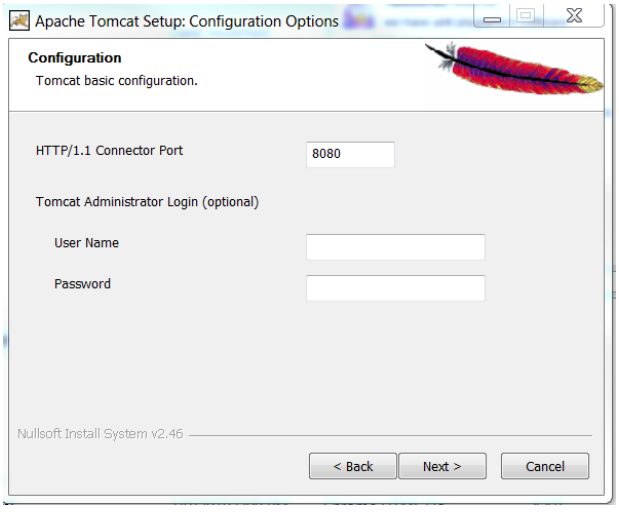


Figure B.8: Step-5

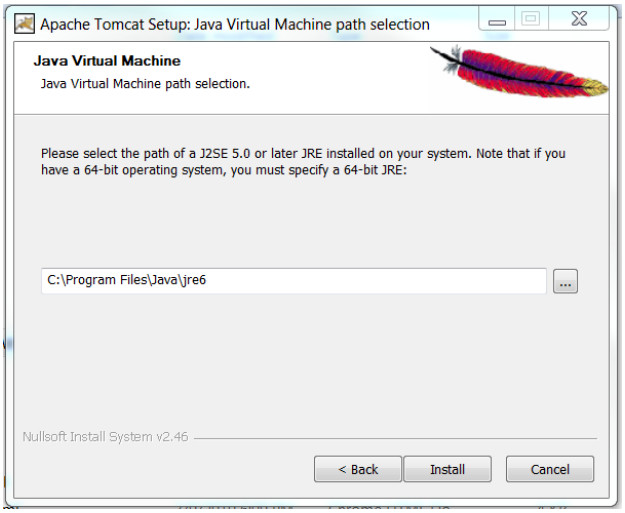


Figure B.9: Step-6

7. How to Use?

7.1 WQL with JSP

Following steps describe how to use WQL with JSP -

• Set the location of Wiki engine using setbase().

WQL.setbase("URL pointing to location of wiki that will be used") ;
• To send query strings to the interpreter, WQL.query() function is used. WQL.query accepts a string as input and returns the result of the query. If the query returns a list of nodes, it should be to a variable of correct datatype.

```
List<Node> a = WQL.query("getpage \"The Fountainhead\"") ;
WQL.query("createpage \"Test" "This is a test page"") ;
```

Be mindful of the fact that any ” inside the quotes will need to be escaped with a backslash.

• To print the data stored in Node objects of the list, iterate over each of the Nodes using a foreach iterator or a for loop.

```
for(Node i:a)
{       out.println(i);
}
```

The wiki text in its original format is returned by the the getText() method of Node class.

```
out.println(i.getText());
```

• To print the list of sections field of a Node object, user can do the following -

```
List<Node> a = WQL.query("");
for (Node temp:a)
{       List l = WQL.listlinks(temp);
}
for (int i = 0; i < l.size(); i++)
{       out.println(l.get(i))
}
```

WQL.listsections lists the top level sections. It doesn’t list the subsections. In case the user is interested in getting a subsection the user can perform another getsection query.

• All the links can be traversed using WQL.listlinks (). WQL.listlinks () takes as argument a Node and returns a list of links. The links themselves are stored as an array with the first one being the link and the second one being the text.

```
List<Node> a = WQL.query("");

for (Node  temp:a)
{       List l = WQL.listlinks(temp);
}
for  (int  i  =  0;  i  <  l.size();  i++)
{       out.println(l.get(i)[0])  //this  prints  the  link
        out.println(l.get(i)[1])  //this  prints  the  text  associated  with  the  link.
}
```

• One can utilise the whole power of JSP along with WQL. Forms, requests can be handled the usual way.

7.2 WSL

Using WSL is very convenient. Because the langauge is dynamically typed the chances for errors are a lot lesser. The code is cleaner and easier to understand without the burden of understanding the types. One needs to understand the process rather than the syntax. Let us see how one can use WSL to fetch data from Wiki.

