CHAPTER 4

PROTOTYPE DEVELOPMENT AND TESTING

The main objective of the study is to identify the needs of the farmers, and then develop and evaluate an integrated platform for realizing on-line agricultural information dissemination. Prototyping can improve the quality of requirements and specifications provided to developers. Since users know the problem domain better than anyone in the development team does, increased interaction can result in final product that has greater tangible and intangible quality. The final product is more likely to satisfy the users’ expectations from the prototype. Prototyping requires user involvement and allows them to see and interact with the prototype allowing them to provide better and more complete feedback and specifications during the testing as well as the development phase.

The prototype platform was proposed for the mushroom component of agriculture which was to be deployed and evaluated at a suitable platform. The on-line portal would include specification for featuring a number of advanced characteristics i.e. access from different devices including WAP enabled devices, personalization, customization support for push services and provision for multilinguality at a later stage. An open XML document structure would aim to support data exchange among different public sector agencies and consumers. The success of the prototype would be judged by the consumers and efficient and effective access to agricultural information would be the critical success factor.

Ensuring efficient knowledge distribution requires a mechanism of unified description of knowledge sources, and using a common vocabulary. Semantic web principles in general and ontologies in particular are the technologies that can best be utilized for knowledge/information distribution. Ontology is defined as an "explicit specification of a conceptualization". Ontologies provide formal semantic representation that describes a particular, real world domain[66].

The IDS that would flourish and remain successful in the future are the ones that would accommodate the changing nature of knowledge and content management. It should be able to deliver multiple types of contents: prose-narrative, graphical
illustration, summary, video representation, audio, hyper links etc. Varying methods of content delivery appeals to users and they would be stimulated to absorb/retain/understand information more easily. The flexibility and extensibility characteristics of XML would facilitate its maintenance and adaptation to potential new requirements. Knowledge components that are XML enabled can be readily adapted to meet current and future standards.

4.1 Research Methodology

In computer science research, generally man made systems are studied rather than natural systems as is the case with other fields of fundamental research. After a study of various research methods [67], it was found that the best research methodology to meet out the objectives will be an approach of constructive methodology (a system would be developed) and empirical methodology (pertains to a practical utility).

The sources of research method were both primary as well as secondary. Primary data was collected from farmers of different regions of Himachal Pradesh and some farmers from other states (i.e., Punjab, Haryana, Uttar Pradesh) by interview and questionnaire methods. Data was collected from the farmers regarding their priorities from an information dissemination system developed for them. Fig. 4.1 shows the block diagram of the research structure followed.

Fig. 4.1 Block diagram of the research structure
4.1.1 Data collection

A survey was carried out to determine the information needs of the farmers of Himachal Pradesh, Punjab, Haryana and Uttarkhand (50 each). Most of the information was collected personally as mushroom growers used to visit the Directorate of Mushroom Research (Solan). Some were sent questionnaires by post while some were contacted through email.

Summary cards[68] were filled to begin the iterative modeling process. These cards are based on the CRC (Class-Responsibilities-Collaboration) Cards, an informal class modeling technique. These cards helped in collecting the input from various sources and transforming it into an object through an abstraction process. Around 300 summary cards were prepared from the data gathered. From the cards, it was fairly straightforward to define class descriptions.

| ID No. .......... |
| Name .................. Father’s Name.......................... |
| State.................. District.............................. Village........ |

Category(Scale). SS/MS/LS ............... Annual Production ............... 
Mushrooms grown 1 .................. 2 .................. 3 .................. 
Processing. Yes/No 
If Yes, Products. 1 .................. 2 .................. 3 ..................
Marketing... Local/Other States/Export ...........................

**Ranking of problems for which advice sought**

1. 
2. 
3. 

*Fig. 4.2 Summary Card*
The information (in the order of priority) as desired by the farmers is shown in Table 4.1:

**Table 4.1 Information desired by farmers in the order of preference**

<table>
<thead>
<tr>
<th>S.No</th>
<th>Information desired</th>
<th>Priority of information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Loan and subsidy schemes of the Government</td>
<td>53 83 64</td>
</tr>
<tr>
<td>2.</td>
<td>Marketing of agricultural products</td>
<td>29 113 58</td>
</tr>
<tr>
<td>3.</td>
<td>Post harvest management and value addition</td>
<td>26 97 77</td>
</tr>
<tr>
<td>4.</td>
<td>Cultivation of export oriented and cash crops</td>
<td>23 86 91</td>
</tr>
<tr>
<td>5.</td>
<td>Integrated Pest Management</td>
<td>21 73 106</td>
</tr>
<tr>
<td>6.</td>
<td>Irrigation and Rain water harvesting</td>
<td>18 137 45</td>
</tr>
<tr>
<td>7.</td>
<td>Latest agricultural equipments</td>
<td>13 71 116</td>
</tr>
<tr>
<td>8.</td>
<td>Seed production</td>
<td>10 24 166</td>
</tr>
<tr>
<td>9.</td>
<td>Self employment scheme</td>
<td>7 15 178</td>
</tr>
</tbody>
</table>

The Table shows that in the present situation, farmers are well aware of the standards expected by the consumers at national and international level.

**4.1.2 Identification of MushML elements**

The proposed Mushroom Markup Language (MushML) addresses the common structure, presentation and interoperability challenges stated in Chapter 1. The components of the proposed document structure were determined for the description of various activities related to mushroom growing. The steps that were followed towards the specification of MushML data elements are:

1. An analysis of the information needs of the farmers was performed.
2. A study of existing governmental portals and Web sites at national and international level was conducted and best practices of online e-Government portals were taken into account. The investigation focused on structure and presentation of the public sector content, from the users’ point of view.
3. Public services that were provided not only online but also non electronically by various public organizations were investigated. Subsequently, the necessary elements needed to describe the investigated agricultural information and services were identified.

4. Mushroom experts were consulted on the MushML data elements and the proposed refinements were noted and evaluated. Thereafter a new version of the MushML data elements was produced.

5. The final MushML data elements were reported according to ISO/IEC 11179-3 standard

The survey revealed that the farmers desire information on virtually all aspects of farming, right from seed production to marketing. The educated ones are convinced that the globalization and liberalization of agricultural methods and increased process of agri-business and commercialized farming and climate change has made information not merely useful, but necessary to remain competitive. They are willing to pay for reliable and timely information just as they are paying electricity and water bills. At present, the farmers depend on trickling down of decision inputs from conventional sources which are slow and unreliable.

The study requires that the information related to different agricultural processes should be standardized. The information should have a common structure for describing different types of information which would foster interoperability with agencies like the seed companies, agricultural implements manufacturers, agri-marketing firms, contract farmers etc., which are in operation because of agriculture. Users should be able to navigate easily through different services offered by these agencies. The more structured the information is, the more easily the users would be able to understand it and use it for making knowledgeable decisions. Once in use, the beneficiaries should have an opportunity to access agricultural information through multiple access channels. The flexibility and extensibility characteristics of XML would facilitate its maintenance and adaptation to potential new requirements.
4.1.3 E-R Representation

A simple E-R representation [69] of the proposed disease treatment database is shown in Fig.4.3, which shows the complete hierarchical tree of the structure described by the XSD (XML Schema Definition). Altova XMLSpy notation was used as a validating XML parser for constraint checking. The algorithm generation process was started from the original E-R model because it is flexible and general enough to be translated into any database system.

