Master Thesis

Integration of
Light Weight Workflow Management System
With Content Management System
Using Web Services

Saarland University
Faculty 6 – Natural Sciences and Technology I
Department 6.2 – Computer Science

By

VARADARAJULU REDDY PYDA

Under the guidance of

Prof. Dr.-Ing. Gerhard Weikum
AG5 Databases and Information Systems Group
Max-Planck-Institut für Informatik
Saarbruecken.

Saarbruecken, July 2004.
Statement Under Oath

I confirm under oath that I have written the thesis on my own and that I have not used any other media that ones mentioned in the thesis

Saarbrucken, the 22nd September 2004  VARADARAJULU REDDY PYDA
Dedicated to my Parents & Sisters
ACKNOWLEDGEMENTS

At the outset, let me convey my whole-hearted gratitude to Prof. Gerhard Weikum, Chair of AG 5, Max-Planck Institute for Informatics, Saarbruecken, for kindly permitting me to undertake this project and also for offering me invaluable suggestions through constant guidance for the completion of my project. I consider it my good fortune to work under his guidance known for his academic excellence through his innovative methods of teaching to tap and develop the hidden talents in his students promoting their intellectual advancement and professional competence. Inspite of his busy schedule, he participated in all the discussions and presentations relating to my project offering me useful suggestions for modifications wherever necessary and also for evaluating each and every point to improve the quality and standard of my thesis pertaining to my project. I consider it my bound and duty once again to thank profusely, Prof.Gerhard Weikum for his cardial treatment, personal support, encouragement and proper guidance, but for him it would not have been possible for me to complete the thesis precisely and promptly.

I am highly indebted and thankful to Mr. Matthias Bender, Supervisor and guide for his continuous supervision and invaluable guidance for the betterment of my thesis pertaining to master’s degree programme. He was kind enough to spend his precious time with me personally providing useful comments and suggestions for carrying out my project in general and assisting me in great extent in the preparation of documentation.

I am extremely grateful to Mr. Stephen Kraus, C.E.O., DeepWeb GmbH, for his proper guidance and encouragement throughout the project. His continuous involvement and personal interest boosted my morale conducive to better and efficient performance. His proper guidance to work according to planned time schedule, which prompted not only presentation of preliminary project report but also the submission of the entire thesis at the right time. But for his personal persuasion and motivation it would not have been possible for me to present the preliminary project report and submit the thesis as per schedule.

I acknowledge with utmost gratitude the personal involvement and selfless service rendered by Mr. Carsten Lex in providing administrative assistance inevitable from the commencement up to the completion of my project.
I am thankful to my partner Mr. Sai Pradeep, Vangala for his whole hearted support and continuous assistance necessary at every stage for the timely completion of my project and submission of thesis in time.

I owe a debt of gratitude always to my (parents) 1) father, Mr. Vijayamohan Reddy, Pyda 2) my mother, Mrs. Vishnupriya, Pyda for their continuous encouragement and invaluable assistance through out my academic career providing mental peace without involving me in domestic problems. I am obliged and grateful forever to my sisters Ms. Praveena, Pyda, Ms. Prathyusha, Pyda, Ms. Vidya nandini, Mrs. Anuradha, Ms. Chaitanya, and Mrs. Annapoorna for inspiring me to choose study of computer science and helping me periodically to boost my confidence through continuous contact. I would like to specially thank my brothers Mr. Parthasarathy and Mr. Karthikeyan Reddy for his constant support and encouragement in all the matters that I indulge in.

Last but not the least, I would be failing in my duty if I do not convey my sincere thanks to my colleagues, friends and well wishers viz, 1) Mr. Sreedhar, Elisetty, 2) Mr. Srikanth, Ramaka, 3) Mr. Venkatakrishna, Thaticherla, 4) Mr. Ravinder, Indarapu, 5) Mrs. Aravind 6) Mr. Rahul, Ray 7) Mr. Debopriyo, Majumdar 8) Srinivasulu Reddy Mekapothu 9) Mr. Sunil Kumar, Chittiboyana, 10) Mr. Balaji, A.G 11) Mr. Mohan Krishna, Gavvaji 12) Mr. Subramanyam, Nadavala 13) Mr. Hareesh Nair for their goodwill, generosity and assistance whenever required by creating a congenial atmosphere for the successful completion of my Master’s degree programme.
# TABLE OF CONTENTS

**CHAPTER 1 INTRODUCTION** ........................................................................................................................................................................ 1

**CHAPTER 2 PRELIMINARIES** ........................................................................................................................................................................... 2-4

  2.1 WORKFLOW TERMINOLOGY ........................................................................................................................................................................ 2-4
  2.2 WORKFLOW MANAGEMENT REFERENCE MODEL .................................................................................................................................. 2-5
    2.2.1 Process Definition Tools ........................................................................................................................................................................ 2-5
    2.2.2 Administration and Monitoring .............................................................................................................................................................. 2-5
    2.2.3 Workflow Client Applications .............................................................................................................................................................. 2-5
    2.2.4 Invoked Applications ............................................................................................................................................................................. 2-5
    2.2.5 Workflow Engines .............................................................................................................................................................................. 2-5
  2.3 PROCESS DEFINITION LANGUAGES ...................................................................................................................................................... 2-6
    2.3.1 Petri nets ................................................................................................................................................................................................. 2-6
    2.3.2 State and Activity charts ...................................................................................................................................................................... 2-7
    2.3.3 XPDL (XML Process Definition Language) ...................................................................................................................................... 2-8
    2.3.4 BPEL4WS (Business Process Execution Language for Web Services) ............................................................................................... 2-9
  2.4 OVERVIEW OF WORKFLOW MANAGEMENT SYSTEMS ....................................................................................................................... 2-10
    2.4.1 Characteristics of a WfMS ................................................................................................................................................................. 2-10
    2.4.2 Types of WfMS .................................................................................................................................................................................. 2-12
    2.4.3 State-of-the-art WfMS .................................................................................................................................................................... 2-13
  2.5 OVERVIEW OF CONTENT MANAGEMENT SYSTEMS ............................................................................................................................ 2-15
    2.5.1 Definition .............................................................................................................................................................................................. 2-15
    2.5.2 Principal Components of CMS .......................................................................................................................................................... 2-15
    2.5.3 Functions of CMS ............................................................................................................................................................................. 2-16
    2.5.4 Benefits of CMS ............................................................................................................................................................................... 2-17
    2.5.5 State-of-the-art CMS .................................................................................................................................................................... 2-18
  2.6 SERVICE ORIENTED ARCHITECTURES .................................................................................................................................................. 2-19
    2.6.1 Characteristics & Benefits of SOA ..................................................................................................................................................... 2-20
    2.6.2 Web Services .................................................................................................................................................................................... 2-20

**CHAPTER 3 DESIGN** .................................................................................................................................................................................... 3-26

  3.1 DEEPWEB CMS OVERVIEW ...................................................................................................................................................................... 3-26
    3.1.1 Features of Deepweb CMS ............................................................................................................................................................ 3-26
    3.1.2 Structural overview of Deepweb CMS ........................................................................................................................................... 3-29
    3.1.3 User roles of Deepweb CMS .......................................................................................................................................................... 3-32
  3.2 MENTOR-WS OVERVIEW ...................................................................................................................................................................... 3-33
    3.2.1 Features of Mentor-WS ................................................................................................................................................................... 3-34
    3.2.2 Structural overview of Mentor-WS ........................................................................................................................................... 3-34
    3.2.3 Components of Mentor-WS .......................................................................................................................................................... 3-35
  3.3 MOTIVATION ............................................................................................................................................................................................ 3-38
  3.4 CONCEPTUAL DESIGN OF THE PROPOSED SYSTEM ............................................................................................................................ 3-39
    3.4.1 Design Goals ..................................................................................................................................................................................... 3-39
    3.4.2 Modeling alternatives .................................................................................................................................................................... 3-40
    3.4.3 Proposed System Architecture .................................................................................................................................................. 3-41

**CHAPTER 4 IMPLEMENTATION** ................................................................................................................................................................. 4-44

  4.1 SYSTEM OVERVIEW ................................................................................................................................................................................. 4-44
    4.1.1 Main System Components Overview ............................................................................................................................................ 4-44
    4.1.2 Working Principle Overview .......................................................................................................................................................... 4-47
  4.2 WORKING PRINCIPLE IN DETAIL ..................................................................................................................................................... 4-49
    4.2.1 Design phase ..................................................................................................................................................................................... 4-49
    4.2.2 Execution phase ................................................................................................................................................................................. 4-57
LIST OF FIGURES

FIGURE (I): CMS FUNCTIONAL SCOPE AND CONTENT LIFE CYCLE ......................................................... 3
FIGURE (II): WORKFLOW MANAGEMENT SYSTEM – REFERENCE MODEL ........................................ 2-6
FIGURE (III): SIMPLE STATE CHART AND ITS STATE TREE REPRESENTATION .......................... 2-8
FIGURE (IV): A XPDL REPRESENTATION AND ITS CORRESPONDING GRAPH-STRUCTURE ........ 2-9
FIGURE (V): CHARACTERISTICS OF WORKFLOW MANAGEMENT SYSTEMS .................................. 2-11
FIGURE (VI): PRINCIPAL ELEMENTS OF A CONTENT MANAGEMENT SYSTEM .............................. 2-15
FIGURE (VII): GENERIC ARCHITECTURE OF SOA ........................................................................ 2-21
FIGURE (VIII): DEEPWEB CMS FUNCTIONAL SCOPE AND CONTENT LIFE CYCLE .................... 3-31
FIGURE (IX): MENTOR-WS ARCHITECTURE .............................................................................. 3-37
FIGURE (X): NOVEL DEEPWEB CMS FUNCTIONAL SCOPE AND CONTENT LIFE CYCLE ............ 3-42
FIGURE (XI): SYSTEM ARCHITECTURE ...................................................................................... 4-45
FIGURE (XII): WORKFLOW ENTITIES .......................................................................................... 4-50
FIGURE (XIII): MODELING WORKFLOW PROCESSES .................................................................. 4-51
FIGURE (XIV): SAMPLE XPDL FILE ............................................................................................ 4-52
FIGURE (XV): SAMPLE STATEMATE FORMAT ............................................................................... 4-53
FIGURE (XVI): MAPPING ELEMENTS BETWEEN XPDL AND STATEMATE FORMAT .................... 4-54
FIGURE (XVII): WEB SERVICE INTERFACE FOR MENTOR-WS WORKFLOW SERVER .............. 4-55
FIGURE (XVIII): MENTOR-WS WORKFLOW SERVER *WSDL ...................................................... 4-55
FIGURE (XIX): A SOAP REQUEST MESSAGE .............................................................................. 4-56
FIGURE (XX): JSP SERVICE JAVA CLIENT CLASS .......................................................................... 4-57
FIGURE (XXI): A WEB SERVICE JAVA CLIENT CLASS .................................................................. 4-58
FIGURE (XXII): XML LOG FILE .................................................................................................. 4-58
FIGURE (XXIII): NEWSSTICKER PROCESS DEFINITION .............................................................. 4-60
FIGURE (XXIV): WORKFLOW MANAGER AND WORKLIST MANAGER ....................................... 4-61
FIGURE (XXV): AVAILABLE ROLES, AVAILABLE ACTORS AND ACTORS VS ROLES .............. 4-62
FIGURE (XXVI): DEEPWEB CMS REGISTRATION PAGE ............................................................... 4-63
FIGURE (XXVII): SAMPLE SCENARIO – CONTENT PUBLICATION ............................................... 4-64
FIGURE (XXVIII): MENTOR-WS PACKAGE HIERARCHY ................................................................ 69
FIGURE (XXIX): FOUR IMPORTANT METHODS OF DSI_SERVER CLASS ....................................... 70
FIGURE (XXX): CODE SNIPPET FOR EXTWRAPPER CLASS .......................................................... 72
FIGURE (XXXI): IMPORTANT METHODS OF JAWEMENTOR PACKAGE ....................................... 76

LIST OF TABLES

TABLE 1 CLASS SUMMARY FOR ORG.MENTOR.DSISERVER PACKAGE ........................................... 71
TABLE 2 CLASS SUMMARY FOR ORG.MENTOR.EXTWRAPPER PACKAGE ....................................... 72
TABLE 3 CLASS SUMMARY FOR ORG.MENTOR.WORKFLOW PACKAGE ........................................ 76
TABLE 4 CLASS SUMMARY FOR ORG.MENTOR.JAWEMENTOR PACKAGE ....................................... 78
TABLE 5 ESSENTIAL LIBRARIES FOR MENTOR-WS ........................................................................ 79
TABLE 6 SOFTWARE REQUIREMENTS FOR MENTOR-WS ............................................................. 81
CHAPTER 1 INTRODUCTION

The rapid developments in the field of information technology of the late twentieth century have brought relentless changes in the way organizations operate and revolutionized the companies’ approach towards their clients and competitors. The advent of cyber era is considered the first and foremost technological development in the last decade. The rapid growth of internet is triggered by the recent technological advances in the field of computing and networking [Wes99]. People started to visualize the internet not just, as a medium for information and communication exchange, but as a medium for performing their day-to-day activities (banking, shopping etc.,) under the same roof with much ease. Parallel to these developments, enterprises are forced to participate in the resulting global competition while trying to fulfill the drastically increasing demands of the customers. To cope up with the global competition, business enterprises are obligated to streamline their organizational structures as well as their way of doing business. In this context, often a process-centered view is employed which focuses on the business processes carried out within these organizations [Ati01]. A Business process is defined as [HC94] “a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer”. In business processes, different processing entities (humans, software systems) have to perform tasks in a coordinated way to accomplish a specific goal. Since organizations normally operate in dynamic environments, their business processes have to be constantly evaluated and optimized to fulfill the varying requirements. To accomplish this, enterprises have to utilize the services of information technology to buttress their business processes.

Typically, only a small number of reoccurring, well-defined processes are responsible for most of a company's workload. This fact gave rise to the intuition of automated execution of business processes, which could minimize the execution costs (Business process reengineering, BPR [Sch99]). To address the above stated problems, workflow management systems (WFMS) came into existence and received a great deal of attention in the recent past [Han2001]. Workflow management is a technology that captures formal descriptions of business processes and supports the automatic enactment of processes based on these formal descriptions. Most of the WFMSs that exist today follow a process-centric approach in contrast to the traditional data-centric approach, which focuses on data requirements as an organizing concept for computer applications [Jon00]. This process-centric approach concentrates on the work from a broader perspective by integrating all related programs, processes, functions, data, documents, persons, organizations etc. This enables the optimization of processes to minimize the costs and increase the efficiency [JB96].
On the other hand, this approach restricts the application of WfMSs to intra-organizational applications. Nowadays, enterprises and organizations have to collaborate with other enterprises and organizations in order to fulfill the requirements of global competition. Workflow interoperability is considered as the pivotal issue in implementing the inter-organizational workflows. Service Oriented Architectures (SOA) offered an attractive solution in accomplishing this goal. Even though technologies such as CORBA and COM introduced the services-oriented distributed concepts, they could never become the universal choice of enterprises and developers due to their tight coupling and platform dependency, respectively. Web Services [ACK+04] have become a key to the service oriented internet world in a very short time period because they promise the interaction of unknown and unrelated applications on the Internet in a more standardized and simpler way than any other approach ever claimed [Man04].

As part of this global economic revolution, companies started to offer their services through their websites. The greatest asset of a company’s website is, without a doubt, its content. Managing the website’s content efficiently is a key goal even for smaller companies. A Content Management System (CMS) is a system that empowers companies to take control of it. The advent of CMS has now enabled users who are familiar with regular word processing to create and publish content in the internet themselves, without the intervention of specialists. This feature of CMS accelerated the publishing process and help companies to gain a competitive edge over their competitors. However, CMSs in turn have to perform a wide array of business processing in accomplishing their goal. Therefore, the integration of automated workflows is seen as the key factor in augmenting the efficiency of content management system.

The process of publishing content can be complex. Typically, participants pass content between the authors, developers, reviewers, editors, translators, approvers, and publishers. However, workflow automation in a CMS is not only about using software to facilitate or automate the transfer of content from person to person, but also about tracking and recording the progress of an activity, delivering the work to an appropriate users, archiving work when necessary, and providing a framework for actions to take place in predefined scenarios. Figure (I) [Han2001] shows the various stages of a Content Management System with an embedded workflow.
In this thesis, we will outline the innovative architecture of a lightweight, distributable and scalable WfMS \textit{(Mentor-WS)}, which is an enhanced version of the Mentor-Lite [GWS+2000] developed by Max Planck Institute for informatics, Germany. We will demonstrate the application of web services as a middleware in Mentor-WS to elucidate the workflow interoperability in inter-organizational as well as in intra-organizational workflows. We will also present a novel approach for integrating Mentor-WS with a Content Management System \textit{(DeepWeb CMS)} developed by DeepWeb GmbH, Germany. We will demonstrate the application of automated workflows at different stages of a CMS functional scope and explore the various features of such an integrated CMS.