• Set the location of Wiki engine using setbase ().
WQL.setbase("URL pointing to location of wiki that will be used")

•To execute WQL queries, query () function is used. If query returns something, it should be stored in a variable. Since, WSL is dynamically typed one does not have to worry about the data type pf the variable.

```
a = query(""" getpage "The Fontainhead"        """)
query(""" createpage "Test" "This is a test page" """)
```

The triple quotes ensure that you don't need to escape the" characters everytime they are used inside the query.

• If one simply wants to print result of the query, the function echo() does the trick. It handles multiple Node objects as well.

echo a

If you would like to print a special message you can iterate over the result of query() using the for each loop

```
for(x  in  a)
{           echo  "Test"
            echo  x
}
```

• To print the list of links present in a Node object, listlinks() can be used.

```
a  =  query(""" getpage ["The Fountainhead", "Atlas Shrugged"] """)
for  (x  in  a)
```

```
 d = listlinks(x)
for (y in d)
{       echo d[0] //print the link
        echo d[1] //print text associated with the link
}
```

• listsections() operates in the same fashion

7.3 Wiki Shell

We have built a command-line application known as Wiki Shell which provides an easy and a faster way to evaluate WSL expressions and get data from the Wiki. It enables users to do quick prototyping and testing with minimal fuss. To write complete WSL code, one needs to add HTML tags. However, if someone simply wants to check and validate the queries, Wiki Shell comes really handy as here one does not have to worry about adding HTML tags. Figure C.1 shows the welcome screen of Wiki Shell.

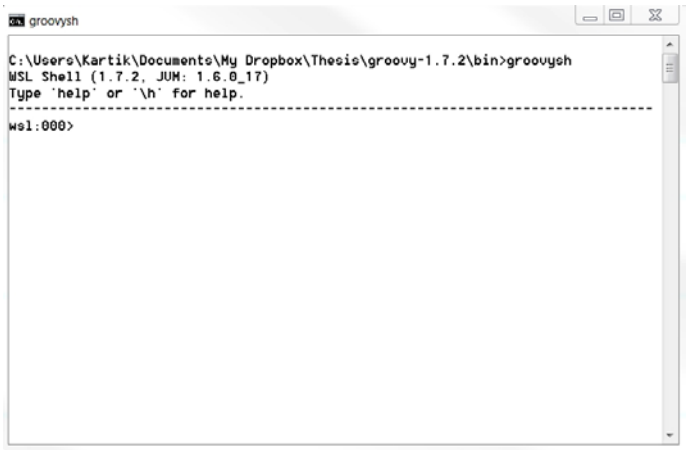


Figure C.1: Wiki Shell: Welcome Screen

Wiki Shell has been made by making changes in the Groovy Shell also known as groovysh. We added our functions to the Groovy Shell code and had to modify few things. Groovy Shell supports several command line options and arguments to control verbosity, ANSI colouring and other features. The Wiki Shell understands WSL syntax and evaluates expressions which are in WSL syntax only.

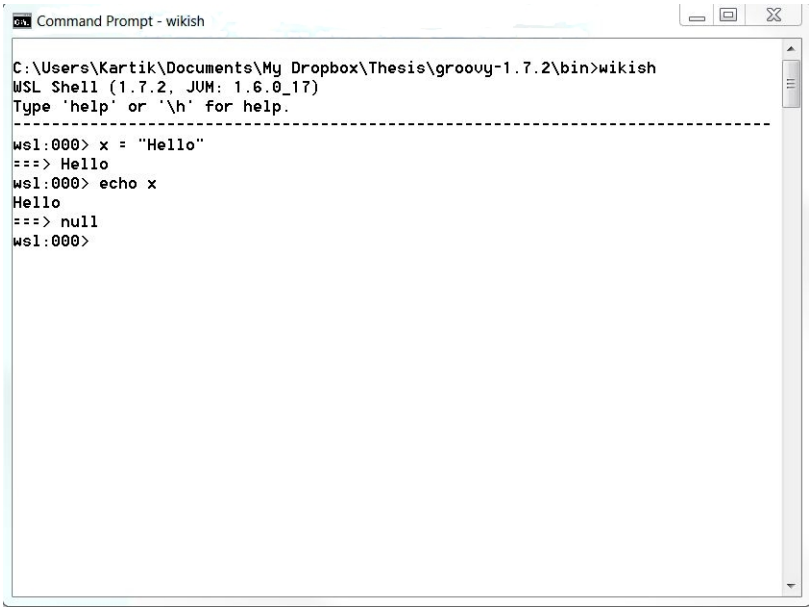
7.3.1 Variables

Wiki Shell variables are all untyped. This will set a shell variable -

```
wsl:000> x = “Hello”
```