![Hierarchical structure of XML schema related to E-R schema for pests and disease management](image)

Using the algorithm, automatic XML schema can be generated for the database scheme expressed with the E-R model and the relational model. This algorithm can be further used for automatically generating a portion of the database extracted from a global or distributed information source, to be hosted on portable devices.
4.2 Prototype construction

XML data can be stored in a file system, a relational database, an object-oriented database, or a native-xml database system. All these approaches have their own advantages and disadvantages as mentioned in Chapter 2.

After a detailed study of the features of all these approaches, it was decided to use Relational Database to store and manage XML data for our prototype. This enabled us to take the benefits of the advantages offered by this approach. Some of these are efficient storage, indexes, security, transactions and data integrity, multi-user access, triggers, queries across multiple documents, and so on. Even when using a relational database management system (RDBMS), there are many different approaches to store XML data. One strategy is to ask the user or a system administrator to decide how XML elements are stored in relational tables. Another option is to infer from the DTD of the XML documents how the XML elements should be mapped into tables. Yet another option is to analyze the XML data and the expected query workload. Many researchers are trying to store XML data into relational database using different methods.

Mapping plays a very essential role in the storage of XML data into relational database (RDB). Mappings can be classified in different ways. According to one of the classification there are three types of mapping: schema mapping, data mapping and query mapping. After taking into consideration the advantages and disadvantages of these, schema mapping was followed in the present study.

4.2.1 Candidate Example of mushroom

A prototype has been proposed and developed for mushroom farming as it is a complex process involving many sub processes like compost preparation, spawn production, crop management, crop protection, harvesting, processing and marketing. In addition to these activities involved, farm design plays a very important role in mushroom production as the environmental parameters of the mushroom house (temperature, humidity, carbon dioxide content) have to be kept under specific range depending upon the mushroom species being grown. All these activities make mushroom growing a highly technical enterprise which requires expert technical guidance[70],[71],[72]. The importance of mushroom as a crop has been described in Chapter 1. The prototype can be scaled up to include other agricultural commodities.
The information needs of the mushrooms growers of four states (Himachal Pradesh, Punjab, Haryana and Uttarkhand) were determined and 6 data dictionaries were compiled. These yielded approximately 2000 data components (elements, attributes and types). Through study and analysis of the similarities and differences among these components, approximately 250 properties (xsd:element) and 40 type definitions (xsd:type) were synthesized to represent a common set of elements and types. The model tries to capture the requirements of the 6 data sources as completely and accurately as possible.
4.2.2 Software requirements

The prototype development and implementation is based on the Microsoft Windows 2003 operating system, MySQL, XMLSpy, the Java 2 Enterprise Edition integrated platform and the Apache-Tomcat web server. The Microsoft Access simulates the database management system. Various XML Editors such as "XMLSpy," "XMLMind" and Web browsers, such as the Internet Explorer and Netscape Navigator were tested and used to develop and test the system. Java platform, Apache web server and Netscape are freeware and open-source products making them easily customized to the specific needs of every application. Almost all of them (except Microsoft Access) have adequate capabilities to support general requirements to manipulate, distribute and store XML documents [73].

XML is non proprietary with no copyrights, patents, trade secrets or Intellectual Property restrictions involved. Domain specific languages developed by extending XML enable users to exchange information within and outside their fields.

The Apache-Tomcat web server has a suitable overall performance that has made it one of the most popular and most reliable products in the highly demanding business world [81]. Its open-source base and its efficiency in almost all the operating systems make it a good solution. The Tomcat server is a Java based Web Application container that was created to run Servlets and JavaServer Pages (JSP) in Web applications. As part of Apache's open source Jakarta project, it has nearly become the industry accepted standard reference implementation for both the Servlets and JSP API [74].

Furthermore, because Apache-Tomcat is coded and based in Java, any application using the same programming language creates a better combination with even better performance. Java acquires a large number of capabilities and advantages that can expand and can improve the efficiency of the implemented application. Due to the object oriented structure of Java, any new features or improvements can be easily attached in the future, keeping the entire solution always updated.

XML spy is an advanced XML editor for modeling, editing, transforming, and debugging XML-related technologies. The XML editor is used to create advanced
XML and Web applications. It is flexible and allows users to work with any XML technology in a way that best suits the complexity of the document and the developer's preferences, for instance, if he/she prefers to develop in a text view or graphical view, or switch back and forth between the two. Using technologies like XML linking or the Resource Description Framework (RDF) would also provide a dynamic and flexible mechanism to relate documents to external information sources and address some of the semantic interoperability challenges of the future.

4.2.3 Server and User (client) side architectures

When the user accesses an XML document, via a Web browser, an XML Editor is used to handle the document itself. This system has enormous flexibility and interoperability because the user's choice of editor has no restrictions and can be either a commercial off-the-self product or an open source. For content creating tasks, the editor will parse and check each document to determine if it is valid. The same is also done on the server side before data is accepted. The Web Service can be written in any of the common languages (e.g. Java, C++, Visual Basic) by utilizing any of the existing technologies, such as the Microsoft's "NET" framework, or the Sun's Java WSDP (Web Service Development Package). When there is a need for a system, which is expandable and interoperable, using technologies that are open source is preferable so that they can be tested and their security can be verified either by using formal or non-formal methods. For that reason, Sun's Java WSDP has an advantage.
over other technologies and has already been used by other major vendors (IBM, Oracle) as the base for their applications or development tools [65], [75].

We have utilized Java for the purpose. Fig. 4.6 and Fig. show the server side and user/client side architectures of the prototype.

![Browser + XML Editor](Connected to the server)

**Fig.4.6 User/client side architecture**

Information to be disseminated was taken as desired by the farmers (their needs were identified by a survey conducted in four states). The entire information needs of farmers was grouped under six thematic heads, which are:

1. About govt. schemes (loans/subsidies by different govt. agencies)
   a. Loans and subsidies by different govt. agencies
   b. Facilities to different categories (SC/ST, women etc.)
   c. How to avail these facilities

2. Farm Design and Machinery
   a. Special farms (indoor) are required for growing mushrooms
   b. Technical details and pictures of machinery would be displayed on request.

3. Mushroom growing (mushroom types, substrate/raw material used, temp. requirements)
   a. 5 types of mushrooms
   b. Substrate/raw material requirements
   c. Growing technology
4. Integrated Pest and Disease Mgmt. (diseases, pests, precautions against them & cure)
   a. 3 mushrooms, 6 diseases and disorders
   b. 4 types of insects, their damage and cure
5. Processing and Marketing
   a. General guidance regarding harvesting and storage
   b. Different products of mushrooms
   c. Marketing options including export
6. Publications
   a. Books and journals
   b. links to relevant sources of information

The realization of the MushML is based on the RDF [76] specification, which provides a lightweight ontology system to support the exchange of knowledge on the Web. The RDF specification has been expressed in the MushML vocabulary in order to model metadata about agricultural sector resources on the Web. The vocabulary provides the content descriptors with concern to the activities metaphor. Content descriptors may contain: Title of the document, URI of the document, the node where the document is located, type of the document, name of activity that the document is associated with, language of the document, keywords.

All activities/services have been characterized in an identical way in order to facilitate their retrieval from the activities/service repository as well as their execution in the runtime environment. The vocabulary of activities/service attributes is regarded as part of the MushML role inside the whole project. Thus, this service vocabulary has been defined and described in terms of MushML specification.

The prototype is based on the three-tier architecture, which was discussed in the previous chapter. The prototype implements a web service provided by the latest version of the Apache-Tomcat web server. It represents the middle-tier in the three-tier architecture that can be integrated with various other commercial products. The Database simulates the third-tier, while the first-tier can be any of the commonly used
Web browsers. The prototype is designed to process everything in the server side and to dynamically create every response to the user.