The remainder of this thesis is organized as follows: Chapter 2 introduces the basics of WfMSs, CMSs and SOAs. Chapter 3 provides a comparative study and an evaluation of the existing system (Mentor-lite) and sketches the design goals, and the proposed architecture of Mentor-WS. Chapter 5 focuses on various implementation aspects of our prototype. Finally, Chapter 6 concludes this work and presents an outlook on future enhancements of the prototype.
CHAPTER 2 PRELIMINARIES

2.1 Workflow Terminology

Workflow is defined as “the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules” [WfMC96]. A business process, as defined in [WfMC94b], is “a coordinated (parallel and/or serial) set of process activities that are connected in order to achieve a common business goal”.

A business process is managed within a process definition and executed by a workflow management system (WfMS). A WfMS can be defined as [Dav95] “A System that completely defines, manages and executes “workflows” through the execution of software, whose order of execution is driven by a computer representation (process definition) of the workflow logic”. “A process definition is the representation of business process in a form which supports automated manipulation, such as modeling, or enactment by a workflow management system (WfMS). The process definition consists of a network of process activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual workflow instances. “A workflow instance is the representation of a single enactment of a process, or activity within a process, including its associated data”. “A process activity is a logical step or description of a piece of work that contributes towards the accomplishment of a process”. An activity can involve manual interaction with a workflow participant (manual process activity), or the activity might be executed using machine resources (automated process activity).

The languages that are used to express process definitions are called process definition languages. Process definition languages aim at capturing workflow-relevant information (data that is used by a WfMS to determine the state transitions of a workflow instance) of application processes and the environment in which application processes are executed, with the aim of their controlled execution by a WfMS. These languages may or may not provide a graphical notation. The common idea of these languages is to provide an intelligible process-view to non-programmers such as business analysts, re-engineering consultants, end-users and supervisors. Additionally, these graphical representations can be used for business process re-engineering (BPR) when benchmarking or optimizing existing processes.
2.2 Workflow Management Reference Model

The Workflow Reference model has been developed from the generic workflow application structure by identifying the interfaces within this structure, which enable products to interoperate at a variety of levels [WfMC95]. Figure (II) [WfMC95] depicts the major components and interfaces inside the workflow architecture.

2.2.1 Process Definition Tools

A graphical or textual tool used for defining the business process. The process definition consists of specific activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities such as participants, associated IT applications and data, etc. It serves as a template to create specific instances of a business process used by workflow engines to control processing.

2.2.2 Administration and Monitoring

This Component manages the Workflow Services, Configuration, Management, and controls the overall workflow environment. Some of the basic functions of this component include establishment of users, assignment of work items, exception and error processing, event generation and notification, auditing, tracking and reporting of results, tracking and reporting of statistics, versioning and change management.

2.2.3 Workflow Client Applications

These applications interact with WFMS to utilize workflow services and the interaction with the workflow engine is done through an interface (work-list) and a queue of assigned work items.

2.2.4 Invoked Applications

A Software application invoked by the workflow engine to perform processing synchronously or asynchronously.

2.2.5 Workflow Engines

This Component provides a runtime execution environment and manages the overall processing and the execution of workflow process instances. The basic functions of this component include the interpretation of business process specification, creation of new process instances, execution and management of instances, navigation between work items, management/control of workflow information, routing data between instances.
FIGURE (II) WORKFLOW MANAGEMENT SYSTEM – REFERENCE MODEL

2.3 Process Definition Languages

The main purpose of a WfMS is the support of the definition, execution and control of the processes. Since processes play a significant role in the functioning of WfMS, it is indispensable to use an established framework for modeling and analyzing workflow processes [HL91, Kou95, and Law97]. The following are some of the approaches used to model the workflow definitions.

2.3.1 Petri nets

The notion of classical Petri nets was invented by Carl Adam Petri in the sixties [Pet62]. The classical petri net is a directed bipartite graph with two node types called places and transitions. The nodes are connected via directed arcs. Connections between two nodes of the same kind are not allowed. Petri nets have been used to model and analyze all kinds of processes with applications ranging from protocols, hardware, and embedded systems to flexible manufacturing systems, user interaction, and business processes [Aal00]. The last two decades have seen many extensions to the classical petri nets, which facilitated the modeling of complex processes, where data and time are important factors [Aal94, Jen96].

The graphical nature, formal semantics, expressiveness, tool-independent framework and the availability of many analysis techniques of petri nets have established petri nets as a workflow-modeling framework [Aal00]. However, the classical petri nets tend to be complex while describing real processes. To address these problems, the classical petri net is extended (high-level petri net) with color to model data, time and hierarchy to structure large models [Jen96, Aal94, Hee94].
Modeling a workflow process definition in terms of a petri net is rather straightforward: activities (tasks) are modeled by transitions and conditions are modeled by places. A petri net which models a workflow process definition (i.e. the life cycle of one case in isolation) is called a WorkFlow net (WF-net). A WF-net satisfies two requirements. First, a WF-net has one input place ($i$) and one output place ($o$). A token in $i$ corresponds to a case, which needs to be handled, a token in $o$ corresponds to a case, which has been handled. Secondly, in a WF-net there are no dangling tasks and/or conditions. Every task (transition) and condition (place) should contribute to the processing of cases. Therefore, every transition $t$ (place $p$) should be located on a path from place $i$ to place $o$. The latter requirement corresponds to strongly connectedness if $o$ is connected to $i$ via an additional transition $t^*$. A formal description of high-level petri nets and their applications to workflow management is provided in [Aal00].

2.3.2 State and Activity charts

State charts have been used as a general-purpose specification formalism for dynamic systems and are pursued in various research projects because of their excellent basis for providing formal proofs. The method of state and activity charts is perceived by practitioners as more intuitive and easier approach to learn than petri nets [DG97]. The model of state and activity charts was proposed by Harel et al. [Ha87a] and was originally developed for reactive systems. Due to their visualization, rigorous semantics, interoperability with other tools and methods and wide acceptance in industry (UML), state charts are used as a workflow specification method.

The model comprises two dual views of specification [DG97]. Activities map directly to the activities of a process definition. They describe the functional decomposition of a system and represent the active components of a workflow specification. The dataflow between various activities is specified in an activity chart in the form of a directed graph with data items represented as arc annotations. The control flow between activities is specified in a state chart, which reflects the behavior of a system. A state chart is a finite state machine with a distinguished initial state and transitions driven by Event-Condition-Action rules (ECA rules). ECA rules are often written as $E[C]/A$. Each transition arc between states is annotated with an ECA triple. A transition from state $X$ to state $Y$ fires, if the specified event $E$ occurs and the specified condition $C$ holds. The result is that state $X$ is exited, state $Y$ is entered and the specified action $A$ is executed. Conditions and actions are expressed in terms of data item variables. An action $A$ can explicitly start or stop an activity and can generate an event $E$ or set a condition $C$. Each of the three components may be empty. If all components are empty, then a transition can fire instantly,
changing a state without any precondition. Every state change in a state chart induces a discrete time dimension.

State charts have two important features namely, *nested states* and *orthogonal components*. Nested states are states, which can themselves contain an entire state chart whereas the orthogonal components denote the parallel execution of two state charts that are embedded in the same higher-level state.

**FIGURE (III): SIMPLE STATE CHART AND ITS STATE TREE REPRESENTATION**

The left portion of the figure (III) [DG97] shows a simple state chart and the right portion shows its corresponding state tree representation. The state chart consists of states from A through I, where the initial states are marked by red arcs without source. The orthogonal components are separated by a dashed-line with their corresponding higher-level state. In the above figure, States C and F are orthogonal states inside a nested state B whereas the states B, D, G and H are initial states. A more detailed formal description about state and activity charts can be found in [DG97].

**2.3.3 XPDL (XML Process Definition Language)**

XPDL uses XML to describe a business process. XPDL provides organizations with a common framework for implementing business process management and workflow engines, and for designing, analyzing, and exchanging business processes. XPDL is based on the recognized workflow standard, WfMC reference model [WfMC95].

XPDL is conceived of as a graph-structured language with additional concepts to handle blocks. Scoping issues are relevant at the package and process levels. Process definitions cannot be nested. Routing is handled by specification of transitions between activities. The activities in a process can be thought of as the nodes of a directed graph, with the transitions being the edges. Conditions associated with the transitions determine the execution of appropriate activity or activities at runtime. The left portion of the figure (IV) shows the definitions of various activities and transitions involved in the workflow process whereas the right portion of the figure depicts its
corresponding graph-structure. All activities from starting from A (Start Activity) through D (End Activity) are represented as nodes and the transitions are represented as the edges between them. Conditions and condition types (XOR, AND) associated with the transitions decide the execution of activity. In the figure (IV), a transition from activity A to activity C will occur if the condition (GOTO_C) is satisfied. Activities E and F are the orthogonal components, as specified by the AND split type at B. A detailed description on specification of XPDL is provided in [WfMC02]. One of the key elements of the XPDL is its extensibility to handle information used by a variety of different tools. Based upon a limited number of entities that describe a workflow process definition (‘Minimum Meta Model’), XPDL thus supports a number of differing approaches [WfMC02]. XPDL is extendable, allowing the addition of elements and attributes as well as the import of data types and operations that are defined in other specifications. Moreover, XPDL provides a natural fit with graphical representations of business processes.

FIGURE (IV): A XPDL REPRESENTATION AND ITS CORRESPONDING GRAPH-STRUCTURE OF A SIMPLE WORKFLOW PROCESS

2.3.4 BPEL4WS (*Business Process Execution Language for Web Services*)

BPEL4WS is an XML based flow languages that provides a formal specification of business process behavior based exclusively on web services. It defines a model and grammar for describing the behavior of a business process based on interactions between the process and its partners. IBM, Microsoft, BEA Systems, SAP and Siebel Systems originally authored the BPEL4WS specification. BPEL4WS is a block-structured programming language, allowing recursive blocks, but restricting definitions and declarations to be top-level.
A BPEL4WS process definition is made up of four basic components: activities (subtasks of process), control links (define the process control flow), a series of partner-links (placeholders for service providers and process callers), variables (hold data). The optional components include the correlations sets, which provide support for process instance identification, the fault handlers that enclose activities performed in case of errors and the compensation handlers that enable compensatory activities in the event of actions that cannot be explicitly undone. The control flow is defined by all the possible activities as defined by BPEL semantics. The control flow is a nice hybrid model half-way between block structured and state transition control flow definitions. The model uses links to establish the dependencies between block definitions. The links are defined inside the flow and are used to connect a source activity to a target activity. Sequential and concurrent control between activities is provided by sequence and flow control constructs. Three types of activities handle the message flow: receive, reply and invoke. Receive and reply may be assigned to the same operation in case of request/response. Invoke is used to define a solicit response operation type. The information is passed between the different activities in an implicit way through the sharing of globally visible data containers. Containers may exchange specific data elements via the use of Assign statements. BPEL4WS allows long-running transactions to run as web services with better consistency and reliability. Like XPDL and state charts, BPEL provides rich visualization, interoperability with other tools and methods, rigorous semantics and wide acceptance in industry.

2.4 Overview of Workflow Management Systems

Workflow Management Systems are designed to enable the enactment of business processes, i.e. to extract and maintain the process logic between business applications. Within the WfMS, heterogeneous applications are connected through a given process definition. WFMS orchestrate the control and the data flow between a workflow's activities, based on a high-level specification of the intended behavior.

2.4.1 Characteristics of a WfMS

From a broader view, WfMSs may be characterized as providing support in three functional areas as depicted in the figure (V) [Dav95].

2.4.1.1 Build-time functions

The Build-time functions are those, which result in a computerized definition of a business process. During this phase, a business process is translated from the real world into process
definition using a process definition language. The process definition enables the interchange of workflow relevant data between the build-time tools and runtime products. Some workflow systems may allow dynamic alterations to process definitions from the run-time operational environment, as indicated by the feedback arrow in the diagram.

### 2.4.1.2 Run-time control functions

In this phase, the process definition is interpreted by workflow engine, which is responsible for creating and controlling operational instances of the process, scheduling the various activities steps within the process and invoking the appropriate human and IT application resources, etc. These run-time process control functions act as an interface between the process as modeled within the process definition and the process as it is seen in the real world, reflected in the runtime interactions of users and IT application tools.

### 2.4.1.3 Run-time interactions

During this phase, individual activities within a workflow process interact with human operations (interactive activities), often implemented in conjunction with the use of a particular IT tool, or with information processing operations requiring a particular application program (automatic activities) to operate on some defined information. Further, Individual activities inside a workflow process interact with the process control software to transfer control between them, to ascertain the operational status of processes, to invoke application tools and pass the appropriate data, etc.

---

**FIGURE (V): CHARACTERISTICS OF WORKFLOW MANAGEMENT SYSTEMS**
2.4.2 Types of WfMS

Numerous segmentations for workflow management systems have been proposed reflecting the existing market structure. This section presents some common denominator of several popular segmentations.

2.4.2.1 Production Workflow Management System

The key goal of production workflow is to manage large numbers of similar tasks, and to optimize productivity. This can be achieved by automating the activities as practical and relentlessly pursuing more and more automation until Straight-Through-Processing has been achieved. These systems are capable of managing huge complex processes and integrating with existing (legacy) systems.

2.4.2.2 Autonomous Workflow Management System

An autonomous workflow management system is functional without any additional application software, with the exception of database management systems and message queuing middleware. These systems are the separate pieces of application software that provide the workflow functionality. The applications that are external to the workflow management system are invoked at runtime; and relevant data is passed between the workflow participants. The modeling of autonomous workflow applications requires the specification of interface information for the invoked applications, relevant data structures, and involved participants, and thus can become a complex challenge.

2.4.2.3 Embedded Workflow Management System

An embedded workflow management system is only functional if it is employed with the surrounding (embedded) system; for instance, an Enterprise Resource Planning (ERP) system. The workflow functionality of embedded workflow management systems is exhibited by the surrounding software system. Common examples include ERP systems, payment and settlement systems. The workflow components are used to control the sequence of the application's functions, to manage queues and to assist with exception processing.

2.4.2.4 Administrative Workflow Management System

The most important feature of an administrative workflow system is the ease to define the process. Typically, there are many definitions running concurrently and they tend to involve a large number of employees. Process definitions are usually created using forms and if the definition is too complex for the form then you just has to use another product. Flexibility is more
important than productivity, and these systems handle one or two orders of magnitude lower numbers of instances per hour than production WfMS.

2.4.2.5 Collaborative Workflow management system

Collaborative workflow system focuses on teams working together towards common goals. Groups can vary from small, project-oriented teams, to widely dispersed people with interests in common. Effective use of collaborative workflow to support team working is now considered a vital element in the success of enterprises of all kinds.

2.4.2.6 Ad-Hoc Workflow Management system

Ad-Hoc workflow systems allow to create and amend process definitions very quickly and easily to meet circumstances as they arise. Therefore, it is possible to have almost as many process definitions as there are instances of the definitions. Ad-Hoc workflow maximizes flexibility in areas where throughput and security are not major concerns. Its users own their own processes.

2.4.2.7 Dependable Workflow Technology

Workflow technology aims to support business processes that may span different, largely autonomous, organizational units within large enterprises or even multiple enterprises. Examples of such workflows would be the processing of company credit requests in an internationally operating bank or nation-wide health-care management. Workflow technology is also an important asset for Internet-based business-to-business (e.g., supply chains) and business-to-consumer (e.g., auctions) e-services.

2.4.3 State-of-the-art WfMS

2.4.3.1 MQSeries workflow (IBM)

MQSeries workflow offers a true object-oriented design together with a high level of re-usability. MQSeries workflow concentrates on procedure management and powerful organization modeling while leaving activity implementation to the programmer. Activities are complex and must be programmed to fit requirements. It uses process model, a graph-based workflow language for specifying the workflows. By using an MQSeries runtime client, users can start the execution of the modeled processes, monitor and manipulate the running process instances, change the work list view by filtering or sorting the work items and transfer work items to work lists of other users (change work assignments). However, dynamic adaptations of running workflow instances are not addressed in MQSeries workflow. Workflow engine can directly co-operate with applications
Chapter 2 – Preliminaries

and other workflow engines through XML encoded MQSeries messages. MQSeries Workflow provides a process automation system for managing people, data, applications, and business processes throughout an organization, including external partners via the Internet or intranets and extranets. With this approach to workflow, IBM provides a robust infrastructure of messaging (MQSeries Messaging) and application integration (MQSeries Integrator) upon which production-level workflow applications are built. MQSeries Workflow helps both business and IT users effectively leverage their existing infrastructure for process automation and management.