Users can now use this shell variable in later expressions. For e.g. -

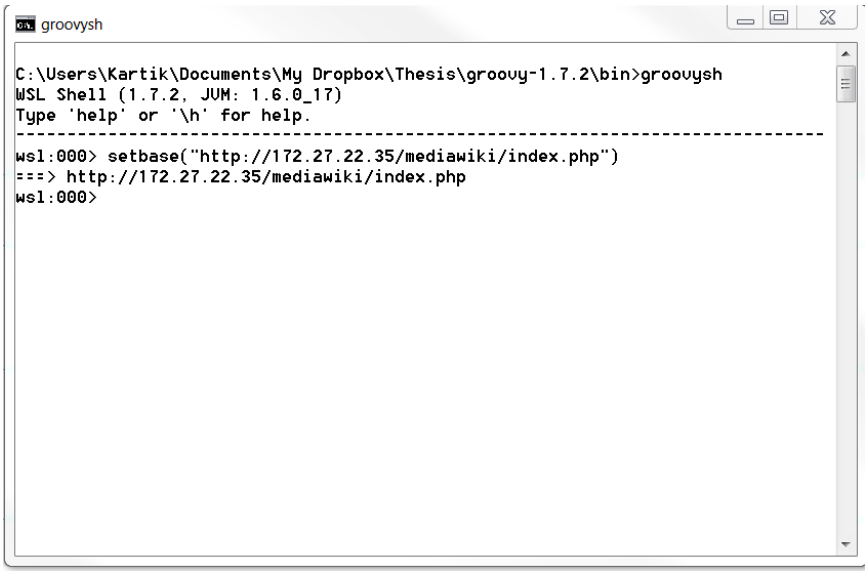
```
wsl:000> echo
x
```



7.3.2 Evaluate an Expression

Users can write WSL statements and when the complete expression is found , it is compiled and evaluated. The result is stored into the variable and the result of the expression is displayed in the shell. Following are the few examples -

• Setbase: wsl: 000>setbase (”Location of the wiki engine”)



• Running a query: wsl:000> a = query(“““ getpage “Himanshu Govil ” ”””). Result is shown in Figure C.4.

• Listsections:
Wsl: 000> for (b in a) d = listsections (a)
Wsl: 000> echo d
Result is shown in figure Figure C.5. Listlinks also works similarly.

```
        ...
wsl:000> for (b in a) {d = listsections(b) }
===> null
wsl:000> echo d
[ Designation ,  Contact Info ,  Companies ,  Recent Updates ,  To Do List ]
===> null
wsl:000>
```

Figure C.5: listsections

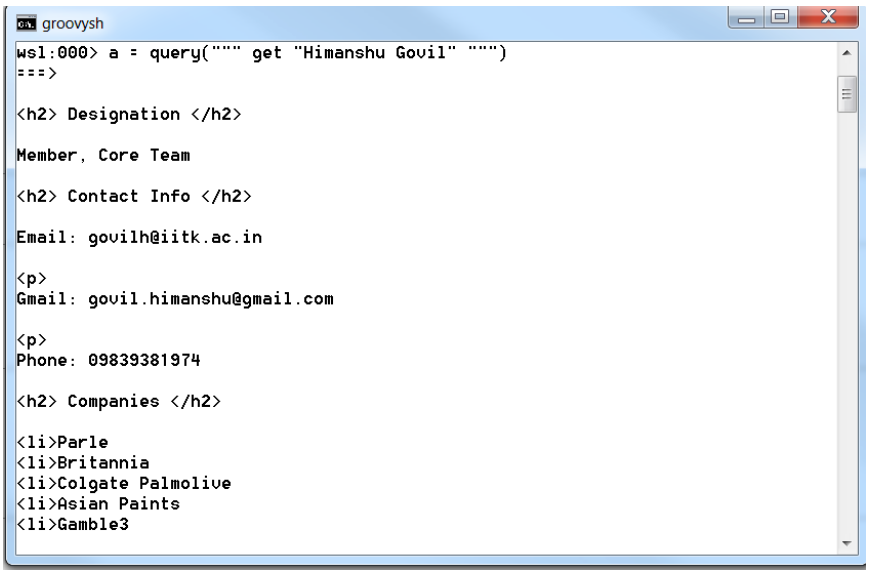


Figure C.4: Query

7.3.3 How to get started ?

• Download the groovyshell zip file that we have provided.