The database is used to store document data, document metadata and user security data and to provide the requested resources to the service, which is the only software component that makes the security decisions according to the policy [77]. The DB is also relied upon to keep separate the different levels of data. For simplicity, the database uses the “document storage” technique that was analyzed in the previous chapter. From the security point of view, it is better to have the web and the database server running on different machines but in the same physical location in order to reduce potential malicious actions during their communication. If for any reason the web server and the database server must be separated, besides the known technologies for secure communication (such as SSL), a modified cryptographic integrity lock can be implemented.

The selected components (e.g. operating system, programming language, web server, database, etc) are those commonly used nowadays [65]. Most of them are also open source or free software, which was the only reason they have been selected and not for their security features. The organization implementing the prototype, depending on its specifications and its requirements must choose the appropriate high assurance operating system that would support the web and database servers. Furthermore, all of today’s architectures for secure authentication and authorization (e.g. PKI, Kerberos, X509) as well as those for secure transmission (e.g. encryption, SSL) between the user and the server can be easily integrated and used because the Java programming language and the XML supports all of them.

4.2.4 Security Policies

As it is already mentioned, all the security decisions are made by the server, which is the only software component that enforces the security policy of the organization. The security manager of the prototype implements an Access Control (AC) policy using security objects that are dynamically generated from XML metadata security tags. For example, even though the information related to a user or a file are retrieved from the
database, the security objects and their permissions are generated in the server according to the enforced security policy. Those objects are related to the file’s (or user’s) metadata tags, and are kept “alive” only during the current user’s session and if for any reason the session is invalidated the objects are destroyed.

Moreover, there is no way for a user, to directly communicate with the back-end database and request any resources because the database server accepts requests only from the assigned web server. Depending on the specific products, their physical location and their configuration, an authentication between the web server and the database server can be done once or in every request.

The security policy of the prototype system allows the authenticated user to open or to save documents at the level logged on and below [78]. However, every time a user saves a file to the system, a user’s object and the date and time are attached to the file, making that file unique. The main purpose of this operation is to make a clear distinction from the various versions of the same file, saved in the system by the same or other users. Only specifically authorized users may change the classification of documents. Concerning the communication between two servers, when a user requests a file that is located in an associated server, there are two different approaches depending on where the final decision is made. In the first approach, the requesting server, after its successful authentication, sends the user’s object and the associated server makes the decision if the requested file is releasable. In the second approach, after the successful authentication of the requesting server, the associated server sends all the file names and their security objects to the requesting server, which will decide which of them are releasable to the specific user. In the prototype implementation, the second approach is used because the intention is to share the documents and not the user’s information. Furthermore it provides a better flexibility because each server can enforce differently its own policy to its own users.
4.3 Mushroom growing operations ontology and semantics

Most of the already developed XML based languages to transport and/or describe data in agriculture are related to a specific domain and cover the data exchange in the supply chain, mostly, from the farmer to the consumer. According to [39], one of the most important challenges of a new standard XML based language, as agroXML, is to allow new expansible structures, country specific languages and local legislation accomplishment [23], [30]. Based on the concept of re-utilization and standardization, the most logical choice is to find the best structured XML based language and expand this language into a specific new domain instead of developing a new language from scratch. However, these modifications should not disturb the compatibility with the original language, otherwise a new XML based language would be created which may not be desirable. Analyzing all present XML based languages [21], [22], [23], [30], [36] and the above restrictions, the most mature candidate for a unique standard in agriculture is the agroXML, so it was chosen to be the base for developing MushML (Mushroom Markup Language). The figures 4.1 and 4.2 will show the agroXML and MushML structures in detail. Before defining the agroXML elements to be expanded and the lists to be added to turn the agroXML into the MushML, it is necessary to define the mushroom growing operations ontology and semantics.

According to the Dictionary.com website [79] ontology may be defined as: “A systematic account of existence”. According to philosophy, ontology is defined as “An explicit formal specification of how to represent the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them”. In this case, the area of interest is the mushroom growing operations domain.

According to the Merriam-Webster.com website [80] semantics means: “The study of meaning”. Another definition is “The study of linguistic development by classifying and examining changes in meaning and form”. Specific and significant names must be chosen for each operation and resource not already described in agroXML. The ontology study implies on defining the type and hierarchic precedence of each mushroom growing operation in order to assure valid sequences of operations.
and appropriate utilization of resources [81], [82], [83]. (Example: ruffling cannot be associated with spawning).

![Fig. 4.7 Categories of mushroom operations](image)

Possible and allowed associations between different operations according to the mushroom growing ontology are shown in (figure 4.6). This information will be utilized in the MushML schema to describe the hierarchy of operations and resources. The semantic study implies on defining names for the operations and resources that are unique and by preference self-explanatory. Table 4.5 presents a list of mushroom growing operation names and respective descriptions. These names will be utilized on the MushML lists and it can be translated into the local language (at a later stage) and updated as necessary. Presently, all names and descriptions in this research work are in English.
After that, all definitions in the included file will be part of the present schema. This type of approach allows the XML schema add complexity to a main file without repeating the same structure definitions [76]. Each included definition could be reused by other schemas. The solution to add new definitions for mushroom growing operations and maintain the compatibility is to include a new schema file that will describe the mushroom specific definitions. The expanded schema is in the file "MushroomGrowProcess.xsd" and begins with the definition of a complex type named "MushroomOperationsType" and it is derived from an abstract type named "AbstractWorkProcedureType" that defines the basis for an agriculture operation inside agroXML. The "MushroomOperationsType", in turn, defines five different categories of allowed elements: MushSpawningType, MushCompostingType, MushCrop_mgmtType, MushHarProMarkeType and MushGeneralType. (Operations relating to Harvesting, Processing and Marketing have been clubbed under category “HarProNet” and those related to government schemes, farm & machinery and external links have been clubbed under category “General”.

Table shows the mushroom growing operations which have been grouped into five categories as described above.

<table>
<thead>
<tr>
<th>Spawning</th>
<th>Composting</th>
<th>Crop_mgmt</th>
<th>HarProMarket</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoclave</td>
<td>Pasteurization:Tunnel</td>
<td>Filling</td>
<td>MushHarvesting</td>
<td>Subsidy</td>
</tr>
<tr>
<td>CulturePrep</td>
<td>CompostingYard</td>
<td>Spawning</td>
<td>MushWashing</td>
<td>Support</td>
</tr>
<tr>
<td>MediaPrep</td>
<td>CasingChamber</td>
<td>SpawnRun</td>
<td>MushPacking</td>
<td>StateHortDept</td>
</tr>
<tr>
<td>SpawnStorage</td>
<td>Boiler</td>
<td>Casing</td>
<td>MushCanning</td>
<td>MarketSite</td>
</tr>
<tr>
<td>Transport</td>
<td>CompostTurning</td>
<td>CaseRun</td>
<td>MushPickling</td>
<td>Training</td>
</tr>
<tr>
<td>Precautions</td>
<td>CompostFormulation</td>
<td>Ruffling</td>
<td>MushSoup</td>
<td>Turner</td>
</tr>
<tr>
<td>Supplements</td>
<td>Spraying</td>
<td>MushBiscuit</td>
<td></td>
<td>FrontLoader</td>
</tr>
<tr>
<td>CacingPasteurization</td>
<td>Pests</td>
<td>MushNugget</td>
<td>PowerTiller</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Diseases</td>
<td>LocalMarket</td>
<td>WaterSprayer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CookOut</td>
<td>SupplyChain</td>
<td>Harvester</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HotelSale</td>
<td>MushDrier</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MushExport</td>
<td>ColdRoom</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MushTransport</td>
<td>Thermometer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Hygrometer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Humidifier</td>
</tr>
</tbody>
</table>
Table 4.3 shows the mushroom growing operations along with their names, category and a brief description of the operations name.