2.4.3.2 Bizflow 2000 (Handysoft)

Bizflow is a flexible workflow engine that provides well designed graphical definition tool associated to a form definition tool BizForm. It has a comprehensive Web based interface as well as a client/server interface based on ActiveX objects. The activity interface is generated from the information given at process definition level, together with applications in the form of documents that can contain scripts to access process context. It has a fair organization model and set of dispatching rule. It is capable of handling dynamic adaptations of running workflow instances. It provides flexible process definition, low cost application definition through forms and interpreted interface and deployment of large administrative and light production applications over the internet.

2.4.3.3 ADEPT

The ADEPT project at the University of Ulm aims at providing flexible workflow support for advanced applications, mainly in the context of hospital environments [RD97, RD98]. The ADEPT build time components enable the definition and management of WF templates (ADEPT workflow editor), the description of inter-WF dependencies (ADEPT interaction editor), the modeling of organizational entities (ADEPT Organization modeler), the specification of security constraints, and the plug-in of application components (ADEPT application configuration tool). All relevant information is stored in the ADEPT repository. The ADEPT WFMS is based on a multi-server architecture [RRD03]. A WF instance may either be controlled by a single server or by multiple servers if favorable. To each server different clients can be connected. ADEPT comprises standard runtime clients for end users as well as for system and process administrators. These clients enable work list display and manipulation (ADEPT work list handler), WF monitoring (ADEPT monitoring client), activity program execution, dynamic WF changes, and system configuration. Dynamic modifications of workflows are modeled using ADEPTflex. ADEPT offers sophisticated modeling concepts and advanced features, like temporal constraint
management, ad-hocWF changes, WF schema evolution, synchronization of inter-workflow dependencies, and scalability [RRD03].

2.5 Overview of Content Management Systems

2.5.1 Definition

“A Content Management System (CMS) is a piece of software that manages or controls the development of a web site in an organization”. “A Content Management System is a tool that enables a web developer in an organization to manage and update the web content with much ease”.

A Content Management System empowers an organization to create and publish its web content in a more structured, effective and convenient manner. A CMS helps an organization in achieving a distributed content management and control by bestowing a web developer with the ability to administer the content without the aid of a webmaster. It can also help an organization in streamlining and automating its content administration by hiding the complexities of HTML from the developer.

2.5.2 Principal Components of CMS

A Content Management System consists of four principal components as illustrated in figure (VI) [RS03].

![Diagram of CMS components](image)

FIGURE (VI): PRINCIPAL ELEMENTS OF A CONTENT MANAGEMENT SYSTEM

2.5.2.1 Front-end CMS

The Front-end of the CMS represents the online/published version of the website, which can be perceived by the visitors of the website. It contains all the webpages, documents, forms, images,
multimedia and other prevalent forms of web content. The front-end acts as a visitor interface to authorize themselves and upload their content (documents) into the server.

### 2.5.2.3 Back-end CMS

The Back-end component provides the strength to the entire CMS from an organization perspective. The back-end of a CMS is an easy-to-use interface which allows non-technical people (such as subject specialist, staff) to edit or enter content. The back-end can be reached through a website, which is equipped with access control for different levels of authority. The web developers are provided with their appropriate access control level so that they will be able to post or remove content from a predetermined section of the website. The web masters are equipped with the access to control/manage the entire website, the ability to change the templates (ie., creation of new web pages with a different design and layout), to include novel features, to change the structure of the site, and to manage the user access to the back-end.

### 2.5.2.4 Templates

Templates are the structures that format and display content following a request from a user for a particular web page. Templates ensure a consistent, professional look and feel for all the content on the site. The templates represent the skeleton of a web page where the content of the web page is embedded into it. These templates are developed by a web master using the web page format designed by professional web developer. A master template is chosen for every web page by the web master, which is used by the web developers in the organization to post their content.

### 2.5.2.5 Database

The Database is the key to Content Management Systems. The database acts as a central repository for retrieving the content, templates, graphics, users and metadata. A true CMS does not store flat, static HTML pages. Instead, the System places the content in a relational database capable of storing a variety of binary and text. The database is also capable of storing metadata including the author info, creation, publishing and expiring dates, content descriptions and indexing information, revision history and a range of other content-related data.

### 2.5.3 Functions of CMS

The basic functions of CMS can be classified into three categories: Creation, Management, and Publication. An advanced CMS can have many functions (archives, built-in search engines, permission control, workflow management etc) in addition to the ones stated above. The CMS functional scope and the content life cycle are depicted in the figure (I).
2.5.3.1 Creation

The Creation Phase involves processing two sub stages: Authoring and developing, before it is forwarded to the management phase. In this Phase, the initial content is provided as an input into the CMS. A Manager is responsible for assigning all the particulars of the task. A professional web designer designs a web page format, which involves specifying the logo, standard navigation options etc. This common web page format is then used to create a master template. All the web developers (Authors) use this master template in generating the content in an environment that is most appropriate for this task. This authored and developed content is sent to the so-called management phase for perusal. This authored content is stored in a repository (XML Files or records in database). The Repository enables the System to exercise version control over the entire content.

2.5.3.2 Management

This Phase consists of editing, assembling, translating, and finally reviewing the content before it is published for viewing by various website visitors. In this phase, the content is sent into an approval, rejection or revisal workflow. An Editor can approve, reject, or revise the content after checking the semantics of the content. If the content needs revision, either he can send it back to the appropriate author for revision or he can edit the content by himself. The content is forwarded to approval stage if it has been accepted by the editor. The approver then forwards the content either to translator or to the publication stage.

2.5.3.3 Publication

This final stage begins upon the approval of material in the management stage. The approved content is embedded into the master template thereby eliminating the time need to format the content to fit the design of the page. The content is then sent to staging upon the final approval by the Quality Assurance. In this stage, the content is seen as if it were delivered in its chosen forms. The content is then published onto web server for viewing by various website visitors.

2.5.4 Benefits of CMS

There is a wide range of business benefits that can be obtained by implementing a CMS. The following are some of the business benefits of Content Management Systems.

2.5.4.1 Content accuracy

Using only standard office applications, business owners can create and update content by simply dragging and dropping files to submit content to the Content Management System, which
automatically applies corporate style and formatting to publish the new content and maintain a consistent look and feel to the site. By achieving content changes in much reduced times and by ensuring content accuracy, a CMS reduces the time to market, delivers improvements to the user experience and saves costs through rationalized resource utilization.

2.5.4.2 Consistent corporate branding
A CMS establishes the design philosophy and ensures it is consistently applied throughout the site to maintain corporate identity and branding. The use of templates allows a ‘design once’ approach, which, by separating content from presentation, reduces the time and effort, required to create and police the brand. The benefit is extended to re-branding activities that, being far easier and less costly to implement, allow the adoption of a more dynamic and vibrant corporate identity.

2.5.4.3 Streamlined centralized management
A CMS that uses template based publishing and a dynamic content repository centralizes control to deliver numerous benefits and efficiencies. Development and deployment investments made for one site can be leveraged for the deployment of further sites that require the same corporate values. Such centralized management delivers on-going efficiencies and cost savings by allowing further changes to be synchronized across multiple sites as a single activity thereby avoiding redundancy.

2.5.4.4 Guarding Content Value
Protecting the value of the content is the pivotal issue to a company’s ability to succeed. It must be accurate, up-to-date and easy to find. Inaccurate content, out-of-date and poorly structured content symbolizes commercial lethargy, hard to navigate and poses a threat to company credibility. A CMS guards the value of content by accelerating and automating publication processes, putting control of content into the hands of the business experts and providing centralized administration of its presentation.

2.5.5 State-of-the-art CMS
2.5.5.1 DeepWeb CMS
DeepWeb CMS is one of the powerful Content Management Systems available in the market today. DeepWeb CMS provides an easy-to-use, user-friendly, platform independent and highly extensible CMS framework to its users. In DeepWeb CMS, using a standard web browser,
business experts can create and update content by either entering text or simply dragging and dropping files to submit content to the Deepweb CMS, which automatically applies corporate style and formatting to publish the new content and maintain a consistent look and feel to the site. By achieving content changes in much reduced times and by ensuring content accuracy, Deepweb CMS reduces the *time to market*, delivers *improvements to the user experience* and *saves costs* through rationalized resource utilization. The use of templates allows a ‘design once’ approach, which, by separating content from presentation, *reduces the time and effort*, required to create and police the brand. Deepweb CMS guards the value of content by *accelerating and automating publication processes*, putting control of content into the hands of the business experts and *providing centralized administration* of its presentation. More detailed information regarding the system architecture and features of DeepWeb CMS can be found at [Var04].

2.6 Service Oriented Architectures

[Nik04] predicts that by 2008, Service–Oriented Architectures (SOAs) will likely be a prevailing software engineering practice, ending the 40-year domination of monolithic software architecture. They model the enterprise as the collection of services that are available across the enterprise. SOA lets the heterogeneous environments and applications exist while leveraging existing applications and infrastructure [Pal01].

SOA is a relatively new term, but the term "service" as it relates to a software service has been around since at least the early 1990s, when it was used in Tuxedo to describe "services" and "service processes" [Her02]. Sun defined SOA more rigorously in the late 1990s to describe Jini [Kei99], a lightweight environment for dynamically discovering and using services on a network. The technology is used mostly in reference to allowing "network plug and play" for devices. It allows devices such as printers to dynamically connect to and download drivers from the network and register their services as being available. SOA is an alternative model to the more traditionally tightly-coupled object-oriented models like Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM). SOA is described to be a set of components which can be invoked, and whose interface descriptions can be published and discovered (W3C). More precisely, one could define the Service-Oriented Architecture (SOA) as "*The aggregation of components satisfying a business driver*" [Pal01].

SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners. SOA is a system for linking resources on demand. In SOA, resources
are made available to other participants in the network as independent services that are accessed in a standardized way. This provides for more flexible loose coupling of resources than in traditional systems architectures.

The most important aspect of SOA is that it separates the service's implementation from its interface. In other words, it separates the "what" from the "how." Service consumers view a service simply as an endpoint that supports a particular request format or contract. Service consumers are not concerned with how the service goes about executing their requests; they expect only that it will. Consumers also expect that their interaction with the service will follow a contract, an agreed-upon interaction between two parties. The way the service executes tasks given to it by service consumers is irrelevant. The only requirement is that the service sends the response back to the consumer in the agreed-upon format.

### 2.6.1 Characteristics & Benefits of SOA

The software components in a SOA are services based on standard protocols and the services in SOA have minimum amount of interdependencies. Communication infrastructure used within an SOA should be designed to be independent of the underlying protocol layer. SOA uses service granularity to provide effective composition, encapsulation and management of services.

SOA provides location independence i.e. services need not be associated with a particular system on a particular network. SOA uses protocol-independent communication framework which renders code reusability. It offers better adaptability and faster response rate to changing business requirements. It allows easier application development, run-time deployment and better service management. SOA is the loosely coupled system architecture, which allows easy integration by composition of applications, processes, or more complex services from other less complex services. It provides authentication and authorization of Service consumers, and all security functionality via Services interfaces rather than tightly-coupled mechanisms. It allows service consumers (ex. Web Services) to find and connect to available Services dynamically. Applications can be exposed more easily to diverse clients: Windows clients, ASP.NET/JSP, etc.

### 2.6.2 Web Services

*Web Services* are an open standards-based way of creating and offering SOAs. Web Services are able to exchange structured documents that contain different amounts of information, as well as information about that information, known as metadata. In other words, Web Services can be coarse-grained (Scope of functionality that a service exposes which is considered to be intelligently structured to meet the business needs). Such coarse granularity is one of the most
important features of SOAs. Web Services are derived from SOAs with additional constraints like, Interfaces must be based on protocols like HTTP, SMTP, FTP and the messages must be in XML, except for binary data. Web Services use SOA because they are similar in their approach regarding inoperable components, a network addressable interface, component re-use and client neutrality.

### 2.6.2.1 Generic architecture of Web Services

A complete Web services architecture consists of three roles (runtime environments)

**A) Service Provider**

The service provider is the system that runs the Web service and provides access to the service through a standard interface. This role is composed of two entities:

1. **Service interface provider**: This entity provides interfaces into the applications used as Web services; for example, a standard organization.

2. **Service implementation provider**: This entity provides the implementation of the applications used as Web services. For example, an application server or any engine that hosts a service.

**B) Service Requester**

This role, also known as the service consumer or client, contains applications that use Web services. The service requester binds to the service offered by a service provider and invokes the service.

**C) Service Registry**

The service registry acts as a service broker for Web services. In other words, the service registry provides a means for service requesters to find Web services offered by service providers.

![FIGURE (VII): GENERIC ARCHITECTURE OF SOA](image-url)
A service provider can publish the services it offers to the service registry. Service requesters query the service registry to find information about available services such as where to locate these services, and how to bind to them. This process can be visualized from the figure [Has03]. An important feature of Web services is that they are defined in terms of the messages that flow on the network between the systems that play the three roles. The messages are defined in terms of a common, standard language (XML), and use standardized protocols such as HTTP running over TCP/IP. This commonality means that Web services can work in an environment where the systems that are playing the different roles can be of different types. Furthermore, a role does not have to be aware of how the other roles are implemented.

2.6.2.2 Technologies of Web Services

A) Simple Object Access Protocol (SOAP)

SOAP is the Web services messaging protocol. It is the core technology that allows Web services to interoperate because of its operating-system and programming-language independence. It is a standard XML messaging light weight protocol used to exchange information between two applications. It has three parts

1. An envelope that defines a framework for describing what is in a message and how to process it.
2. A set of encoding rules for expressing instances of application defined data types. This contains the information about security, caching, routing, and management etc.
3. A convention for representing remote procedure calls and responses.

B) Web Services Description Language (WSDL)

WSDL is the XML language for describing Web services. It is used to create descriptions of Web service interfaces and their implementations so that they can be found and integrated automatically. WSDL provides a grammar for describing services as a set of endpoints that exchange messages. A WSDL document serves as a language and platform-agnostic (XML) description of one or more services. It describes the services, how to access them, and what type of response to expect. A WSDL document can be either privately exchanged or posted to a UDDI registry (either public or private) to allow broader access. In truth, you could expose a pure Java interface, text-based documentation, or any other interface for the service if you choose to do that. WSDL offers a neutral, standardized interface format. WSDL is an XML-based file format for describing types, messages, operations, interfaces (called PortTypes), locations, and protocol bindings. You use WSDL to describe Web services as a set of endpoints operating on messages.
WSDL can describe either document-oriented or procedure-oriented information. In WSDL, *messages* describe the communication between client and service, described by the data types exchanged. The operations consist of incoming and outgoing messages. *PortTypes* consist of a collection of *operations*. The port types are then bound to certain protocols (this is called a *binding*). WSDL supports bindings to SOAP 1.1, HTTP GET/POST, and MIME protocols. However, WSDL is extensible and can be used with other types of network protocols and message formats.

**C) Universal Description Discovery and Integration (UDDI)**

*UDDI* is the Web services database technology. It provides a standardized method for publishing and discovering Web services. The following entities comprise the core of the UDDI data model:

1. **Business entity**: Business entities are the white and yellow pages of the registry. They represent basic business information about a service provider such as business name, contact information, description, identifiers, and categorization.

2. **Business service**: A Business service classifies an individual, logical service. The green pages, part 1, are used to describe, in general, nontechnical terms, a set of services provided by the business.

3. **Binding template**: A binding template, sometimes called access locator, contains technical service access information. The green pages, part 2, contain pointers to technical descriptions and the access point URL. A binding template points to a service implementation description (typically WSDL).

4. **tModel**: A tModel, also called service type, is an abstract description of a particular specification or behavior to which the Web service adheres. It is a digital identification, holding metadata about type specifications as well as categorization information. A tModel points to a service interface description, typically WSDL (for example, through a URL).

**2.6.2.3 Applicability of Web Services**

Web services build on the concept of distributed computing. The categories of the Web Services:

**A) Business information**

Businesses can share information with consumers or other businesses. Examples are stock quotes, news, and weather forecasts.
B) Business process externalization

Businesses can provide transactional, fee-based services for customers. Examples are reservation systems, auction sites, and credit checking.

C) Business integration

Web services can be used to dynamically integrate a company's processes with those of other companies or institutions. For example, a Web service could provide a set of high-level travel features by coordinating lower-level Web services for air travel, hotels, and car rental.

2.6.2.4 Evaluation of Web Services with Various SOA Technologies

Some of the major advantages of Web Services against CORBA, DCOM, and RMI are illustrated below.