• Extract the same in a directory/folder of your choice. The extracted folder will look like as follows -

• Copy the .groovy folder present in this folder into C:\Users\ <User Id> in caseof Windows. In case of linux copy into /home/<username>

• Open command prompt or the terminal.
• Change directory to the location where groovyshell folder has been extracted.
• Go to the bin folder present in that folder.
• Run wikish and you are ready to go.

8. Conclusions and Future Work

8.1 Conclusions

A Wiki engine as a component in the technology stack enables developers to make complex web applications in an easy and convenient way. Wiki will become a platform and lots of web applications can be built using Wiki engines. It will find its usage in many different scenarios. It will definitely reduce the development time as time spent on database designing and form creation is saved. Moreover, it is easier to edit information in a Wiki page and this ensures that the web applications will always have the current data. Using Wiki to build web applications will eliminate the need of having a database administrator as Wiki itself is a web application which can be accessed through a web browser and anyone can maintain it.

8.2 Future Work

Many times actions are such that they involve many independent sub-actions. Consider this example - getpage ["The Fountainhead", "Atlas Shrugged"]. Now in this case, Wiki Interpreter needs to download two wiki pages which can be done in parallel as both the things are independent of each other. Therefore, parallel programming paradigms could be incorporated in the implementation of Wiki Interpreter. Independent actions can then be performed in parallel by making use of threads. Each thread after completing the job assigned to it can write back the result into the list of Nodes. This will increase the throughput and will make applications faster.

Moreover, the list which we are using for storing the results is sequential. Consider that two different threads have completed the job and now both wants to write into the list. Since, the list is sequential only one of them can write at a time and the other thread has to wait. The threads will also have to wait if some other thread is reading the list. To overcome this issue, concurrent linked list can be used instead of the sequential one. This will enable the threads to access the list concurrently and will further decrease the time time required in extracting the required information from the Wiki engine.

Wiki also allows users to upload files links to which are present in Wiki pages and viewers can download these files from these links. Our current implementation does not support this feature of Wiki. So the implementation can be extended to take care of the uploaded documents.

Bibliography

[1] David F. DelGreco, “The World-Wide Web: a quicktour”, ACM SIGLINK, vol. 2, issue 2, pp. 8-9, September 1993. http://portal.acm.org/citation.cfm?id=164399.164405.

[2] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, Arthur Secret, “The World Wide Web”, Communications of the ACM, vol. 37, issue 8, pp. 76-82, August 1994. http://portal.acm.org/citation.cfm?id=179606.179671.

[3] Sven Ziemer, “An Architecture for Web Applications”, Essay in DIF 8914 Dis- tributed Information Systems, November 2002.

[4] Weiquan Zhao and David Kearney, “Deriving Architectures of Web-Based Appli- cations”, Proceedings of the 5th Asia-Pacific web conference on Web technologies and applications, pp. 301-312, 2003

[5] http://news.netcraft.com/archives/2010/02/22/february 2010 web server survey.html

[6] D. A. Keim, H.P. Kriegel, “Issues in Visualizing Large Databases”, Proc. Int. Conf. on Visual Database Systems (VDB-3), March 1995.

[7] Anja Ebersbach, Markus Glaser, Richard Heigl, Alexander Warta, “Wiki Web Collaboration”, Springer, 2005

[8] Tapscott. D. and Williams, A.D. 2007 Wikinomics. How Mass Collaboration Changes Everything. Penguin. NewYork.

[9] Leuf, B. and Cunningham, W. The Wiki Way, Addison Wesley, Longman, 2001.

[10] Nicole Schadewitz and Norhayati Zakaria, “Cross-cultural Collaboration Wiki Evolving Knowledge about International Teamwork”, Proceeding of the 2009 in- ternational workshop on Intercultural collaboration, pp. 301-304, 2009.

[11] http://www.mediawiki.org/wiki/How does MediaWiki work

[12] Peter Buneman, “Semistructured data”, Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp. 117-121, 1997

[13] Serge Abiteboul, “Querying Semi-Structured Data”, Proceedings of the 6th Inter- national Conference on Database Theory, pp. 1-18, 1997

[14] http://www.w3.org/XML

[15] http://www.w3.org/QA/2002/04/valid-dtd-list.html

[16] Eric T Freeman, Elisabeth Robson, Bert Bates and Kathy Sierra, “Head First Design Patterns”, O’Reilly Media, October 2004, first edition.

[17] http://www.antlr.org/about.html