Table 4.3 Mushroom growing operations name, category and description

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Short Name</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Spawning</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Autoclave</td>
<td>Instrument for heating at a particular temp./pressure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Culture</td>
<td>Vegetative mycelium of mushroom grown on a medium</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Media</td>
<td>Media (PDA etc) for growing culture</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Storage</td>
<td>Preservation of spawn</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transport</td>
<td>Transportation to other places from the spawn lab precautions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Storage and transportation precautions</td>
<td></td>
</tr>
<tr>
<td><strong>Composting</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>past_tunnel</td>
<td>Pasteurization of compost/casing soil</td>
<td></td>
</tr>
<tr>
<td></td>
<td>comp_yard</td>
<td>For making compost</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cas_chamber</td>
<td>Used for preparing casing soil</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Boiler</td>
<td>Raising the temperature and killing bacteria</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c_turning</td>
<td>Turning of compost pile</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c_formulation</td>
<td>Different formulations of compost</td>
<td></td>
</tr>
<tr>
<td></td>
<td>supplements</td>
<td>Supplements added to the compost</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c_pasteur</td>
<td>Pasteurization of casing soil</td>
<td></td>
</tr>
<tr>
<td><strong>Crop-mgmt</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Filling</td>
<td>Filling of bags with compost</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Spawning</td>
<td>Mixing spawn with compost</td>
<td></td>
</tr>
<tr>
<td></td>
<td>spawn_run</td>
<td>Spreading of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Casing</td>
<td>Putting a casing layer over the surface of the bag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>case_run</td>
<td>Spread of mycelium over the casing layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ruffling</td>
<td>Breaking up of casing layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Spraying</td>
<td>Spraying of water for maintaining humidity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pests</td>
<td>Pest attack during growing process</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Diseases</td>
<td>Diseases occuring during growing process</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cook_out</td>
<td>Killing of bacteria after cropping cycle is over</td>
<td></td>
</tr>
<tr>
<td><strong>HarProMarket</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>harvesting</td>
<td>Picking of mushrooms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Washing</td>
<td>Washing process of mushrooms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Packing</td>
<td>Packing of mushrooms into bags/trays</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Canning</td>
<td>Preservation in tins for longer time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pickling</td>
<td>Process of making mushroom pickle</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Soup</td>
<td>Making soup powder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Biscuit</td>
<td>Preparation of mushroom biscuits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nugget</td>
<td>Preparation of mushroom nuggets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>local_market</td>
<td>Local selling of mushrooms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s_chain</td>
<td>Selling through a group</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hotel_sale</td>
<td>Understanding with the hotel industry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Export</td>
<td>Selling to other countries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transport</td>
<td>Transportation of mushrooms to other cities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>General</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subsidy</td>
<td>Various concessions (financial) given by the government</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Support</td>
<td>Other facilities given by the government</td>
<td></td>
<td></td>
</tr>
<tr>
<td>state_hort</td>
<td>Links to horticultural websites of different organizations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>market_site</td>
<td>Access to websites related to marketing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training</td>
<td>Links to various training centres</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turner</td>
<td>Gives turning to the compost pile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f_loader</td>
<td>Handling mixing, turning operations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p_tiller</td>
<td>Transport raw material and compost</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sprayer</td>
<td>Sprays water on mushroom bags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harvester</td>
<td>Harvests the mushroom crop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drier</td>
<td>Used for drying mushrooms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cold_room</td>
<td>For storing spawn and harvested mushrooms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thermometer</td>
<td>Measures the temperature of cropping room</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hygrometer</td>
<td>Measures the humidity of cropping room</td>
<td></td>
<td></td>
</tr>
<tr>
<td>humidifier</td>
<td>Used for increasing humidity in the cropping room</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each category, then, will define the allowed operations \( T_{\text{action}} \) and then the elements of each operation \( T_{\text{elem}} \). All other information about farm identification, field identification, crop types, producers, machinery work, personal work, and other standards will be treated by the already existent agroXML structures when the system is upgraded to include other agricultural commodities.