A) Interoperability

There is a tight coupling between client and the server in CORBA, RMI since the both ends must share the same interface whereas in Web Services, Everything is loosely coupled since it uses the universally accepted language as the medium of transferring the messages between client and the server.

B) Serializability

This factor influences various issues such as persistence, extensibility and ease of interoperation with other frameworks. Web Services allows for user-defined serialization mechanisms. Web Services being XML based, is too verbose as compared to CORBA.

C) Ease of Deployment and Assembly

Deploying any service and discovering the same is very cumbersome in CORBA, RMI where as in Web Services there are some technologies used like UDDI, WSDL, SOAP which makes easy service deployment and discovery.

D) Secure architectures with firewalls

Web Services uses HTTP to transfer or exchange information between two systems and most firewalls have been configured to accept HTTP traffic hence messages can tunnel through firewalls. CORBA does not use a specific port for communication.
E) **Degree of familiarity**

This relates to the degree of familiarity with the technology, learning curve, and number of developers required/available. In this context, Web services have an upper hand, since they are based on well-known and heavily used protocols/formats, namely HTTP and XML.
CHAPTER 3 DESIGN

3.1 Deepweb CMS Overview
A Content Management System (CMS) is a piece of software that manages or controls the development of many web sites in organizations. A CMS empowers organizations to create and publish its web content in a more structured, effective and convenient manner. A CMS helps an organization in achieving a distributed content management and control by bestowing a web developer with the ability to administer the content without the aid of a webmaster. It can also help in streamlining and automating its content administration by hiding the complexities of HTML from the average user. The user interface can be operated intuitively without special knowledge to administer the internet and intranet contents. What really counts is the content. Even administrative tasks can be fulfilled in a matter of minutes. One of the powerful commercial products which support all of the above mentioned functionalities is Deepweb CMS [Deepweb]. In a standard web browser, business experts can create and update content by either entering text or simply dragging and dropping files to submit content to the Deepweb CMS, which automatically applies corporate style and formatting to publish the new content and maintain a consistent look and feel to the site. By achieving content changes in much reduced times and by ensuring content accuracy, Deepweb CMS reduces the time to market, delivers improvements to the user experience and saves costs through rationalized resource utilization. The use of templates allows a ‘design once’ approach, which, by separating content from presentation, reduces the time and effort, required to create and police the brand. Deepweb CMS guards the value of content by accelerating and automating publication processes, putting control of content into the hands of the business experts and providing centralized administration of its presentation.

3.1.1 Features of Deepweb CMS

3.1.1.1 WYSIWYG-creation of content
The contents are entered using a specialized graphical user interfaces within the internet browser. These interfaces blend in the website itself and create additional editorial functions. Text format can be changed using the control bars and shortcuts, and these changes appear in real time i.e. it gets a WYSIWYG-effect. The formatting is done with design-templates, which the editor can choose. The editor can decide when and how long a text goes online by specifying two dates. According to these dates the system will automatically put the contents online or offline. The
editor can manually switch the contents online or offline, so that the contents, which are not, yet perfect cannot be seen on the actual website.

3.1.1.2 Moderation/Presentation

As far as the maintenance is concerned, there are different user rights, namely ‘read’, ‘write’ and ‘switch online’. Thus, it is possible to create moderate contexts: There are editors who can write and maintain new contents and the moderator proofreads the contents and puts them online.

3.1.1.3 Using News tickers and Newsletters

In addition to their content list the contexts are provided with a news ticker. Via a special registration mechanism users can subscribe to a newsletter which will be sent either via e-mail or via SMS. The content of the newsletter be maintained via the CMS-system and can be sent at the push of a button. Deepweb CMS automatically creates a survey of the actual contents. The subscriber will immediately be directed to the contents of the website by clicking on the area he is interested in.

3.1.1.4 Easy administration of contexts by an administrator

The structure of the contexts can be easily maintained via the browser. A navigation editor supports the creation and changing of sub-contexts. Just like the contents, these can be switched online and offline individually.

3.1.1.5 User administration based on contexts and rights

The administration of the users and their rights within the context structure is also maintained via the browser. The rights that are defined for a special context (‘read’, ‘write’, ‘switch online’, ‘administration’) will be transferred to the respective sub-contexts.

3.1.1.6 Areas with limited access

Not all the information is meant to be available for the public. Important and confidential contents can be put in a special area, which is placed under password protection. Only registered users will have the access, and the contents will still be available from anywhere around the world.

3.1.1.7 Broken links

Old hyperlinks will automatically be identified within the contents. The author/owner of the website will automatically be informed about such ‘broken links’. Thus one could have the possibility to keep the website up to date and react immediately to changing structures in the
internet. When ever the contents are missing for a navigation position, then the system notifies via e-mail.

3.1.1.8 Printed version

This system allows offering the internet customers a specialized printed version of the contents. Thus the visitor can easily print a report. The printed version will be automatically generated on the existing webpage.

3.1.1.9 Media database to set up download elements and pictures

The media database transfers graphics and download elements via browser based mask to the server and integrates them into the contents. Thus detours like FTP will become unnecessary. The user simply has to choose his data file via the search engine.

3.1.1.10 Search function and archive

This system offers search functions for the texts or adapted search engines to look for the contents on the website. This includes the administration of older reports in an automatic archive.

3.1.1.11 Voting-module and Multi-layout capability

Conduct surveys on the website, asking the visitors to provide possible answers (multiple choice) for certain questions and analyze the results statistically. There are layout variations in the system to create a diversified website.

3.1.1.12 Portal-assistant

This system has a portal-assistant which supports in creating new portals and new internet projects. Here the portal-assistant accesses a prepared basis version of the website which has been created individually by agreement. The assistant then creates a basic version of the new domain. New portals can be amplified using the administrative tools to an extensive new internet presentation.

3.1.1.13 Deepweb CMS publication server

The publication server is especially adapted to serve the public section of the internet presentation. The CMS contents are provided with an additional protection so that it will be virtually impossible for any invader to change the contents. By using an automatic mechanism, the publication server will take the contents out of the content management system within the
firewall at a particular time or at the push of a button. Then the publication server will transfer these write-protected contents to the public internet.

3.1.1.14 Extensibility
Deepweb CMS offers an open architecture: The core of the system consists of Java components which can be amplified. Java represents a world wide standard: Via JSP (Java Server Pages) one can access one’s own or external modules with in the JSP-environment. Thus there is a possibility to integrate existing modules in the CMS based internet presentation. HTML modules can also be integrated into the CMS internet presentation in a matter of seconds.

3.1.1.15 Platform
Deepweb CMS is built upon standard internet components. Thus the availability and the constant development of the systems are guaranteed.

3.1.2 Structural overview of Deepweb CMS
Deepweb CMS is responsible for the creation, management, and publishing of chunks of information known as content elements. Schematically, one can envision the process as shown in Figure (VIII). The contents which have to be used throughout the system are specified in the Content Entry block. The components of Deepweb CMS are

3.1.2.1 Templates
The main purpose of the templates is to provide a consistent appearance (corporate design) of your website and such a consistent appearance is increasingly important to all businesses. In addition, consistent websites attest to professionalism and allow the visitors to read comfortably because of a calm and consistent typeface and layout. The templates instantiate: the formatting syntax and surrounding standard text and media elements of the target publication platform, the page structure and syntax of the target publication platform, Content components and metadata on the target pages, and the standard text and binary files from the repository onto the target pages.

3.1.2.2 Creation System
The creation system is responsible for all content elements before they are ready for publication. It turns raw information (unstructured information) into a well-organized set of content components (structured information which includes graphics, media, text, pictures etc to be published on web). The main phases of creation system are
A) **Authoring:** In this phase, the content is created from scratch.

B) **Acquisition:** In this phase, the content is gathered from some existing source.

C) **Developing:** Creating a master template to put the created content into it

Firstly, Administrator assigns all the particulars of the task to the actors (employees) and then the web developers (Authors) create and structure the content (elements of content-entry block); the professional web designer designs a web page format, which involves specifying the logo, standard navigation options etc. This common web page format is then used to create a master template. All the web developers (Authors) use this master template in generating the content in an environment that is most appropriate for this task. This authored and developed content is sent to the so-called management system for perusal.

### 3.1.2.3 Management System

The management system is responsible for the approval of content to publish on the web site and long-term storage of content elements. It contains the repository, and the administration facilities. The authored content obtained from the creation system is stored in a repository (records in database). The repository enables the system to exercise version control over the entire content. In this system, editing and reviewing phases are carried out before the content is published for viewing by various website visitors. An Editor can approve, reject, or revise the content after checking the semantics of the content. If the content needs revision, either he can send it back to the appropriate author for revision or he can edit the content by himself. The content is forwarded to approval stage if the editor has accepted it. The approver then forwards the content to the publication system that begins upon the approval of material by the Quality Assurance team in the management system. This component is capable of telling the details about the existing content elements, the bottlenecks that are to come up, usage of content elements in publications, unused content, who has access to what content, and who has contributed the most.

### 3.1.2.4 Publishing System

The publishing system is responsible for pulling content components out of the repository and automatically creating publications out of them and formatting them using the templates. When content is added to the repository it cannot be determined where and when it will be used in a publication. Therefore, the publication system is able to read and resolve content links when the publication is being produced. For example, if component A has a link to component B in the
FIGURE (VIII): DEEPWEB CMS FUNCTIONAL SCOPE AND CONTENT LIFE CYCLE

the appropriate file and directory set for the target publication. Additionally, the system has some mechanisms to arrange and deploy the built publication via content router (which acts as a dummy component in the architecture) onto web server for viewing by various website visitors.
3.1.2.5 Content Router

This component routes the content from the publication system to application web server for the deployment of the content on the web.

3.1.2.6 Front-end CMS

The front-end represents the online/published version of the website, which can be perceived by the visitors of the website. It contains all the webpage’s documents, forms, images, multimedia and other prevalent forms of web content. It also acts as a user interface by authorizing users and allowing them to upload new content to the server.

3.1.2.7 Database

The Database is the key to Deepweb CMS. It acts as a central repository for retrieving the content, templates, graphics, users and metadata. Deepweb CMS does not store flat, static HTML pages. Instead, it places the content in a relational database capable of storing a variety of binary and text. The database is also capable of storing metadata including the author info, creation, publishing and expiring dates, content descriptions and indexing information, revision history and a range of other content-related data.

3.1.3 User roles of Deepweb CMS

The content management roles provide the organizational framework required to develop and support a website as an online service. Without these, effective management of web content will be unsuccessful and reflect poorly on the usability of the web site. Some of the typical roles in CMS are editors, publishers, site administrators etc. Several users can be assigned to a single role or a single user can be assigned to multiple roles depending on the workload of the other users or on the amount of tasks such as editing, designing graphics, writing HTML code, etc., to be accomplished. For example, the person who approves content may be the same person who publishes it. Each role has a particular set of privileges. Deepweb CMS has the following roles involved in the process of creation, management and publication:

3.1.3.1 Roles in creation process

A) **Content contributor:** supplies the content.

B) **Content Author:** designs graphics and content containers/templates, populates or categorizes content. Authors could be graphic designers, content managers, or taxonomy specialists for categorization.
3.1.3.2 Roles in Management process

**Content Reviewer/Approver:** reviews and approves content before it goes live.

3.1.3.3 Roles in Publication process

**Content Publisher:** publishes the content on the web and notifies the author.

3.1.3.4 Other Roles

A) **Content System Administrator:** set ups and maintains the technical environment

B) **Content Consumer:** searches or browses the content

C) **Community User:** registers for the use of guest book, forums.

3.2 Mentor-WS Overview

The Mentor-lite prototype has evolved from the Mentor WfMS ("Middleware for Enterprise-Wide Workflow Management") [MWG+98, MWW+98]. The Mentor-lite prototype has been developed within the research project "Architecture, Configuration, and Administration of Large Workflow Management Systems". Mentor-lite was aimed at developing a simpler architecture compared to its predecessor (Mentor). The main goal of Mentor-lite has been to build a lightweight, extensible and tailorable workflow management system (WFMS) with small footprint and easy-to-use administration capabilities [WGR+00]. Mentor-WS is the extension of Mentor-lite which aims to provide workflow interoperability, loose coupling between systems, and open source standards. Mentor-WS stands for Mentor Web Services. It was developed in a parallel master thesis and, is an integral component of this project [Sai04]. Mentor-WS uses Web services as the communication medium due to its excellent features such as interoperability, loose coupling, ease of deployment and assembly, familiarity, etc., Mentor-WS is used to improve the control of work in any business processes, increase productivity of the professional staff, improve efficiency through automation of business process, provide better flexibility through software control over the processes and improve focus on business process.
3.2.1 Features of Mentor-WS

3.2.1.1 Light-Weight and Flexible WfMS

Retaining a light-weight nature in WfMSs is an important feature in achieving a flexible WfMS. Mentor-WS achieves the light-weight nature by modeling the system components like history management and worklist management as extensions on top of the workflow kernel [as depicted in Figure (IX)] and thereby implementing these extensions as workflows themselves. This approach provides a flexible and open-modular architecture, where components can be added or replaced by alternative implementations.

3.2.1.2 Platform Independence

By using the Java and Internet technology (XML, DHTML), Mentor-WS provides a high degree of platform independence, since it is feasible to access the system from a variety of hardware and software platforms. Mentor-WS is successfully implemented in both linux and windows platforms.

3.2.1.3 Flexible exception handling and dynamic modifications

The Interpretation of workflow specifications at runtime by the Mentor-WS’s workflow interpreter enables Mentor-WS to achieve flexible exception handling and dynamic modifications during runtime.

3.2.1.4 Conformance with WfMC Standards

The Mentor-WS prototype has been built based on the reference architecture of WfMC. This feature serves as a basis for evaluating Mentor-WS’s architecture with various architectures of real systems.

3.2.2 Structural overview of Mentor-WS

Figure (IX) shows an architectural overview of Mentor-WS. In the Design Phase, the designer defines the workflow specifications using the WfMC process definition language and converts it into an intermediate state and activity chart format by using the state and activity chart converter. In the Runtime Phase, the service requestor invokes the workflow server using the Web Service interface of the workflow engine, which then interprets the control flow data and forwards to workflow engine for execution. The Communication Manager (CM) receives the workflow instance, puts it onto a workflow queue and invokes the workflow interpreter as well as the log
manager with the first workflow instance in the workflow queue. The workflow interpreter then interprets the workflow specifications defined at the design time and carries out the execution invoking the corresponding host or external applications. These applications in turn use the data stored in business data repository, obtained from the requestor’s web browser to set control flow variables. The communication between different workflow engines is accomplished by invoking their corresponding web service interfaces, where invoking workflow engine acts a client and the invoked workflow engine serves a server. Based on the control flow data (variables) set by the application wrappers, the workflow execution continues through various states and activities until the termination state is reached. Mentor-WS makes use of Web Services as middleware for interacting with both external applications and other workflow engines. Since all the messages between the external applications and workflow server are control flow messages and handled by Web Services itself which enables the communication between the external applications and the workflow server.

3.2.3 Components of Mentor-WS

3.2.3.1 Workflow Engine

The Mentor-WS workflow engine provides a runtime execution environment and manages the overall processing and the execution of workflow process instances. The Mentor-WS workflow engine comprises of three basic components.

A) Workflow Interpreter

Workflow Interpreter is the basic building block of Mentor-WS for workflow specifications. The workflow interpreter interprets at runtime, the workflow specifications, which are modeled at design time using state and activity chart process definition language. The workflow interpreter performs a stepwise execution of the workflow specification at the runtime according to its formal semantics [DG97]. The interpreter checks the values of conditions and performs the steps in the workflow following the specification by setting conditions or starting activities.

B) Communication Manager

The Communication Manager is responsible for sending and receiving synchronization messages between the engines and invoking the workflow interpreter [WGR+00]. These messages contain information about locally raised events, updates of state chart variables and state information of the local engine. When a synchronization message is received, the corresponding updates at the receiving site are performed.
C) Log Manager

The log manager provides logging facilities and recovery mechanisms [WGR+00]. It records all the detailed information about the changes to state, event, and data-element information of all active workflow partitions that the engine server is responsible in a DBMS. It can be used to detect a crashed state in the event of system crash. The stored logging information (conditions) enables log manager to recover from a crashed state by resuming the execution from crashed state without processing its previous states.

3.2.3.2 Workflow Specification Environment

The workflows are specified in terms of state and activity charts specification formalism [as discussed in Section 2.3.2]. Mentor-WS uses a commercially available run-time system for state and activity charts as the workflow engine, which serves as a common denominator of different modeling languages that can be mapped onto such an engine. Thus, workflow applications can be specified in whatever modeling language is considered most appropriate, and this specification will be transformed into a "canonical" representation that is suitable for our engine.

3.2.3.3 Worklist Manager

Each activity that becomes ready for execution during a workflow execution constitutes a work item that is communicated to a worklist manager. The worklist manager assigns a work item dynamically to one of the actors based on their expertise, experience, availability and workload. The worklist manager retrieves these role resolution parameters from worklist database. Furthermore, the worklist manager also uses additional time-related information (deadlines) about previously processed work items that is extracted from the history database. Organizational structure of the enterprise and workflow-instance-specific restrictions is the additional parameters, which can influence the dynamic role resolution of roles into actors.

3.2.3.4 Repositories

Mentor-WS uses three repositories for storage and retrieval of information. Each of these repositories is associated with a component of the Mentor-WS workflow engine. Worklist and Workflow repositories can be shared by Mentor-WS workflow engines at different sites.

A) Worklist Repository

Mentor-WS uses this repository to store all the roles specified in workflow specification, all the actors who are responsible and capable of performing the tasks of specified roles, and all the detailed information about interactive activities (i.e., priority, start time, dead line, status,
FIGURE (IX): MENTOR-WS ARCHITECTURE
description of the activity) that are assigned to corresponding actors. Worklist manager issues queries to retrieve the information about Worklist actors dynamically from this repository and assigns work items to the actors based on the workload, experience, presence.

B) Workflow Repository

This repository stores information about the workflow specifications in the form state, activity, and module charts. Workflow interpreter uses this repository to interpret the workflow specifications at runtime.

C) Workflow Log

Mentor-WS provides logging information and recovery details based on the information stored in this repository. This repository stores detailed information about all the running workflow instances (i.e., states, activities, events, conditions, user data, and timeouts for every workflow instance). The log manager can use this repository to retrieve information about all the activities that are already performed, current state, start and end time of workflow instances, condition values etc.

3.3 Motivation

Managing objects (i.e., content, such as a web page, an image, etc.) in a website is typically organized as a workflow within the company. An example is a company’s press release: an employee writes a press release and submits it to an editor for review before it is published. This review process is a workflow that is used by the administrators to ensure that site content is correct. It is called workflow as the work is distributed among the actors and the accomplished work is flown among the actors. Advantage of having workflow system in CMS is to automate the approval and review processes for content, thereby increasing the manageability of the site, and improving quality control. So even though it seems very natural to integrate CMS with WfMS, most existing CMS do not support this functionality. The functions of workflow system in CMS are

- Involving in every step of the process, from authoring through final deployment of each publication should be able to be modeled and tracked within the same system.
- Representing all of the significant parts of the process including:
  - Staff members who are responsible to do certain tasks such as content collection, content creation, content management, content publication, notification etc.
  - Time and data flow with a variety of transitions and charting representations.
• Representing any number of small cycles within larger cycles with some sort of drill down and up to the appropriate level of detail.

• Offering a visual interface that shows cycles and process members graphically.

• Making metadata in the repository available. The workflow system should not have to store its own staff members, content types, outlines, and other metadata. It should be able to read the data that is stored in the repository and make it available when appropriate in its dialogues and selection screens. For example, an editor might select a content type for an article in a workflow screen in order to forward it to the right next reviewer. The list of selections should come from the repository, not from the workflow system's own internal data store. As an alternative, the workflow's data store (which would need to be some sort of open database) could be considered part of the repository that is responsible for storing certain metadata.

Thus the workflow system integration is one of the key goals of the CMS in order to build large-scale, robust and efficient CMS. The proposed system should be able to provide a framework, which enables seamless integration of Deepweb CMS with Mentor-WS (A light weight workflow management system) using service-oriented architecture (Web Services).

3.4 Conceptual Design of the Proposed System

This section introduces the conceptual design of a novel Deepweb CMS; the section is organized as follows: First, design goals for the proposed system are discussed. The section concludes by presenting the modeling alternatives and design architecture of the proposed system.

3.4.1 Design Goals

3.4.1.1 A flexible framework for integrating WfMS with Deepweb CMS

The Workflow management system being integrated in Deepweb CMS is a light weight system with lot of features such as interoperability, platform independence, open source support, easy-to-use process definition tool, etc.

3.4.1.2 WfMC standard process definition language

Process definition languages play a significant role in the functioning of WfMS. Therefore, it is indispensable to use an established framework for modeling and analyzing workflow processes. As part of the standardization program, WfMC has specified the interface one of its reference model for supporting process definition import and export. This interface includes a common meta-model for describing the process definition and an XML schema for the interchange of
process definitions. The meta-model identifies commonly used entities within a process definition. A variety of attributes describes the characteristics of this limited set of entities. Based on this model, vendor specific tools can transfer models via a common exchange format. The proposed system should be able implement this process definition language. Further, the proposed system should be capable of providing functionality, which can enable the workflow engine to interpret the process definitions modeled using this new process definition language.

3.4.1.3 Efficient Content management

To make the entire process of content creation, storage, and publication run effectively and efficiently according to well-defined timelines and actions after the integration of the workflow system.

3.4.2 Modeling alternatives

3.4.2.1 A flexible framework for integrating WfMS with Deepweb CMS

There are many WfMS’s such as Mentor-lite, Mentor-WS, etc. In order to achieve the flexible framework, the WfMS has to provide workflow interoperability and loose coupling architecture. One possible solution is to use a middleware in WfMS which supports the above mentioned features. Since Web Services is loosely coupled and supports interoperability, since Mentor-WS is based on web services, it can provide a flexible framework needed for the integration of Deepweb CMS. The alternatives to web services could be CORBA, RMI, RPC, Java Beans, HTTP, etc, which also provides the framework but not the flexible framework. There is a tight coupling between client and the server in CORBA, RMI, Java Beans since the both ends must share the same interface whereas in Web Services, Everything is loosely coupled since it uses the universally accepted language as the medium of transferring the messages between client and the server. Deploying any service and discovering the same is very cumbersome in CORBA, RMI where as in Web Services there are some technologies used like UDDI, WSDL, SOAP which makes easy service deployment and discovery. Due to these factors web services is chosen to provide the flexible framework.

3.4.2.2 Process definition Interface

The Mentor-WS’s workflow engine is capable of interpreting process definitions based on state and activity chart representations. The workflow engine of the Mentor-WS will be able to interpret the state/activity chart representations for further execution of the defined process. Mentor-lite uses the commercial tool namely Statemate in order to represent the process
definitions based on state and activity charts. But our design goals are not to rely on commercial products so the alternative way to solve this issue could be developing a common interface that is capable of interpreting the process definitions modeled using the process definition language of WfMC and translating it into the existing state and activity chart representations, which the workflow interpreter is capable of interpreting. A converter eliminates the usage of commercial products for Mentor-WS. Thus a converter needs to be developed for translating from one representation into state and activity representation.

3.4.3 Proposed System Architecture

Based on the design goals and modeling alternatives discussed above, a novel system as shown in the Figure (X) has been developed. In the proposed system, Deepweb CMS uses Mentor-WS as a plug-in tool to assure that the entire process of creation, management and publication is done effectively and efficiently. The proposed system comprises of two different systems namely, Deepweb CMS and Mentor-WS. The middleware used to connect these two systems is the well known technology of web services. Web services\(^1\) provide easy integration, loose coupling between these two systems. Deepweb CMS and Mentor-WS use common database to store and retrieve business data. Prior to the execution of Deepweb CMS, the roles involved in CMS system are assigned to actors of Mentor-WS and stored in the common database. At design time, a Workflow designer specifies the workflow specifications using a standard process definition language and the converter translates into state/activity chart representations. All the activities defined are implemented as external wrappers which get business information and CMS information from the common database. The activities could be either automatic or interactive. If the activities are automatic then the external wrappers performs the necessary functionality and returns to the engine. If the activities are interactive then the activities to be performed are assigned to the corresponding actors based on the roles defined during the design time by the worklist manager. The worklist manager provides all the details about the activities such as activity name, priority, start time, deadline, etc, to the actors. The engine waits until the actors perform their tasks and submit the results back to the engine. At regular intervals of time, the engine checks a `putfile` in which all the results of the interactive activities are put. Once the user performs his/her task then he/she puts some condition along with the activity name in a `putfile` and thus the engine comes back into the running state. The engine runs until all the states in the state chart are visited. A Log manager keeps track of the entire execution flow. When ever the crash occurs during the execution, Log manager recovers the previous state of the last run and

\(^1\) Refer 2.6.2
FIGURE (X): NOVEL DEEPWEB CMS FUNCTIONAL SCOPE AND CONTENT LIFE CYCLE
starts executing from the current state. Crash could be due to power failure, server down (due to heavy traffic), etc. Mentor-WS stores the completed state names and the condition names and their values in an xml file and when ever the crash occurs then Mentor-WS is terminated abruptly. During the next run of Mentor-WS for the same workflow instance, Mentor-WS check in the xml file for that instance and retrieve all the states and conditions from the xml file and puts all the conditions into the engine. The system blocks the invocation of external wrappers of the completed states and thus the engine starts from the state where crash occurred in the last run. Whenever a designer wants to update the layout of a website, he clicks a button on Deepweb CMS webpage, which then starts Mentor-WS using its web services interface. For each activity, an external wrapper is invoked and performs a certain action. Possible activities could be reviewing, editing, publishing the layout and notifying the user about the publication or rejection. These activities are managed by Mentor-WS, thus the entire process of content management and content publication is automated. Mentor-WS allows setting and administering the chain of events around creating, managing and publishing.
CHAPTER 4 IMPLEMENTATION

This chapter discusses the implementation of integrating Mentor-WS into Deepweb CMS based on the conceptual design presented in the previous chapter. Section 4.1 presents a system overview and section 4.2 illustrates the main system components while section 4.3 presents the working principle of the system. Section 4.4 elucidates the entire system with the help of a sample application. Finally the implementation issues are discussed in section 4.5.

4.1 System Overview

The conceptual design presented in the previous chapter lead to the development of a system design structured along the technical details shown in Figure (XI).

4.1.1 Main System Components Overview

An overview of the main components used in Figure (XI) is described in this section.

4.1.1.1 Deepweb CMS

Deepweb CMS\(^2\) is a powerful content management systems developed by Deebweb GmBH. It is built upon standard internet components. The technical details of Deepweb CMS are

A) Platform

Deepweb CMS is implemented in 100% java. Since java is a platform independent language, it runs on all the operating systems supporting java.

B) Web Server

The reference web server for Deepweb CMS is Apache web server. Apache web server is Open Source and free of charge. It uses Apache from version 1.3 [Apache].

C) Servlet Runtime Engine

The Java Servlet API (JSDK) 2.3 and a servlet engine are needed by the web server to start up Java programs on http requests. The servlet standard has been developed by SUN, [Servlet]. Deepweb CMS uses Tomcat as a Java Servlets and Java Server Pages reference implementation from the Apache Foundation [Tomcat]. It uses Apache Tomcat from version 3.

\(^2\) Refer 3.1
FIGURE (XI): SYSTEM ARCHITECTURE
D) Database

Deepweb CMS architecture allows using any SQL capable databases that offers JDBC connector. Presently, Deepweb CMS uses open source database namely, MySQL [MySQL].

4.1.1.2 Mentor-WS

Mentor-WS\(^3\) stands for Mentor Web Services. It is a light weight workflow management system and was developed in a parallel master thesis and thus is an integral component of this project [Sai04]. The technical details of Mentor-WS are

A) Platform

Mentor-WS is implemented in ‘C’ and java. We have compiled ‘C’ and created dll for the project using Microsoft VC++. So presently Mentor-WS can be operated only in windows. But Mentor-WS is expected to work in all platforms by compiling and creating dynamic link libraries in their respective platforms.

B) Middleware

Mentor-WS uses web services\(^4\) as the middleware to connect to the external systems. It uses Apache Axis framework to deploy its main service in Tomcat application web server. Apache Axis is the development toolkit or environment that helps one to deploy Java Web Services. Axis was created by the collaboration of IBM who developed SOAP4J (Java SOAP tools) and Apache working on SOAP definitions and implementations. The goal of Axis is to make deploying a web service very simple and hassle free. Axis has an "easy to deploy" feature that lets one to drop a source file into the AXIS Web application and have it become a Web service within seconds. With the help of Axis, it is very simple to create a working, accessible service compared to other technologies. Alternative technologies for hosting and accessing web services are Microsoft .NET framework, SUN JWSDP, IBM Websphere SDK. The reason for choosing Axis framework is due to its simplicity in deploying and accessing web services.

C) Database

Mentor-WS uses MySQL database. MySQL database is chosen in order to provide the open source support. The database schema that Mentor-WS uses is specified in Appendix B.

---

\(^3\) Refer 3.2
\(^4\) Refer 2.6.2
4.1.1.3 Java Workflow Editor (JaWE)

JaWE (Java Workflow Editor) is a graphical application for Process Definition Modelling developed by Enhydra. It is compatible with WfMC specification - XPDL (XML Process Definition Language) version 1.0 - 102502. It is an OpenSource project developed entirely in java using java swings. It is the first graphical Java workflow process editor fully according to WfMC specifications supporting XPDL as its native file format. Its user-friendly graphical user interface lets users to create workflow process definitions quickly and intuitively. It validates the process definitions and stores them in either a text file format or in a XPDL file format for further use. The valid XPDLs can be imported/referenced into new ones, thus shortening the time and effort needed to define new workflow processes [NSV+04].

4.1.2 Working Principle Overview

4.1.2.1 Design Phase

As shown in Figure (XI), a workflow designer reviews the business process to be modeled and defines it using JaWE. These specifications are then converted into XPDL\(^5\) (represented in the form of XML) by JaWE. The designer then uses XPDL-StateChart converter to convert the XPDL-style workflow specifications to an intermediate StateChart representation format. We have developed XPDL-StateChart\(^6\) converter as a part of this thesis and the conversion is shown in Figures (XIV/XV). The workflow designer can validate the workflow specifications during conversion and in case of validation errors, the designer has to edit the workflow specifications using JaWE and re-submit the edited version to the converter. Finally, the workflow designer uses the worklist manager to assign roles (defined and assigned to activities using JaWE) to the workflow participants. The information about roles and workflow participants are stored in a relational database schema\(^7\) using MySQL database. Process modelling using JaWE is explained in detail in section 4.3.1. Let us assume that Mentor-WS’s main web service is already deployed in Apache Tomcat application web server using Apache Axis frame work. So the details about deployment are not shown in the figure. Now this main service of Mentor-WS will be invoked by Deepweb CMS. Deployment of this service is done is only once and is explained in more detail in section 4.2.1.

---

\(^5\) Refer to Section 2.3.3  
\(^6\) Refer to Appendix A.1  
\(^7\) Refer to Appendix A.3
4.1.2.2 Execution Phase

Deepweb CMS initiates the execution of a workflow by submitting the business (data) as well as the control flow data. An end user can intuitively enter this data into JSP/DHTML pages displayed by his browser and submit it to Deepweb CMS. The JSP page separates the business data from the control flow data. It stores the business data into the MySQL database using JDBC and sends the control flow data using (Apache-AXIS) web service to the workflow engine. The web service interface of Mentor-WS is a java wrapper on Dsiserver. Dsiserver is a thin layer which receives the control flow data from the web service interface and creates a unique workflow instance identified by the triplet (Workflow ID, Workflow Type & Process ID) and attaches it to a thread. This thread object is then registered in a local registry built by JNDI and forwards it to the workflow engine (developed in ‘C’) for execution using Java native interface (JNI). JNI provides an interface between Java and native languages such as ‘C’, ‘C++’, ‘Perl’, etc. The workflow engine receives this unique workflow instance, sends it to the communication manager, which then invokes the workflow interpreter and the log manager simultaneously. The workflow interpreter interprets the statechart workflow specifications generated at the design time and performs a systematic execution of activities, while the log manager records all the logging information into a file. Depending on the type of activities (i.e., interactive or automatic), the workflow engine invokes the corresponding application wrappers, for executing the appropriate workflow client applications. In case of interactive activities, the wrappers invoke the worklist manager that assigns the workitems (tasks) to appropriate workflow participants in accordance with the roles that have been assigned to the activities during design time. Then the workitems are accomplished by the workflow participants. Based on the control flow data set by the workflow participant, the workflow execution continues through the next subsequent states and activities until the termination state is reached. Mentor-WS processes all the activities of CMS (editing, reviewing, rejecting, and publishing) through the external wrappers. These external wrappers store all its data in the common MySQL database again. Subsequently, the Deepweb CMS uses the processed data during its execution phase.
4.2 Working Principle In Detail

4.2.1 Design phase

The first step in the design phase is to model a business process using JaWE.