4.3.1 Resource lists

The agroXML standard allows the creation of resource lists \([36], [42]\). Resource list is a group of resources that could be machinery names and types, fertilizer formulas and concentrations or other supply product description that could be referred inside
the agroXML. To support mushroom growing operations resource lists specific to the
country and region could be involved. As these resources are different from location
to location it, is not feasible to provide standard names and technical definitions for
one country that are not used in another one.

```xml
<xsd:schema xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns="http://www.agroxml.de/schema/agroxmll.4"
attributeFormDefault="unqualified"
elementFormDefault="qualified"
targetNamespace="http://www.agroxml.de/schema/agroxmll.4"
version="1.4"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:include schemaLocation="./CommonBasicComponents.xsd" />
<xsd:include schemaLocation="./WorkProcess.xsd" />

<xsd:complexType name="MushroomOperationType">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Parent abstract operation for mushroom.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent mixed="false">
    <xsd:extension base="AbstractWorkProcedureType">
      <xsd:choice>
        <xsd:element minOccurs="0" ref="MushSpawningType" />
        <xsd:element minOccurs="0" ref="MushcompostingType" />
        <xsd:element minOccurs="0" ref="MushCrop_mgmtType" />
        <xsd:element minOccurs="0" ref="MushHarProMarketType" />
        <xsd:element minOccurs="0" ref="MushGeneralType" />
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="MushroomOperation" substitutionGroup="_WorkProcedure"
  type="MushroomOperationType">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Abstract element that represents the group of mushroom
    operations.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

Figure 4.9 Begin of the schema document for the MushXML (MushroomGrowProcess.xsd).

77
Figure 4.10 – Schema for definition of the spawning category.
4.4 Structure of the Prototype

The prototype is mainly based on the object-oriented fundamentals, which are perfectly supported by the Java programming language and the XML features. The user of the system and its security attributes are represented as an object. Every file selected by the user either for opening or for storing is parsed and an associated security metadata labels object is created. The system compares those two objects and according to the security policy, determines the response to the user’s request. The files’ related objects, the user’s objects and the processing software are all kept and are all managed in the server area, which is assumed to be a secure environment. The interface with the user is achieved though web pages that are generated dynamically according to every situation.

As mentioned earlier, the XML file’s security attributes are mapped to a Java object [77], [78]. Due to the structure of XML, it is also possible to map every paragraph of the document to a Java object and to manipulate each paragraph differently. Even though this technique can be implemented easily from a technical point of view, there would be problems with the logic of the document itself. Accessing the paragraphs of a document that are not allowed due to their security attributes could make the document non-contiguous and potentially difficult to understand. The only exception is when the author of the document has the special training required to write those multi-level documents, which must be understandable, persistent and without revealing sensitive information by inference. This is an issue that the Community Metadata Working Group has already noted in their various reports and conferences [77].
4.4.1 Description of System’s Logic-Flow Diagrams

The first page presented to the user by the server is a simple html file, which briefly describes the application and the necessary links to advance to the other pages. One of the links is “Help,” which opens another html file with general directions about the use of the application and some general queries of the users one might encounter.

Another link, in the main page is “User Login,” which opens a simple html file with a form waiting to receive the user’s login name and password. The inputs of the login page are sent to the first java servlet, called *Login.class*, which has the responsibility to authenticate the user according to the user name and password received.

The *Login.class* initiates itself and attempts a connection with the third—tier database server. When the connection is established, this class queries the database using the user’s credentials. The response from the database is processed inside this class. In case those credentials are not valid, an error HTML page is displayed with a respective message and the appropriate options to continue. If the user’s credentials are valid according to the database, a number of queries are sent again to the database server to retrieve the authorizations of the user and those files are then available according to those authorizations. A *UserBean* object is created and attached to the new session. This object will be used during the whole procedure.

---

**Figure 4.12 Login Flow Diagram**

![Login Flow Diagram](image-url)
After the authentication and authorization of the user, the control is passed to the `UserOptions.class` that generates a web page with some of the user's personal data, such as first name, last name, and other available options. These options are the main menu and they are presented in a drop down list. When one of them is selected, the control is passed to the `UserSelection.class`. The `UserSelection.class` does not generate any web pages. It is only responsible for redirecting the control to the next respective servlet, according to the user's selection.

![Figure 4.13 Main Menu Flow Diagram](image)

As can be seen in Figure 4.13, the user is presented with four choices. The first choice, named "Save to Database," is used when the user has created or copied a file inside his own machine and wants to send it to the server. The first servlet called in that option is `ImportToSave.class`, which generates a web page waiting for a filename to be sent to the server. The selection of the filename can also be done through the "browse" button that will initiate the "open file" window, provided by the operating system.

Choosing from the local file system will be more convenient than writing the name of the file. It will increase the abstraction level of the user. On this web page, some of the user's data and some other administrative information that will be appended on the selected file are presented. The user cannot change any of his or her data associated with the file, because this is done on the server side, and their presentation on the page cannot affect their processing on the server, because the
system does not accept those kind of information from the user. The user can only change the presented administrative information in order to characterize the file. Furthermore, the user can select the classification of the file, which according to the enforced policy (prohibits "write-up" or "blind-write") could not be higher than the user’s present classification. The presented classification choices are only those accepted from the logic of the system (e.g. "write-down"). Finally, the page displays the date and the time that will be appended to the file in order to be used later as the key reference for the version of the file.

When the user inputs the filename and selects the “upload to the server” button, the SaveFile.class is called. The first task of this class is to initiate the upload procedure. During the uploading, a temporary file position is created in the web server area in order to accommodate the file. After a successful uploading, the server parses the file in order to complete a number of required checks. The first of them is to identify that the file is an XML file and that it is well formed and valid according to the Metadata Standard’s. This task is achieved by the Echo4.class that parses the file using one of the “SAX Parsers” provided by the “javax.xml” packet of Java. If the file is well formed and valid, a FileTags.class object containing all of its security attributes is created for that specific file. The system retains the FileTags object only during the required time for processing inside the server or until the end of user’s session.

During the uploading and parsing process, a number of messages are created and presented to the user in order to give him/her a view of the procedure. If the system encounters any problem or errors, during the above process, an appropriate message, indicating the cause is generated and displayed. If the process is completed successfully, the newly created FileTags object for that specific file as well as the UserBean object are sent to the SecMetadataManager.class. This class acts as the security manager. The important task that the SecMetadataManager.class performs is to compare the two objects and to generate the decision regarding that specific file and that specific user. When the user requests to store the file with a different classification than the XML security attributes contained in the file itself, an appropriate message is generated. This operation helps to prevent unintentional change of the file’s classification, made by user’s mistake or malicious code, to downgrade a document for disclosure to lower classification users. For example,
when a user currently logged on as “SECRET,” requests to save a file as “UNCLASSIFIED,” but the XML file contains security classification “CLASSIFIED,” then the system generates a “downgrade” message to warn of the difference. If the user confirms the change of the file from “CLASSIFIED” to “UNCLASSIFIED,” then the system downgrades the file and saves it in the respective area with the changed security attributes. According to the policy, there is no case for a user to have the capability to request the system to save a file with a classification higher than the classification for which the user is currently logged on. The SecMetadataManager.class is the one and only security enforcement module, which is always invoked for security decisions when a file is saved to the system.

![Save to Database Flow Diagram](image)

**Figure 4.14** Save to Database Flow Diagram

Viewing the menu, the user may select the second choice named “Open from Database.” It is used when the user wants to retrieve a file from those stored in the system’s database. The first class is called OpenFile.class. This class opens a connection with the database and retrieves all the files that are available and can be released to the specific user according to his or her security attributes. User can also input filename criteria so that the system can search among the releasable files. This operation is performed by generating queries to the database, according to user’s security attributes and the enforced security policy. The OpenFile.class is the security enforcement module, which is always invoked for security decisions when a file is retrieved from the system. The name and the version (simulated by the date and the time the file entered the database) are presented to the user who must select one of the files to open. If a user has a higher classification, the files are separated and displayed in categories according to their classification levels.
The user selects a file and its respective filename is sent to the same OpenFile.class, which retrieves the file from the database, creates a fileTag.class object for the file, and then gives the user two opening options. The first one is to send the file to the user's workstation to directly open the file, where the underlying operating system will activate the default XML editor or where it will save the incoming file to a local directory. As it is mentioned in the beginning of this chapter, the user's operating system is assumed as untrusted. The other opening choice is to send the file to the user's workstation to be processed by an applet simulating a distributed XML editor residing in the server. The first choice gives the user more flexibility but requires a powerful XML Editor installed in the user's machine and a well maintained operating system (regarding the security). On the other hand, the applet-based XML Editor might restrict the user's capabilities but could also provide better security features due to the reduced user interaction. Furthermore, in an applet-based editor, many customized options, specific to the organization can be added or removed at the same time for all the users.
Coming back to the user's main menu, the third choice, "Open from External Database," is used when the user wants to obtain a file that is kept and controlled by another server. It is assumed that the user's server is already associated with other servers, running the same or similar implementation, and they have agreed to share their XML documents. Obviously, the number and the kind of the associated servers, depends on the policy the organization currently uses, which can be modified at any time restricting or allowing access. The screen shorts of some of the menus are shown in annexure V.

If the user selects the third option, the first class called is ImportFile.class, and this in turn generates a web page on which the user must select the affiliated enclave or organization whose server is associated with the system. There is also a field where the user can input the complete filename or any portion of it so that the system can search among the releasable files. When the submit button is selected, the ImportFile.class initiates a connection with the respective server by calling the Login.class of the associated server. After a successful connection, an authentication and authorization occurs, and when they are completed successfully the releasable filenames and their security classifications are returned. The user's server has the responsibility to determine which files the specific user is allowed to obtain. Another possible approach could be to send the user's object to the associated server which will be responsible for making the decision. This approach is not recommended.
because it could reveal many information (directly or indirectly) about the users related to a server. The intention is to share only the documents and not the users’ information. The ImportFile.class compares the classifications of each file with the user’s classification and keeps only those files that are releasable to the user. If the user has inserted a filename or any word previously, the system searches among the file names to find those that match. Finally, a web page is generated and the results are displayed through the OpenFile.class, which now assumes control and handles the file similarly to the previous choice.

The last option in the user’s main menu is “Logout,” which closes all the connections, deletes any temporary files and kills every object created during the user’s session. The respective class is named Logout.class, and after a successful execution it generates a web page displaying a greeting message and links to the home page. As a future work, many other options can be added to the main menu extending the performance of the implementation. One of those options could be the capability, for specifically authorized users, to save a document in an external database.

4.4.2 Analysis of the Java Servlets

The prototype is implemented using Java servlets running on the latest version of Apache—Tomcat. Their objectives, an analysis of their functionality, and their main methods are presented below:

a. Login.class

The main objective of the Login.class servlet is to authenticate the user and to retrieve his or her authorizations. Login.class is the first servlet called after the user inputs his or her credentials to an HTML form. The Login.class begins by initializing the ServletConfig.class, which is its super class and thus inherits many properties. The next step is to open the connection with the third-tier database calling the JdbcOdbcDriver.class provided by the Sun Microsystems. The connection object created is attached to the session. The Login.class can accept either a GET or a POST request from the html file.

The first method called is validateUser, which makes the query to the database to determine if the username and password entered are contained in the database. When the database determines that the user’s credentials are valid, a number of other
methods are called in order to retrieve all the user’s information. The user’s personal and security information and any authorized files are all packed in a UserBean.class object. Then the startSession method is called. This instantiates a RequestDispatcher object, initiates a new Session object by attaching the newly created UserBean object, and transfers the control to the next servlet. In case of any error or if any exception is thrown, a respective error message is passed to the exitPoint method. This generates a dynamic HTML page displaying the cause of the problem.

b. UserOptions.class

After successfully authenticating and authorizing the user, the Login.class passes the control to the UserOptions.class, which creates a page presenting the user’s information and the available options. The first method called is printData that obtains the personal and security data from the UserBean object and displays them to the user. The next method called is printOptions that creates a drop-down menu displaying the available options to the user. There is also a link in case the user wants to return to the previous page.

c. UserSelection.class

The user’s selection is submitted via a POST request to the next servlet, the UserSelection.class. This servlet’s only responsibility is to obtain the selected option and redirect the control to the appropriate servlet. The doPost method, using a number of “if” statements, determines what the selected option is and then calls the startSession method that dispatches a RequestDispatcher object to the respective servlet.

d. ImportToSave.class

The objective of the ImportToSave.class is to create the necessary page so that the user can input some required information about the file and file’s name. The first method called is the printStandardMetadata that draws a table and displays the user’s data that will be appended to the file. The user cannot change this data. However, by calling the dropDownClass method, the user can change the file’s classification, provided that the user, according to the policy, has an equal or higher classification. The dropDownClass method displays and permits him or her to choose only those allowed. Adding more methods like the dropDownClass, an implementing
organization can allow its users to select more options according to its policy. After the `printStandardMetadata` method, the control is transferred to the `inputFile` method that creates the text field used for inserting the name of the file. This field is created by the “`INPUT TYPE='file'`” HTML tag that also provides a browse capability to the user. The necessary submit button, labeled “Upload to the server” is also displayed as well as a “Reset” button. Finally, the `exitLinks` method is called displaying the links to cancel the session and to login as another user or to go back to the main menu page and select another option.

e. `SaveFile.class`

The name of the file selected by the user is passed to the next servlet, `SaveFile.class` that is responsible for retrieving the file from the user and after a few checks to save it in the server. The method `uploadFile` is called within a “try-catch” block in order to be ready to catch any I/O exception that might be thrown by the process. A temporary directory and a temporary file are created in a specific server area to accommodate the uploaded file. Due to the selected way of uploading the file (through the “`INPUT TYPE='file'`” HTML tag), the rest of the passed information accompanying the file are in the beginning of the input stream. The necessary processing will distinguish and separate those parameters from the file itself. Moreover, at this point, the system will determine if the name of the file is a valid XML filename, and if the file is starting with the XML version declaration. After completing those checks successfully, the reading writing procedure starts.

After the whole file has been saved to the temporary server area without any errors or exceptions, the `checkingFile` method is called. Using the temporary name of the file, the `checkingFile` method creates an object of the `Echo.class`, which will parse the XML file. The method `parseFile` of the `Echo.class` is called obtaining the name of the file and the parsing procedure starts. The parser used by the `Echo.class` is a `SAXParser` from the `SAXParserFactory` class of the “`javax.xml.parsers`” package. While parsing the file, a `FileTags` object is created by the `parseFile` method and returned to the `checkingFile` method if no errors are encountered. Returning a “null” `FileTags` object means that the parser found validation errors in the document, so an appropriate error page is created. When the validation-parsing process is completed without errors, the `FileTags` object as well as the `UserBean` object are passed to the
SecMetadataManager.class. The main task of the SecMetadataManager.class is to compare the passed FileTags and UserBean objects and to return the decision. In this specific situation the file is checked to determine the classification in which it will be stored in the server.

The decision can be “auto-downgrade”, “downgrade,” “upgrade” or “normal.” “Auto-downgrade” means that the file was characterized by the user with a higher classification than the specific user is permitted. So the system will automatically downgrade the classification of the file due to the user’s current logged on level. This feature enforces the policy that prohibits users to write in a higher level than the level they are currently logged on and could not be used as a normal procedure. If a user wants to “downgrade” a file that is already saved in the system with a high classification, he must log on, at least in the same level as the file. The decision “Upgrade” means that the user has the authority and asks to save the file with a classification higher than the file already contained in its security attributes. It is actually only an observation of the system, which will not have any negative impact on the overall security of the system. The final possible decision is “normal,” which means that the security attributes are in agreement.

During the above procedure, the system generates appropriate messages to inform the user about the progress. Assuming that all the procedure are completed without any errors, the system displays the decision and waits for the user’s confirmation. That confirmation will trigger the transfer of the control to the next servlet.

f. UpdateDb.class

The next servlet called is UpdateDb.class, which as its name implies, will mainly update the database. First, the copyFile method is called in order to simulate the transfer and storage of the file to the database server and to an isolated secure area of the system. The next method called is the updateDatabase that updates the tables of the database containing the metadata of the file.

g. OpenFile.class

When the user selects the option “Open File” in the main menu, the OpenFile.class servlet is called. The main task of this servlet is to coordinate the opening procedure