4.2.1.1 Workflow Modeling

Modeling a business process can be viewed as a six-step process in JaWE. They are

A) Defining a Package

In this step, the business processes are grouped and assigned to a package identified by a unique package id and a package name. The workflow designer has to specify a unique package id, which can only be set once and an optional package name in the package properties window. The designer cannot proceed further without assigning a unique package id to the package. The package id should be unique in case if you are using Mentor-WS workflow engine for executing workflows.

B) Defining Processes

In this step, the processes are defined inside a package. The designer can create a new process by clicking on the process button which exists in the package toolbar. The designer has to specify a unique process id and name for each of the processes that are to be modeled in the process properties window. The process id should be a valid number.

C) Defining Workflow Entities

In this step, the designer can define various workflow entities such as workflow relevant data, workflow participants, workflow applications, type declarations, and external dependencies at either the package level or the process level. Defining the workflow entities at the package level makes these entities available to all the processes in the package, while process level restricts it access to only that process. Figure (XII) shows the definition of workflow participants at the package level and the definition of workflow relevant data (workflow variables) at the process level. In the Figure (XII), workflow relevant data has a unique id, name/description associated with the id, and the data type. For example, LOGIN_OK is the id which is of type Boolean and the name/description is LOGIN SHOULD BE OK. The Workflow participants have a unique id, name, and the participant type. For example, ADMINISTRATION is the participant name, id is 2, and the participant type is ROLE.
D) Modeling Workflow Processes

In this step, the workflow process is modeled by creating different kinds of activities (such as generic, route, block & sub-workflow activities). The designer can define transitions and conditions between the various activities as shown in figure (XIII). Before creating the activities, the designer has to assign each activity to one of the workflow roles or participants defined at either the package level or process level. The designer should specify unique activity ids and names while creating the activities. The top user interface shown in Figure (XIII) is the process layout in which all the states/activities are created. Each state/activity has the properties section where the transitions, conditions can be set. For example, `LOGIN_ACTIVITY` is a state/activity,
FIGURE (XIII): MODELING WORKFLOW PROCESSES
and there is a transparent block (details the properties of the \textit{LOGIN\_ACTIVITY} state) between the \textit{LOGIN\_ACTIVITY} and \textit{ROUTE1} states. The middle user interface shown in Figure (XIII) shows all the activities that are created in the process. And the bottom user interface shows all the transitions that are stated for every state/activity.

\textbf{E) Validating the Process Definition}

In this step, the modeled business processes are validated to make sure that there are no XPDL schema errors, broken connections between the activities, inconsistencies in graphs, and logic errors. The designer can perform this validation at either the package level or the process level by using the validation tab.

\textbf{F) Exporting the Process Definition}

After validating the processes at the package level, the designer can export the process definitions into an XPDL file by using the save option from the file menu of the main editor window.

A detailed description about the syntax, usage and example workflow specifications using JaWE can be found at [NSV+04]. As discussed in the modeling alternatives of the previous chapter, a converter is developed to convert from XPDL format into Statemate format. Sample XPDL file and its corresponding Statemate format are shown in Figures (XIV) and (XV).
FIGURE XV: SAMPLE STATEMATE FORMAT

The details of the Figure (XV) are explained in detail in section 4.3.1. Let’s now map the elements of XPDL file into state mate format elements. The mappings are specified in a table shown in Figure (XVI) taken from the Figures (XIV) and (XV). The first column in the table indicates all the elements for which the mapping is done. The second column indicates the XPDL format of the corresponding element and the third column shows the Statemate format of the respective element. Some more elements like participants, package name, etc that were not taken into consideration for mapping are also mapped in the similar way.
<table>
<thead>
<tr>
<th>Element Name</th>
<th>XPDL Elements</th>
<th>Statemate Elements</th>
</tr>
</thead>
</table>
| Process Name | `<Workflow><Processes>`<br>`<Process AccessLevel="PUBLIC" id="1" name="Registration">`
`<ProcessHeader DurationUnit="D">`
`<Created 2004-05-11 17:32:55</Created>`
`</ProcessHeader>` | `chart`<br>`name : REGISTRATION_SC`
`type : STATEMATE`
`usage : REGULAR`
`creator : `
`</chart>` |
| State/Activity Name | `<Activities>`<br>`+ <Activity Id="LOGIN_ACT" Name="LOGIN ACTIVITY">` | `state`
`name : LOGIN_S`
`type : OR`
`parent : REGISTRATION_SC`
`</state>` |
| Transition | `<Transitions>`<br>`+ <Transition From="LOGIN_ACT" id="1" Name="Transition" To="SENDMAIL ACT">`
`<Condition Type="CONDITION">LOGIN_OK</Condition>`
`<ExtendedAttributes>`
`<ExtendedAttribute Name="RoutingType" Value="NOROUTING" />`
`</ExtendedAttributes>`
`</Transition>` | `arrow`
`type : TRANSITION`
`source : LOGIN_S FROM`
`target : SENDMAIL_S TO`
`label : [LOGIN_OK] / [SENDMAIL_ACT]`
`</arrow>` |
| Condition | `<DataFields>`<br>`+ <DataField Id="LOGIN_OK" IsArray="FALSE" Name="LOGIN SHOULD BE OK">`
`<DataType>`
`<BasicType Type="BOOLEAN" />`
`</DataType>`
`</DataField>` | `condition`
`name : LOGIN_OK`
`data_structure : SINGLE`
`</condition>`

**FIGURE XVI: MAPPING ELEMENTS BETWEEN XPDL AND STATEMATE FORMAT**
4.2.1.2 Deploying Mentor-WS main service

The thin Java layer as illustrated in the system architecture provides the Web Service Interface for the clients. The Web Service Interface has been developed using the AXIS Web Service API and the corresponding WSDL shown in figure (XVIII) is deployed on Tomcat Server.

```java
public interface DsiServer {
    void startwf(int wftype, int wfid) ...;
    String get(int wftype, int wfid, String variable) ..;
    short put(int wftype, int wfid, String variable, String activity) ...;
}
```

**FIGURE (XVII): WEB SERVICE INTERFACE FOR MENTOR-WS WORKFLOW SERVER**

```
    <wsdl:message name="startwfResponse" />
    <wsdl:message name="getResponse">
        <wsdl:part name="getReturn" type="xsd:string" />
    </wsdl:message>
    <wsdl:message name="putRequest">
        <wsdl:part name="wftype" type="xsd:int" />
        <wsdl:part name="wfid" type="xsd:int" />
    </wsdl:message>
    <wsdl:message name="mainRequest">
        <wsdl:message name="getRequest">
            <wsdl:message name="mainResponse" />
        </wsdl:message>
        <wsdl:message name="putResponse">
            <wsdl:message name="mainService">
                <wsdl:binding name="DsiServer_MainSoapBinding" type="impl:DsiServer_Main">
                    <wsdl:service name="DsiServer_MainService">
                        <wsdl:port name="DsiServer_Main" soap:address="http://localhost:8080/axis/services/DsiServer_Main" />
                    </wsdl:service>
                </wsdl:binding>
            </wsdl:message>
        </wsdl:message>
    </wsdl:message>
    <wsdl:portType name="DsiServer_Main">
        <wsdl:binding name="DsiServer_MainSoapBinding" type="impl:DsiServer_Main">
            <wsdl:service name="DsiServer_MainService">
                <wsdl:port name="DsiServer_Main" soap:address="http://localhost:8080/axis/services/DsiServer_Main" />
            </wsdl:service>
        </wsdl:binding>
    </wsdl:portType>
</wsdl:definitions>
```

**FIGURE (XVIII): MENTOR-WS WORKFLOW SERVER WSDL**

---

8 Refer to section 2.6.2.2
Clients have to download the WSDL file into their respective machines and view all the operation elements (methods) available to invoke. The service provides three basic operations as depicted in figure (XVII). The main method to start the workflow engine is startwf which requires two parameters as input. The two parameters are workflow type and workflow id. The second method is put which requires four parameters as input. The four parameters are workflow type, workflow id, variable name with its associated value and the activity name. This method will set the variable value for the corresponding activity in the running workflow engine. The third method is get which requires three parameters as input. The three parameters are workflow type, workflow id and the variable name. This method retrieves the value associated with the variable from the running workflow engine. The Clients can access these methods by invoking Web Service Interface deployed at the address location specified in the WSDL. The client has to provide the workflow ID, workflow Type for invoking the corresponding workflow instance. The messages exchanged between the clients and the Mentor-WS’s web service, are the SOAP messages. The sample SOAP request message is shown in Figure (XIX). There are some standard namespaces defined for the SOAP messages.

```xml
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Header />
  <SOAP-ENV:Body>
    <ns1:startwf xmlns:ns1="http://localhost:8080/DsiServer_Main"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <wftype xsi:type="xsd:string">1</wftype>
      <wfID xsi:type="xsd:string">4</wfID>
    </ns1:startwf>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**FIGURE (XIX): A SOAP REQUEST MESSAGE**

The SOAP message in Figure (XIX) contains two SOAP-specific sub-elements within the overall SOAP_ENV:Envelope namely an SOAP_ENV:Header and an SOAP_ENV:Body. The contents of these elements are application defined and not a part of the SOAP specifications, although the latter do have something to say about how such elements must be handled. A SOAP Header is
optional, and is an extension mechanism that provides a way to pass information in SOAP messages that is not application payload. The SOAP Body is the mandatory element within the SOAP Envelope, which implies that this is where the main end-to-end information conveyed in a SOAP message must be carried. All the technical details, such as invoking operations with certain parameters associated with the type are specified in the SOAP Body element. Here, the method invoked is “startwf” with parameters wftype and wfID which are of type string. The values sent are 1 and 4.

4.2.2 Execution phase

Deepweb CMS initiates the execution of a workflow by submitting the business (data) as well as the control flow data. An end user can intuitively submit this data based on his interests like registering for newsletters, updating design layout of the Deepweb web site, etc into JSP/DHTML pages displayed by his browser.

```xml
<% String name = request.getParameter("username");
    String mailid = request.getParameter("mailid");
    int wftype = 1;
    String hostname = "http://localhost:8080";
    if(hostname.equals("\") || mailid.equals("\")){
        casclient.StoreDB storeDB=new casclient.StoreDB();
        if(storeDB.storeName(mailid,wftype)){
            System.out.println("wftype +"+wftype +"mailid +"+mailid +"procid +"+storeDB.getProcessID(mailname));
            new casclient.StartNewThread(wftype,storeDB.getWorkflowID(mailname),hostname);
        }
    }
%>

FIGURE (XX): JSP SERVICE JAVA CLIENT CLASS

In Figure (XX), the JSP page separates the business data from the control flow data. It stores the business data into the MySQL database using JDBC and sends the control flow data using (Apache-AXIS) web service to the workflow engine. Figure (XXI) shows a web service client class for invoking the workflow server. In the CMSClient class, the setTargetEndpointAddress method sets the address of the workflow server (endpoint) to the address location specified in the WSDL. The setOperationName method refers to startwf method of the web service Interface, while the invoke method invokes the startwf method of the workflow server along with two parameters wftype (workflow type) and wfID (workflow id).
public class CMSClient {
    public void startCMSClient(int wftype, int wfid){
        String endpoint = "http://localhost:8080/axis/services/Dsiserver_Main";
        call.setTargetEndpointAddress( new java.net.URL(endpoint) );
        call.setOperationName( method );
    }
}

FIGURE (XXI): A WEB SERVICE JAVA CLIENT CLASS

Mentor-WS service starts the workflow engine with the parameters sent by the client. These parameters when combined with a string (dsiserver) will form a unique name (e.g. dsiserver_1_4). This unique name refers to the particular workflow instance. Mentor-WS interprets the state machine format (state chart) at run time and parses all the states, activities, transitions, conditions, data items, connectors based on each block defined in the state chart as shown in Figure (XV) and stores in the main memory. The control is flown between the states according to the transition conditions. Figure (XXII) shows a sample xml log file. The names of completed activities and, if applicable, the values of conditions can be obtained from this xml log file.

<?xml version="1.0" encoding="UTF-8" ?>
<WorkflowProcess>
  <Activity name="LOGIN_ACT" status="complete">
    <Variable name="LOGIN_OK=1" />
  </Activity>
  <Activity name="SENDMAIL_ACT" status="complete">
    <Variable name="SENDMAIL_OK=1" />
    <Variable name="CONFIRM_OK=1" />
  </Activity>
  <Activity name="COMMIT_ACT" status="complete">
    <Variable name="COMMIT_OK=1" />
  </Activity>
</WorkflowProcess>

FIGURE (XXII): XML LOG FILE

In Figure (XXII), there are Activity elements with attributes, name and status. The status can be either complete or incomplete. It indicates whether the activity is processed or not in a workflow instance. For example, in the figure, the status is complete for the Activity LOGIN_ACT, which
means the processing of the activity \textit{LOGIN_ACT} is done. The Variable element specifies the variable name and its value. This variable and its value are set in the workflow engine after the activities are processed. For example, in the figure, variable \textit{LOGIN_OK}=1 is set after processing the activity \textit{LOGIN_ACT}. The details about the sample application wrappers, database schema are specified in Appendix A.3.

4.3 Sample Applications

This section sketches two sample scenarios, each describing the realization of two different goals of the prototype. The first sample application process (Newsticker) exemplifies the integration of an automated workflow inside a content management system. The second sample application process (Design Layout) also describes an automated workflow in a CMS.

4.3.1 Newsticker

This sample application process illustrates the simple and effective usage of a workflow in Deepweb CMS. It is essential for any person or organization to keep their customers, investors, and employees informed about new products, services and announcements. Providing email communications in form of a newsletter is the easiest, least expensive, and most effective way to keep everyone informed about the latest news and announcements. Deepweb CMS includes a component called Newsticker that offers exactly this functionality. It lets the users/customers know about the latest happenings of the company and thus the number of visits to the Deepweb site is increased. This process is explained as follows:

4.2.1.3 Design phase

A) Workflow Modeling

Let’s assume that the basic steps of defining a package, processes, workflow data items, workflow conditions, workflow participants are already done. Now the workflow process has to be modeled which generates an xpdl file from the modeled workflow process. Process definition is shown in Figure (XXV). JaWE-to-Statemate converter converts xpdl file into state charts, activity charts. For converter, XPDL file, state chart format refer to Figure (XVIII).
The process definition of Newsticker application is shown in Figure (XXIII). This application has a unique workflow type id. The rectangle boxes are the states and the arrows are the transitions from one state to another. For every process, there are atleast two states, start and end state that needs to be defined. There are two roles defined in this application namely auto and administration which is shown at the left side of the figure. The activities which come below the center line are to be performed by the administration role; remaining activities are automatically performed by the external activities. Roles are assigned to the actors through the worklist manager which is shown in the Figure (XXIV). Worklist manager is one of the components of Workflow manager. After starting the Worklist manager, one can view the list of actors available, list of roles available which were defined and assigned to workflow activities using JaWE, and the roles assigned to the actors. One can assign/modify actors to roles from this Worklist manager component.
Available actors, available roles and the list of roles assigned to actors can be viewed through the Worklist Manager and are shown in the Figure (XXV). For every actor, there is an actor id. Depending on the status and the workload, the workflow activities are assigned to the actors. There is a role id and a workflow type associated with every role.
4.2.1.4 Execution phase

Customers visit Deepweb’s web site and register to receive the announcements of Deepweb GmbH. The registration page of Deepweb is shown in Figure (XXVI). A user has to enter his name and email id in to the registration form. When the user clicks the register button, Deepweb CMS invokes Mentor-WS via web services with a unique workflow id. Mentor-WS interprets the workflow specifications at run time and the control is flown according to the transition conditions. In Figure (XXIII), there is a total of 5 states excluding start and end states namely, LOGIN_ACT, SENDER, DELETE, COMMIT, and ROUTE1. The state named ROUTE1 acts a dummy state which simply passes the control to the next state. For each state there is an external wrapper defined but for dummy states there aren’t any. LOGIN state is the first state to be executed in the flow. In this state, email validation is done. If the email id is invalid then the user’s information is
deleted from the database and the Mentor-WS exits via the state \textit{ROUTE1} else sets \textit{LOGIN\_OK} condition to true and passes the control back to the engine. The control flow between the states depends on the condition value set by the processed activity for the condition variable. Let us assume that the \textit{LOGIN\_OK} condition is set to true, then the engine enters into the \textit{SENDMAIL} state. In this state, the mail is sent to the user for the confirmation and passes back the control to the engine. The engine waits until the user clicks the link in the mail sent by Mentor-WS. At regular time intervals, the engine checks a \textit{putfile} in which the results of all the interactive activities are put. When the user clicks the link, Java servlets in the web server sets the condition \textit{CONFIRM\_OK} to true and puts this condition in the \textit{putfile}. Let us assume that user has clicked the link and so the condition is set and put in the file. Now the engine enters into the \textit{COMMIT} state. In this state, Worklist manager assigns new task to the Administrator about the new user. The task here is to commit the user’s information permanently in the database so that the user could receive the news letters regularly. Administrator will check in the worklist viewer (JSP) for the new tasks regularly. When the administrator performs his task he sets the condition \textit{COMMIT\_OK} to true and thus the engine exits passing through the dummy state \textit{ROUTE1}.
4.3.2 Content Publication

The sample application process illustrates the functional scope of a typical CMS and is informally described as follows: The process starts with the web developers (Authors) creating and structuring the content, the professional web designers designing a web page format, which involves specifying the logo, standard navigation options etc. This common web page format is then used to create a master template. All the web developers (Authors) use this master template in generating the content in an environment that is most appropriate for this task. This authored and developed content is forwarded to the editor for further perusal. An editor validates or rejects the content after checking the semantics of the content. The content is forwarded to the publisher if it has been reviewed and accepted by the editor. If the content is rejected, editor notifies the author. After publishing the content online, the publisher notifies the author, that his content is published online. This sample application process is depicted in the Figure (XXVII) using a directed graph. In this graph, the nodes specify the different states of the workflow and their transitions are specified by their directed edges between the nodes. The functions of the participants are also depicted in the figure.