either for a file saved in the local or in an associated server. The printAvaFiles method displays all the files that are available to the specific user. Those files are retrieved from the database through the findFiles method, which opens a connection with the database and queries the respective table containing the file’s metadata. Similar to the printAvaFiles method is the printServerAvaFiles method, which is used when the files are obtained from one of the associated servers. The available files are displayed with their name and their version, namely, the date and the time saved in the system. The name of each file is presented as a link. When the user selects a file, the same OpenFile.class is reactivated but with a different sequence of methods, which are, determined from conditional “if” statements.

The system provides two different options to the user to open the file. The first choice is by using the operating system’s incoming file options, which in the case of Microsoft Windows can be to open the file by the default XML editor or to save it to the user’s machine for future process. The default application that Windows uses to open a file can be changed at the “Folder Options” contained in the “Tools” menu in the Windows Explorer. The tab “File Types” will display the list of the file extensions and the applications to which each of them are associated. When the user has not defined an application for a specific type of file, the Internet Explorer will be used.

Another choice that the system provides is an applet-based XML editor that can be loaded on the user’s machine at the time a file is requested. This capability simulates a distributed XML editor, provided by the server in the case where the user does not have one installed locally. It is implemented by calling the MainApplet.class, which is a Java applet extending the functionality of the Japplet.class contained in the java.swing package. The name of the file to be opened is passed to the MainApplet.class applet using the “PARAM NAME=filename VALUE = The Actual Filename” html tag.

h. ImportFile.class

The ImportFile.class objective is to initiate the import procedure from one of the associated servers of the system. Its basic method is the inputUrl that creates one “radio” button for each of the affiliated servers and a text field for the name of the requested file. The system accepts an empty filename and translates as “find them all.”

90
The necessary submit and reset buttons are also created and displayed to redirect the control to the next servlet and to clear the field, respectively.

i. **FindUrl.class**

The requested filename and the associated server where the file resides are passed to the `FindUrl.class`. This will establish a connection with the server and then retrieve the file. Using multiple "if -else" statements, the `FindUrl.class` determines the progress of the procedure. In the initial stage, the `getConnected` method is called with the name of the server as an argument. The `getConnected` method using a "try-catch" block encodes the URL of the selected server and calls the `ServerLogin.class`. The `ServerLogin.class`, as the name implies, is a login class dedicated to the authentication and authorization of the calling server. Its logic is similar to the `Login.class` mentioned above, which authenticates and authorizes a user. After an established connection and a successful authentication, the `ServerLogin.class` of the associated server retrieves the available files and transmits their metadata back to the calling server.

The `FindUrl.class` receives the available files from the associated server, and assuming that a filename (or a portion of it) has been entered, disregards those files whose names do not match. From the remainder, those that do have matching filename, the system compares the user’s security attributes to those of each file. If a file has a higher security classification than the user posses at this session, then the file is removed from the list. Finally, the remaining list of files, if any, is passed to the `OpenFile.class`, which will proceed to the opening procedure as already described. The only difference now is that instead of the `printAvaFiles` method, the `printServerAvaFiles` method will be called.

j. **Logout.class**

The `Logout.class` purpose is to terminate the user’s session. Its basic method is the `processRequest` that ends the session by calling the `invalidate` method of the `HttpSession` class of the `javax.servlet.http` package.
4.5 Main features of the prototype

MushML structures take advantage of the re-usability and extensibility, which characterize XML Schema complex types. As a result MushML defines data structures which are open, flexible, extensible and easy to use and maintain.

The MushML vocabularies are implemented using XML technologies. In order to define and implement MushML documents in XML format, an XML schema was implemented for the validation of their structure. The XML Schema validation mechanism [84] was preferred over DTD because it provides a richer set of data types including data, integer etc. and allow users to derive their own data types and take advantage of inheritance of elements, attributes and definitions of data-types.

New XML documents describing new services or events can emerge from the XML Schema of MushML. Consequently, the appropriate XSL transformations (XSLT) have to be applied for transforming the MushML documents to the desired format (HTML etc.). XML Schema can be easily, extended, modified and maintained keeping in view the user’s future needs.

Example how new information is added and displayed

Supposing a new welfare scheme is launched by the Government for the farmers and it wants to provide its details to the users, which could be the supposed benefits, eligibility of farmers to claim this scheme, other conditions etc. A public servant is assigned the role of an administrator. In the next step, the description of the schema is generated in MushXML. The Service Description Tool (SDT), which has been developed to create, retrieve, update and delete XML documents in a user-friendly GUI (Graphic User Interface), is utilized for the purpose.

The administrator fills in the elements of the new MushML document. The content is derived from the rules and regulations of the concerned government department providing the scheme. Title and Description describe the Title and a short Description of the provided service. Eligibility element explains which class (economic, SC/ST, women etc.) have the right to claim the benefits of the scheme. Required-documents element provides the list of transcripts which are considered as a pre-requisite for the provision of the service (kissan pass book, income proof etc.). The procedure would describe the analytical steps the user has to follow.
In this way, the administrator fills in the rest of the data elements and stores the new MushML document in the database. The new document is stored in the repository (of the Govt. Schemes) which contains all MushML documents. Now the final step is to transform the document into the specific display format (HTML, WML etc.). This is achieved with the utilization of the appropriate XSLT script by the administrator. The user is given access to the desired information in multiple formats, example HTML through a web browser or WML through a mobile phone supporting the WAP Protocol. XML documents can be shown on TV screens, printed on paper, bound in books, read by speech synthesizers, beamed to Palm Tops and projected on to movie screens.

All the style information is placed in a separate document called the style sheet rather than being stored as part of the document itself. Hence a single XML document can be formatted in many different ways just by changing the style sheet. Different style sheets can be designed for different purposes – for print, the Web, presentations and other uses – all with the styles appropriate for the specific medium, and all without changing any of the content in the document itself. CSS [85] allow a developer to vary styles to match the medium in which the content is displayed.

In addition to the information being accessible in multiple formats (text, video etc.), other important facilities provided in the prototype are, accessing external information via links, and provision for Personalized access of information.

As already explained in Chapter 2 some information which is critical and which may change at periodic intervals needs to be updated regularly. Otherwise the farmer may get incorrect information. What has been done to ensure recent information is to use XLink Standard to provide an “external” link to reference the information directly from the website of the concerned department/firm. Uniform Resource Identifier (URI) references offer an excellent mechanism to establish a link from a data item into an additional resource available on the network. It provides globally unique identifiers. This would enable real distributed data storage and also link data to build real knowledge bases.

The XLink specification defines an element called “extended link”, which enables several resources to be listed as participating in the link [49]. A “locator” element is used for this purpose. An individual relationship between two resources is
defined by a separate “arc” element. An arbitrary number of relationships can thus be
defined among the locators. A specified link construct innovates a notion of a link for
which all the locators are remote. Dedicated link databases are supposed to be
established to store these links. Two separate specifications, namely XPath and
XPointer are available as flexible addressing mechanism to refer to external links
[25], [49].

The extended links can include multidirectional links between many
documents. An extended link consists of a set of resources and a set of the
connections between them. The resources could be local (part of the extended link
element) or remote (generally in another document).

4.6 Prototype testing and evaluation

After the prototype development phase, it has to be seen whether the user’s
expectations have been satisfied from it. This phase required the interaction with the
users who could give relevant feedback and comments for improvement or better
performance. Also, every software product has a target audience. For example, the
audience for video game software is completely different from banking software.
Therefore, when an organization develops or otherwise invests in a software product,
it can assess whether the software product will be acceptable to its end users, its target
audience, its purchasers, and other stakeholders. Software testing is the process of
attempting to make this assessment.

So, prototype testing can also be stated as the process of validating and
verifying the system designed. Verification is the process of evaluating a system or
component to determine whether the products of a given development phase satisfy
the conditions imposed at the start of that phase. Validation is the process of
evaluating a system or component during or at the end of the development process to
determine whether it satisfies specified requirements [49]. The testing methods can be
classified as Alpha & Beta and functional and non-functional. Both of these are
explained briefly in the next section.

4.6.1 Alpha and Beta testing

Alpha testing is simulated or actual operational testing by potential users/customers or
an independent test team at the developers’ site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to Beta testing.

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.

4.6.2 Functional vs non-functional testing

Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation. Functional tests tend to answer the questions of the users regarding the functions available to them and the performance of particular features (whether they are executing successfully or not).