![Diagram](image-url)

**FIGURE (XXVII): SAMPLE SCENARIO – CONTENT PUBLICATION**
4.4 Implementation Issues

This section discusses some of the problems that we encountered while implementing Mentor-WS system and integrating it with Deepweb CMS using web services.

4.4.1 Web Services Related Problems

When we started to use web services for integrating Mentor-WS with Deepweb CMS, we used JWSDP package developed by SUN which uses Ant tool for deploying and accessing web services. We encountered some problems with accessing a web service from another web service using Ant tool. The reason for this problem is still unclear for us. So we switched to Apache Axis framework which supported the composition of web services, easy framework for hosting and accessing web services. Alternative technologies which solve this problem are Microsoft .NET framework, IBM websphere SDK. The reason for choosing Axis framework is due to its simplicity in using web services.

Since Mentor-WS uses JNI function calls, the main web service has to invoke JNI function calls on Apache Axis to start the core engine of Mentor-WS and so we were getting the exception “Unsatisfied link error” which means the system is not able to locate Mentor-WS’ dynamic link library. We solved this problem by putting the dll in the correct location (windows/system32) which is not mentioned in any of the documents correctly.

4.4.2 Mentor-WS Related Problems

We encountered a problem related to multiple threads. The main JVM instance (process) is created by tomcat itself in which each request to the Mentor-WS’s web service is a thread associated with a unique name. For each process, only one dynamic link library is loaded in the main memory. Thereby the conflicts are occurred due to global and static variables which reside in the common memory location for all the threads. As soon as the first thread is created in java, it invokes methods in ‘C’ to start the core engine (using JNI), and the static and global variables are initiated. When the second thread enters into ‘C’ which also accesses the same dll and so the same static variables, the first thread is somehow suspended and second thread continues running till the termination. Likewise the next thread starts by suspending the running thread. We had two options to solve this issue. The first one is to create a java virtual machine for each thread thereby each thread would become a process and so each process separately loads dll into the memory and so no conflicts occur due to static and global variables. This is the approach we followed in this Thesis. But there are some performance issues regarding main memory space usage. This is because each thread loads dll separately at different locations in main memory which requires
more space. The second option is to implement a round robin scheduling for threads. Each thread is given a certain time limit and stored in a queue. When a thread is in the running state, all the other threads will be in the sleep state. The first thread loads the dll and starts the core engine and stores the results (variable names with values and activities that are already processed) into an xml file with the file name as the thread name. After the time limit, the thread terminates and the thread object is stored at the last position in queue. Next thread from the queue is started and the core engine starts processing and the thread is terminated after the time limit. After all the threads are processed, again the first thread is started and it restores its previous state in its last run by the restore mechanism offered by Log Manager. Thus all the threads run till their execution is completely done.

4.4.3 Deepweb CMS Related Problems

In Mentor-WS we used all the latest versions of all the softwares like MySQL 5.0 (alpha version) for storing workflow information. But later when we tried to integrate it with Deepweb CMS we encountered that Deepweb CMS uses PhpMyAdmin extensively as the graphical interface for MySQL database. But PhpMyAdmin doesn’t support MySQL 5.0. So we had to replace these softwares with MySQL 4.1.6 which PhpMyAdmin supports.
CHAPTER 5 CONCLUSION AND FUTURE WORK

Summary

This thesis elucidates the possible advantages of integrating a WfMS with CMS by presenting the integration of Mentor-WS WfMS with Deepweb CMS. To enable this integration, Mentor-WS WfMS was enhanced from Mentor-lite. The overview of workflow management systems, content management systems, different process definition languages, and service oriented architectures has been presented in the Preliminaries chapter. Design chapter presents an overview of Deepweb CMS and Mentor-WS and an evaluation of the existing system (Deepweb CMS). The positive aspects of Deepweb CMS include easy way of creating, managing, publishing content and lot more. The negative aspect is the lack of workflow system in Deepweb CMS which automates the entire process of content creation, management and publication. The positive aspects of Mentor-WS are light weight, scalable, persistent in nature, support for the establishment of workflow standards by adopting the standards proposed by the WfMC, while building its system architecture and provides support to platform independency and open source components (unlike Mentor-Lite) making it highly accessible for users around the world to evaluate it with much ease. The major design goals of novel Deepweb CMS include the integration of Mentor-WS using web services as the middleware since web services accomplish an interoperability feature and loose coupling nature between Deepweb CMS and Mentor-WS. Further, Mentor-WS provides a flexible framework for the integration of Deepweb CMS by the application of web services and the usage of various open source components. Deepweb CMS and Mentor-WS use the common database to store the business and workflow information. The workflow specifications for Mentor-WS are defined graphically using Java Workflow Editor (JaWE) tool which generates workflow specifications in XML Process Definition Language (XPDL) and is converted into statemate format which Mentor-WS’s core engine understands. Here, a converter has been developed to convert XPDL format to statemate format because the tool, which generates statemate format directly, is a commercial one but JaWE is a free tool. Another reason behind this multiple conversions (JaWE-XPDL and XPDL-statemate) is the lack of one popular workflow processing standard. The whole project (thesis) makes the Deepweb CMS accessible to even novice users, who are not computer wizzards. More technical details of Mentor-WS are illustrated in Appendix A.1.
**Future work**

The limitation of Mentor-WS is that the workflow engine distribution is not supported. So we would like to develop an API for achieving distributed execution of workflows. The API should be capable of generating different workflow instances for execution by various workflow servers (at different sites) from a given state and activity chart workflow specification without hampering its semantics. There are some performance issues of Mentor-WS when multiple threads run. This problem has been mentioned in the section 4.4.2. We would like to implement the alternative approach that was discussed in that section. The other limitation of the Mentor-WS WfMS is that even though the prototype has been built to work on any platform, it has not been implemented and tested on any platform other than windows. So we would like to develop and successfully test the prototype on various platforms such as Linux, Solaris etc. The other enhancement would be to provide a more sophisticated and user-friendly, features for the existing process definition tool (JaWE). The existing process definition tool (JaWE), can only provide XPDL support for describing information about a particular application associated with an activity rather than mapping it to a specific application (such as a JFC class or Java Bean or Java applet etc). Further, application templates (such as send/check mail, calculate expressions/formulas etc) can be developed for more generic activities of the workflow. These templates can then be presented at the modeling phase to the user for mapping it to a particular activity. This approach provides a more user-friendly, flexible, and reusable process definition framework to the user.
APPENDIX

This section provides a technical overview of the Mentor-WS WfMS. Section A.1 presents the package hierarchy and outlines the classes associated with each package. The section also provides some code snippets to illustrate the usage of important methods in the package. Section A.2 presents the essential libraries that have to be included in the Mentor-WS WfMS, while section A.3 describes the database schemas. Section A.4 concludes this chapter by stating the system requirements.

A.1 Mentor-WS Module Structure

The Mentor-WS module comprises of four main packages as illustrated in figure (XXVIII).

![Package Hierarchy Diagram](image)

**FIGURE (XXVIII): MENTOR-WS PACKAGE HIERARCHY**

### A.1.1 DSISERVER

This package contains all classes and interfaces needed for implementing the *thin Java layer* which acts as a Web Service interface for clients and as a gateway between mentor workflow engine and external wrappers. This package also contains classes for establishing database communication and registering workflow objects using JNDI registry. Table 1 presents all the classes in this package and important methods in each class. The *DsiServer_i* interface defines all methods needed for implementing Web Services. The *DsiServer* class implements the above stated interface and provides the Mentor-WS Workflow functionality to the clients. The four important methods in this package are depicted in the figure (XXIX).
public void startwf (int wftype, int wfid, int procid) //Starts Workflow Engine
public short put (String variable, String activity) //Commits Workflow Condition
public String get (String variable) //retrieves value of a Workflow Variable
public startExtWrapper (String[] args) //Starts a specified External Wrapper

FIGURE (XXIX): FOUR IMPORTANT METHODS OF DSISERVER CLASS

<table>
<thead>
<tr>
<th>CLASS NAME</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
<th>IMPORTANT METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBConnection</td>
<td>Public Class</td>
<td>Establishes database connections</td>
<td>void getConnection()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>java.sql.connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>closeConnection (String databaseName)</td>
</tr>
<tr>
<td>DsiServer</td>
<td>Public Class</td>
<td>Provides web service interface methods</td>
<td>String get (String variable)</td>
</tr>
<tr>
<td></td>
<td>(Implements</td>
<td></td>
<td>short put (String variable, String activity)</td>
</tr>
<tr>
<td>DsiServer_i)</td>
<td></td>
<td></td>
<td>void startwf (int wftype, int wfid, int procid)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void startExtWrapper (String[] args)</td>
</tr>
<tr>
<td>DsiServer_Main</td>
<td>Public Class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DsiServerConnector</td>
<td>Public Class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DsiServerEngine</td>
<td>Public Class</td>
<td>Provides methods for controlling the</td>
<td>void ProcessResume (String wftype, String wfid, int</td>
</tr>
<tr>
<td></td>
<td></td>
<td>execution of workflows</td>
<td>procid)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void ProcessSleep (String wftype, String wfid, int</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>procid)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void ProcessSuspend (String wftype, String wfid, int</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>procid)</td>
</tr>
<tr>
<td>DsiServerObject</td>
<td>Public Class</td>
<td>Maintains information regarding all</td>
<td>DsiServer setDsiServerObject (String wfengine, DsiServer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DsiServer objects</td>
<td>dssrv)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DsiServer getDsiServerObject (String wfengine)</td>
</tr>
</tbody>
</table>
**DsiServer**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lookupDsiServer</td>
<td>(String wfengine)</td>
</tr>
</tbody>
</table>

**DsiServerPut**

<table>
<thead>
<tr>
<th>Public Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commits conditions into workflow engine</td>
<td>void putinMentor (int actorid, String actname, String variable)</td>
</tr>
</tbody>
</table>

**DsiServer_i**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides methods for retrieving workflow objects from JNDI registry</td>
<td>Object getobjectInstance (Object obj ,javax.naming.Name name, javax.naming.Context ctx, java.util.Hashtable env)</td>
</tr>
</tbody>
</table>

**MentorFactory**

<table>
<thead>
<tr>
<th>Public Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides methods for retrieving workflow objects from JNDI registry</td>
<td>Object getobjectInstance (Object obj ,javax.naming.Name name, javax.naming.Context ctx, java.util.Hashtable env)</td>
</tr>
</tbody>
</table>

**RegisterObject**

<table>
<thead>
<tr>
<th>Public Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.naming.Reference</td>
<td>getReference ()</td>
</tr>
</tbody>
</table>

**Util**

<table>
<thead>
<tr>
<th>Public Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getProcID (int wftype)</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 1** Class Summary for ORG.MENTOR.DSISERVER package

**A.1.2 EXTWRAPPERS**

This package contains classes, which provide functionality needed for interacting with external wrappers. The package consists of two main classes. Table 2 presents these two classes and important methods associated with it. The ExtWrappers class invokes the specified application wrapper, while the TimeOuts class keeps track of the timeouts. The **startExtWrapper** method is the most important of all the methods in this package. This method invokes the specified application wrapper using the Java reflection API as shown in the figure (XXX).

```java
public static void startExtWrapper(String args[]) {
    Integer WFTY = new Integer(args[0]);
    Integer WFID = new Integer(args[1]);
    Integer PROC = new Integer(args[2]);
    paramTypes = new Class[] {DsiServer.class,WFTY.getClass(),WFID.getClass(),PROC.getClass()};
    ...
    obj = (Object) Class.forName ("org.mentor.mentorExtWrappers."+packname.toUpperCase()+"."+actname.toUpperCase().toLowerCase()).newInstance();
    ...
    DsiServer dssrv = DsiServerObject.getDsiServerObject(wfengine);
    Object[] inargs = new Object[] {dssrv,WFTY,WFID,PROC};
    methd = obj.getClass().getMethod("startWrapper", paramTypes);
    methd.invoke(obj, inargs); // Invokes the startWrapper method of the specified Activity class
    ...
} // with DsiServerObject, workflow Type, workflow ID & process ID as parameters.
```
A.1.3 WORKFLOW

This package contains all classes and interfaces needed for implementing the GUI based Workflow manager, which includes the worklist manager, statechart viewer, statechart converter, control flow viewer, log and recovery managers. The package also contains classes, which maintain information regarding the statechart workflow specifications, information and status of the workflow engine. Table 3 presents the class summary for this package while the important methods are listed in the figure (XXXI).

```java
public void suspendWFEngine (String wfengine) // Suspend Workflow Execution
public void wfRecovery(String xmlfile) // Perform Workflow Recovery
//Assign Roles to Actors
public void setActorToRoleName (String actorName, String roleName, String wfType)
//Assign workitems to Actors
public void assignWorkToActors (String actname, int priority, String rolename, String href)
public String getLogFile() // Retrieve Workflow Log File
```
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFManager</td>
<td>Public Class</td>
<td>This class generates a frame with overall information about the workflow system.</td>
</tr>
<tr>
<td>WLManager</td>
<td>Public Class</td>
<td>This class generates a frame with the information about the worklist management of a WFMS.</td>
</tr>
<tr>
<td>WorkFlow</td>
<td>Public Class</td>
<td>Maintains basic information about workflows.</td>
</tr>
<tr>
<td>WorkFlowAction</td>
<td>Public Class</td>
<td>Provides functionality for controlling workflow execution.</td>
</tr>
<tr>
<td>WorkFlowActivity</td>
<td>Public Class</td>
<td>Maintains information about all workflow activities.</td>
</tr>
<tr>
<td>WorkFlowActor</td>
<td>Public Class</td>
<td>Maintains information about all workflow actors.</td>
</tr>
<tr>
<td>WorkFlowArrow</td>
<td>Public Class</td>
<td>Maintains information about all workflow arrows.</td>
</tr>
<tr>
<td>WorkFlowChart</td>
<td>Public Class</td>
<td>Maintains header information about all workflow charts.</td>
</tr>
</tbody>
</table>

For interacting with log files in XML:

- **writeToFile** (String actname, String variable, String str)
- **setTextArea** (java.awt.TextArea logInfo)

```java
WFManager
Public Class
This class generates a frame with overall information about the workflow system.

void setTextArea(java.awt.TextArea logInfo)
```

```java
WorkFlow
Public Class
Maintains basic information about workflows.

void setAuthor(String author)
void setBackupVersionNumber(String backup)
void setConfigVersionNumber(String cfgversion)
```

```java
WorkFlowAction
Public Class
Provides functionality for controlling workflow execution.

short putVariableValue(String variable, String activity)
void startwf(int wftype, int wfid, int procid)
void suspendWFEngine(String wfengine)
```

```java
WorkFlowActivity
Public Class
Maintains information about all workflow activities.

void setName(String name)
void setImplementedBy(String impl)
void getName()
```

```java
WorkFlowActor
Public Class
Maintains information about all workflow actors.

int getActorID()
String getActorName()
int getWorkLoad()
```

```java
WorkFlowArrow
Public Class
Maintains information about all workflow arrows.

void setSource(String source)
void setTarget(String target)
void setLabel(String label)
```

```java
WorkFlowChart
Public Class
Maintains header information about all workflow charts.

void setName(String name)
```
<table>
<thead>
<tr>
<th>Class</th>
<th>Type</th>
<th>Description</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>WorkFlowCondition</td>
<td>Public Class</td>
<td>Maintains Information about all workflow actors</td>
<td>void setName (String name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setDataStructure (String data_structure)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void getName ()</td>
</tr>
<tr>
<td>WorkFlowConfigFiles</td>
<td>Public Class</td>
<td>Maintains configuration details about all workflow specification files</td>
<td>void WorkFlowConfigFiles (String wftype, String packagename, String processname)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setName (String name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setParent (String parent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void getName ()</td>
</tr>
<tr>
<td>WorkFlowConnector</td>
<td>Public Class</td>
<td>Maintains Information about all workflow connectors</td>
<td>void setName (String name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setParent (String parent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void getName ()</td>
</tr>
<tr>
<td>WorkFlowDataItem</td>
<td>Public Class</td>
<td>Maintains Information about all workflow dataitems</td>
<td>void setName (String name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setDataType (String data_type)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void getName ()</td>
</tr>
<tr>
<td>WorkFlowEntity</td>
<td>Public Class</td>
<td>Maintains Information about all workflow entities</td>
<td>void setName (String name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setEntityId (String entityid)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void getName ()</td>
</tr>
<tr>
<td>WorkFlowFiles</td>
<td>Public Class</td>
<td>Reads workflow files in state &amp; activity chart formats</td>
<td>String getLogFile()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>String getWFConfigFile()</td>
</tr>
<tr>
<td>WorkFlowInfo</td>
<td>Public Class</td>
<td>This class provides information about all workflow specifications</td>
<td>java.util.Hashtable getAllProcesses()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>java.util.Hashtable getAllStates(int processId)</td>
</tr>
<tr>
<td>WorkFlowModule</td>
<td>Public Class</td>
<td>Maintains Information about all Module chart specifications</td>
<td>void setName (String name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setType (String type)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void getName ()</td>
</tr>
<tr>
<td><strong>WorkFlowProcess</strong></td>
<td><strong>Public Class</strong></td>
<td>Maintains Information about all processes defined in the workflow</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>void setName (String name)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>void setProcid (String procid)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>void getName ()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>WorkFlowRole</strong></th>
<th><strong>Public Class</strong></th>
<th>Maintains Information about all roles defined in a workflow process</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>void setName (String name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void setRoleid (String roleid)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void getName ()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>WorkFlowSetInfo</strong></th>
<th><strong>Public Class</strong></th>
<th>Sets workflow process Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Implements WorkFlowInfo_i)</td>
<td></td>
<td>void WorkFlowSetInfo (String wfConfigFile)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void setProcessInfo (WorkFlowProcess wfp)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void setChartInfo (WorkFlowChart wc, String filename)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>WorkFlowState</strong></th>
<th><strong>Public Class</strong></th>
<th>Maintains Information about all states defined in a workflow process</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>void setName (String name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void setSubWorkFlow (String is)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void getName ()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>WorkFlowStatus</strong></th>
<th><strong>Public Class</strong></th>
<th>Provides workflow status</th>
</tr>
</thead>
<tbody>
<tr>
<td>(implements WorkFlowStatus_i)</td>
<td></td>
<td>WorkFlowProcess</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getCurrentProcess (int ProcessId)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WorkFlowState</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getCurrentState (String filename)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vector wfActivity_Recovery (String filename)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void wfRecovery(String xmlfile)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>WorkListManager</strong></th>
<th><strong>Public Class</strong></th>
<th>Provides WorkList Manager functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>void setActorToRoleName (String actorName, String roleName, String wfType)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void addRoles (String roleid, String roleName, int wftype)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void assignWorkToActors (String actname, int priority, String rolename, String href)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>void updateWorkLoad (int actorId)</td>
</tr>
</tbody>
</table>
WorkFlowInfo_i  Interface  Provides methods for retrieving workflow information

WorkFlowStatus_i  Interface  Provides methods for retrieving status of workflow.

Table 3 Class Summary for ORG.MENTOR.WORKFLOW package

A.1.4 JAWEMENTOR

This package consists of classes (as listed in Table 4) that enable the conversion of workflow specifications in XPDL modeled using JaWE to an intermediate state and activity chart workflow specifications. The package consists of two interfaces namely JaWEInf and JaWEMentorInf, which define methods for parsing XPDL file and converting the XPDL into state and activity chart representations. The parseXpdl class of this package implements the JaWEInf interface, thereby providing the functionality required for parsing a XPDL file using the JDOM API. The JaWEMentor class implements the JaWEMentorInf interface, thereby enabling the conversion of XPDL. The important methods in this package are depicted in the figure (XXXII)

FIGURE (XXXII): IMPORTANT METHODS OF JAWEMENTOR PACKAGE

<table>
<thead>
<tr>
<th>CLASS NAME</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
<th>IMPORTANT METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorClass</td>
<td>Public Class</td>
<td>Provides Error information</td>
<td>String setErrorDescription()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>String setErrorNumber()</td>
</tr>
<tr>
<td>JaWEMentor</td>
<td>Public Class</td>
<td>Provides Conversion of XPDL workflow specification to State and activity representation</td>
<td>void JaWEMentor(String XpdlSource)</td>
</tr>
<tr>
<td></td>
<td>(Implements JaWEMentorInf)</td>
<td></td>
<td>void convert()</td>
</tr>
<tr>
<td>JaWEMentorActivityChart</td>
<td>Public Class</td>
<td>Builds Activity Charts from XPDL workflow specifications</td>
<td>void JaWEMentorActivityChart(String XpdlSource, String ProcessId)</td>
</tr>
<tr>
<td>Class</td>
<td>Type</td>
<td>Description</td>
<td>Method</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>JaWEMentorConfig</td>
<td>Public Class</td>
<td>Builds Mentor configuration from XPDL workflow specifications</td>
<td>void JaWEMentorConfig(String XpdlSource)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(String XpdlSource, String SourceProcess)</td>
</tr>
<tr>
<td>JaWEMentorModuleChart</td>
<td>Public Class</td>
<td>Builds Module Charts from XPDL workflow specifications</td>
<td>void JaWEMentorModuleChart(String XpdlSource)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(String XpdlSource, String SourceProcess)</td>
</tr>
<tr>
<td>JaWEMentorStateChart</td>
<td>Public Class</td>
<td>Builds State Charts from XPDL workflow specifications</td>
<td>void JaWEMentorStateChart(String XpdlSource, String SourceProcess)</td>
</tr>
<tr>
<td>ValidateJaWETF</td>
<td>Public Class</td>
<td>Validates the XPDL workflow Specifications with reference to State &amp; Activity chart specifications</td>
<td>void ErrorClass[] ValidateWF(String sourceXPDL, String sourceProcess, boolean DebugMode)</td>
</tr>
<tr>
<td>JaWEMentorInf</td>
<td>Interface Class</td>
<td>Provides methods for conversion of XPDL to state &amp; activity chart format</td>
<td></td>
</tr>
<tr>
<td>JaWEInf</td>
<td>Interface Class</td>
<td>Provides Interface methods for parsing XPDL workflow specifications</td>
<td></td>
</tr>
<tr>
<td>Activity</td>
<td>Public Class</td>
<td>Maintains Information about various activities defined in a XPDL workflow specification</td>
<td>String getActivityId()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setSubFlowID(String SWFlowID)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setImplementationType(String IType)</td>
</tr>
<tr>
<td>Connector</td>
<td>Public Class</td>
<td>Maintains Information about various connectors defined in a XPDL workflow specification</td>
<td>void setConnectActivity(String ConnAct)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setConnectorType(String ConnType)</td>
</tr>
<tr>
<td>DataField</td>
<td>Public Class</td>
<td>Maintains Information about various datafields defined in a XPDL workflow specification</td>
<td>void setDataFieldName(String DFName)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>void setIsArray(String Isarray)</td>
</tr>
</tbody>
</table>
ParseXpdl  
*Public Class*  
*(Implements JaWEInf)*  
Reads and parses XPDL files using JDOM  
java.util.Vector `getSourceProcess()`  
Participant[] `getParticipants()`  
java.util.Vector `getWorkflowProcesses()`  
java.util.Vector `getSubFlows(String WorkflowProcess)`  

Participant  
*Public Class*  
Maintains Information about various participants defined in a XPDL workflow specification  
String `getParticipantId()`  
String `getParticipantName()`  
String `setParticipantType(String PType)`  

ProcessInfo  
*Public Class*  
Maintains Information about various processes defined in a XPDL workflow specification  
String `getFormalParameterMode()`  
String `getFormalParameterType()`  
void `setFormalParameterID(String FormalParameterID)`  
(String FormalParameterID)  

Transition  
*Public Class*  
Maintains Information about various transitions defined in a XPDL workflow specification  
String `getSourceActivity()`  
String `getTargetActivity()`  
String `getTransitionID()`  

**Table 4** Class Summary for ORG.MENTOR.JAWEMENTOR package

### A.2 Library Dependencies
The libraries listed in Table 5 are needed in your CLASSPATH or installation directory lib directory.

<table>
<thead>
<tr>
<th>LIBRARY NAME</th>
<th>DEPENDENCIES</th>
<th>DESCRIPTION</th>
<th>AVAILABLE AT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache AXIS</td>
<td>axis.jar, axis-ant.jar, commons-discovery.jar, commons-logging.jar, log4j-1.2.8.jar, jaxrpc.jar, saaj.jar, wsdl4j.jar</td>
<td>Needed for Implementing Web Services Framework</td>
<td><a href="http://apache.easywebs.de/ws/axis/1_1/axis-1_1.zip">http://apache.easywebs.de/ws/axis/1_1/axis-1_1.zip</a></td>
</tr>
</tbody>
</table>
### Bibliography

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI</td>
<td>fcontext.jar, providerutil.jar</td>
<td>Needed for accessing JNDI registry for storing mentor objects</td>
</tr>
<tr>
<td>JDOM</td>
<td>jdom.jar</td>
<td>Needed for Parsing XPDL files and writing log information into XML files</td>
</tr>
<tr>
<td>DeepWebCMS</td>
<td>Deepwebcms.jar</td>
<td>Needed for integrating Mentor-WS with DeepWeb CMS</td>
</tr>
<tr>
<td>Mentor-WS</td>
<td>DsiServer_main.dll, DsiServer_main.lib</td>
<td>Needed for interacting with mentor-ws workflow engine (core)</td>
</tr>
<tr>
<td>JNI</td>
<td>jni.h, jvmdi.h, jvmpi.h, jni_md.h</td>
<td>Needed for JNI calls to C</td>
</tr>
</tbody>
</table>

#### Table 5 Essential Libraries for MENTOR-WS

### A.3 Database Schemas

#### CREATE TABLE **ACT2URL** (activityname VARCHAR(40) NOT NULL DEFAULT '', href VARCHAR(255) NOT NULL DEFAULT '', PRIMARY KEY (activityname, href));

#### CREATE TABLE **ACTIVITY** (username VARCHAR(40) NOT NULL DEFAULT '', name VARCHAR(40) DEFAULT NULL, customer VARCHAR(40) DEFAULT NULL, description VARCHAR(40) DEFAULT NULL, wftype INT(11) DEFAULT NULL, wfid INT(11) DEFAULT NULL, procid INT(11) DEFAULT NULL, priority INT(3) DEFAULT NULL, start_time DATE DEFAULT NULL, deadline DATE DEFAULT NULL, status VARCHAR(10) DEFAULT NULL, href VARCHAR(255) NOT NULL DEFAULT '', PRIMARY KEY (username, href));

#### CREATE TABLE **ACTIVITY_TMP** (count INT(8) NOT NULL DEFAULT '0', actorname VARCHAR(40) DEFAULT NULL, activityname VARCHAR(40) DEFAULT NULL, PRIMARY KEY );
CREATE TABLE `ACTORS` (
  `id` INT(10) NOT NULL DEFAULT '0',
  `name` VARCHAR(40) DEFAULT NULL,
  `status` VARCHAR(40) DEFAULT NULL,
  `workload` INT(10) DEFAULT NULL,
  `Email` VARCHAR(20) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`));

CREATE TABLE `ENVIRON_VARS` (
  `mlite_home` VARCHAR(30) NOT NULL DEFAULT '',
  `jawe_home` VARCHAR(30) NOT NULL DEFAULT '',
  `jndi_registry` VARCHAR(30) NOT NULL DEFAULT ''
);}

CREATE TABLE `PLAYSROLE` (
  `actorID` INT(10) NOT NULL DEFAULT '0',
  `role_name` VARCHAR(20) NOT NULL DEFAULT '',
  `wftype` INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`actorID`, `role_name`, `wftype`));

CREATE TABLE `ROLES` (
  `roleid` INT(3) NOT NULL DEFAULT '0',
  `rolename` VARCHAR(20) NOT NULL DEFAULT '',
  `wftype` INT(11) NOT NULL DEFAULT '0'
);}

CREATE TABLE `USERS` (
  `name` VARCHAR(30) NOT NULL DEFAULT '',
  `email` VARCHAR(30) NOT NULL DEFAULT '',
  `passwd` VARCHAR(30) NOT NULL DEFAULT '',
  `status` INT(11) NOT NULL DEFAULT '0',
  `wftype` INT(11) NOT NULL DEFAULT '0',
  `wfid` INT(11) NOT NULL AUTO_INCREMENT,
  `procid` INT(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`wfid`));

CREATE TABLE `WFENGINE` (
  `wftype` INT(11) NOT NULL DEFAULT '0',
  `wfid` INT(11) NOT NULL AUTO_INCREMENT,
  `procid` INT(11) NOT NULL DEFAULT '0',
  `starttime` VARCHAR(20) NOT NULL DEFAULT '',
  `endtime` VARCHAR(20) NOT NULL DEFAULT '',
  PRIMARY KEY (`wftype`, `wfid`, `procid`));
CREATE TABLE WFPACKAGES (  
  ID     INT(11)   NOT NULL  DEFAULT '0',  
  packagename   VARCHAR(30)  NOT NULL  DEFAULT '',  
  xpdlfile    VARCHAR(50)  NOT NULL  DEFAULT '',  
  processid    INT(11)   NOT NULL  DEFAULT '0',  
  PRIMARY KEY  ( ID )  
);  

CREATE TABLE WFTIMEOUTS (  
  wftype   INT(11)   NOT NULL  DEFAULT '0',  
  wfid    INT(11)   NOT NULL  DEFAULT '0',  
  procid   INT(11)   NOT NULL  DEFAULT '0',  
  id    INT(11)   NOT NULL  AUTO_INCREMENT,  
  activity   VARCHAR(20)  NOT NULL  DEFAULT '',  
  begin    DATE    DEFAULT  NULL,  
  end    DATE    DEFAULT  NULL,  
  PRIMARY KEY  (wftype, wfid, procid ,id)  
);  

A.5 System Requirements

The current version of Mentor-WS prototype has been successfully tested on Windows XP professional edition. Table 6 lists the softwares that have to be installed along with the dependent libraries listed in table 5 for successful installation of Mentor-WS prototype. The prototype can be installed on any system, which has a minimum processing speed of 400 MHz (Pentium Celeron/II/III or AMD K6/K7) along with 128 MB (RAM) memory and atleast 200 MB of free space.

<table>
<thead>
<tr>
<th>Software Title</th>
<th>Version</th>
<th>Platform</th>
<th>Description</th>
<th>Available at</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>J2SE</td>
<td>Windows</td>
<td></td>
<td><a href="http://java.sun.com/j2se/1.4.2/download.html">http://java.sun.com/j2se/1.4.2/download.html</a></td>
</tr>
<tr>
<td></td>
<td>1.4.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JaWE</td>
<td>1.3</td>
<td>Windows</td>
<td>Workflow Modeling tool</td>
<td><a href="http://jawe.objectweb.org/">http://jawe.objectweb.org/</a></td>
</tr>
<tr>
<td>MySQL</td>
<td>4.1.4 -</td>
<td>Windows</td>
<td>Database</td>
<td><a href="http://dev.mysql.com/downloads/mysql/4.1.html">http://dev.mysql.com/downloads/mysql/4.1.html</a></td>
</tr>
<tr>
<td></td>
<td>gamma</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tomcat</td>
<td>4.1.6</td>
<td>Windows</td>
<td>Standalone Web Server</td>
<td><a href="http://jakarta.apache.org/site/binindex.cgi">http://jakarta.apache.org/site/binindex.cgi</a></td>
</tr>
<tr>
<td>Apache HTTP</td>
<td>1.3</td>
<td>Windows</td>
<td>HTTP Server</td>
<td><a href="http://httpd.apache.org/download.cgi">http://httpd.apache.org/download.cgi</a></td>
</tr>
<tr>
<td>Server</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 6 Software Requirements for MENTOR-WS*
BIBLIOGRAPHY


[WGR+00] Jeanine Weissenfels, Michael Gillmann, Olivier Roth, German Shegalov, Wolfgang Wohner.: The Mentor-lite Prototype: A Light-Weight Workflow Management System (Demo Description), 16th International Conference on Data Engineering, San Diego, CA, USA, 2000.