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or security.

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirement gaps, e.g., unrecognized requirements, that result in errors of omission by the program designer. A common source of requirements gaps is non-functional requirements such as testability, scalability, maintainability, usability, performance, and security.

4.7 Prototype Testing

The concepts proposed in this study (Chapter 3) were tested for the Ag-IDS for India. These are the concepts of reusability, customizability, and localizability, which would be the strong areas of the proposed framework. In the first phase the prototype was tested for these three concepts.
4.7.1 Reusability

Reusability is defined by Berard [86] as "the degree to which a piece of software can be used for more than one application or in more than one place in the same application". Coad [87] expands upon this by stating that portability is a part of reuse even though his discussion related to the reuse of design results in OOD. In [86] it has been defined that, "Portability is a major component of reusability." It can be agreed on both these points because porting is, by definition, "a reuse of the whole system on a different platform". The whole subject of reuse however is much more than just porting.

Separating the user interface from the application logic would allow user interfaces to be reused when the application itself is moved from one development environment to another, saving a great deal of money and time for the company developing it.

The concepts of reuse and portability can naturally be extended to declarative languages like XML. Another level of reuse is that of portability and this is achieved by the fact that XML is a platform independent textual language. Templates provide a way to separately define code segments that are likely to be used in more than one application or in more than one place within the same application.

4.7.2 Customizability

Customizing an interface using XML is also made easier by the fact that the interface can be updated without making any changes to the rest of the application. A problem with more traditional methods is that it is generally the programmer who has to perform the actual coding of a user interface, even though someone else might originally have conceptualized and designed it. Customization has been defined by Phanouriou in the context of providing specialized content for different user groups in XML language [88]. Customization need not necessarily be just about changing how things look but also how things work. XML promises to support that by providing a unified standardized solution. XML is text based but can make use of style based customization (through CSS, XSLT etc.). The fact that XML is text based makes it easy to alter XML based user interface definitions without having to work with special purpose or even proprietary software for compilation. This allows a
developer to have a number of different styles, each of which is customized for a particular user or user group. Parts that are rendered as buttons in one application could easily be replaced by menu items in another, without even the slightest change in the basic user interface structure. Separating these properties from the main structure does not only increase reusability, but also the degree of customizability.

4.7.3 Localizability

Localizability means that a user interface built on XML as a base standard can be easily translated to one of the Chinese or Japanese scripts as well as any other script supported by the Unicode standard. The concept of a centralized storage for user interface strings makes it easier to find all the relevant strings and to translate them without having to go through all of the interface code locating the relevant strings individually. It is further stipulated that one content element could be defined for each language supported and that the name of a content should be supplied when rendering, causing the renderer to use the strings in that particular content.

4.8 Some user requirements

For each user group some user requirements were formulated and the system was evaluated for the same. These are defined in this section.

i) Services relevant for the user group (e.g. Functionality)

It should be clear for the user what each service is good for and how it works (functionality). At best it is represented in a way, that the user knows intuitively how to use it (awareness), but there should also be additional help available if needed.

ii) Technical aspects at the user’s site (e.g. Security)

The user is expected to be able to use certain technology. In the present case, the user is expected to have a browser and certain technical equipment. It should be mentioned that the level of technology demanded from an ordinary user should be kept as low as possible.
iii) Reliability

The user should be able to rely on the way the system works and to count on its timeliness. Therefore the system should be predictable, in other words the user should not be surprised by the way the system reacts and it should react within a short time, so that the user is informed that something is going on and what is going on.

iv) Usability

The user should have the feeling, that the system is easy to use, that s/he can operate it intuitively, that interacting objects have a consistent behaviour (look and feel), and that the system does what s/he wants in an adequate time with only little effort of the user.

In specific the standard ISO 13407 [89] defines usability as the following:

"Extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specific context of use."

The system was evaluated for the above mentioned four qualities and the performance was found to be as desired.

This standard (ISO 13407) also defines some additional qualities which should be present in a given product (which in our case is the prototype developed). These are defined as following:

- Effectiveness: accuracy and completeness with which users achieve specific goals
- Efficiency: resources expended in relation to the accuracy and completeness with which users achieve goals.
- Satisfaction: freedom from discomfort, and positive attitudes to the use of the product.

The system was run and tested for the above mentioned qualities. Our product was in tune with the above mentioned requirements.
In addition, Nielsen [90] also sees the following as influencing criteria for usability

- **Learnability**: The system should be easy to learn so that the user can rapidly start getting some work done.
- **Memorability**: The system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again.
- **Error rate**: The system should have a low error rate, so that users make few errors during the use of the system, and so that if they do make errors they can easily recover from them. Further, catastrophic errors must not occur.

The system was test run for the above mentioned features and the results were well appreciated by the users.

### 4.9 Navigation concepts reflecting the specific user needs

Some performance factors are concerned with the navigation concept [91] of the system. The navigation must be designed in a way that the user knows:

- where s/he is at the moment
- what s/he can do at this point
- where s/he can go next
- how s/he can get there
- how to get back

at any time when s/he is working with the system.

Considering the above mentioned points, the prototype was tested and the results are presented in the next section.
4.10 DBMS Evaluation parameters

In addition to these points, the characteristics of databases [92] mentioned in Section 3.1., i.e.

- DBMS has to be “scalable”, in both performance capacity and incremental data volume growth
- DBMS must also have a “powerful” design in order to support complex decisions with multi-users
- DBMS must be “Manageable” which will allow one to create and implement the new tables at any time.
- DBMS must have a high “availability” in terms of down time (repair and maintenance)
- architecture should also be “flexible” and “extensible” to keep pace with evolving business requirements
- should have “interoperability” i.e. support multiple applications from different business units

were kept in mind and the prototype was evaluated for the above mentioned points. The performance, as judged by the computer literate audience, is shown in the Section 4.11.1.

4.11 Results

The approach followed in this study is the Alpha testing approach wherein the prototype was test run for two categories of users. One was the computer literate category (mushroom growers as well as non-growers) and the second was the computer illiterate category (only mushroom growers).

The platform was a Windows 2003 Server machine with 4 GB RAM and internet facility. For the computer literate category the prototype was run at NIC (National Informatics Centre) Solan where 20 personnel had gathered for a Seminar. After test running, the personnel were given a questionnaire (Annexure- III) which contained questions relating to the performance of the prototype.
For the computer illiterate (mushroom growers) category, the prototype was run at the Directorate of Mushroom Research, Solan where 35 farmers had assembled for a Training session. They were also given questionnaire (Annexure-IV) which contained questions related to the response of the prototype for various options.

### 4.11.1 Testing of computer literate audience

The prototype was run in front of the computer literate audience and after meaningful interaction, they were asked to rank the parameters (questionnaire shown in Annexure-III) from poor to very good. Their valuable suggestions were also taken at the end of the testing session. Table 4.4(a) shows the results of the ranking of testing parameters after the execution of the software and Table 4.4(b) shows the data relating to the DBMS functioning. The same have also been shown in the form of charts in Fig. 4.17(a-f) and Fig. 4.18(a-e)

#### Table 4.4(a) Testing results for computer literate audience

<table>
<thead>
<tr>
<th></th>
<th>Poor</th>
<th>Average</th>
<th>Good</th>
<th>Very Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>Security</td>
<td>-</td>
<td>6</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Functionality</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Usability</td>
<td>-</td>
<td>1</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>Repeatability</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>Efficiency</td>
<td>-</td>
<td>2</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

![Reliability Chart](Fig. 4.17(a))  
![Security Chart](Fig. 4.17(b))
The prototype was got tested among the computer literate faculty for the parameters related to reliability and security. The reliability (Fig. 4.17(a)) aspect was checked by running the system and gave predictable results. Testing against the security (Fig. 4.17(b)) parameter gave good results as the users were unable to access the backend database.

![Functionality](image1)

**Functionality**

<table>
<thead>
<tr>
<th>Level</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor</td>
<td>0%</td>
</tr>
<tr>
<td>Average</td>
<td>0%</td>
</tr>
<tr>
<td>Good</td>
<td>25%</td>
</tr>
<tr>
<td>Very Good</td>
<td>75%</td>
</tr>
</tbody>
</table>

For the functionality aspect (Fig. 4.17(c)) 75% of the users gave very good to the system. The aspects like the “help” availability and the awareness of the user in operating it were taken into consideration. Regarding the usability (Fig. 4.17(d)) the system was tested in relation to the ease of use and whether users can operate is intuitively, and whether the system was consistent in behaviour. 85% of the respondents gave “very good” for the functionality.

![Usability](image2)

**Usability**

<table>
<thead>
<tr>
<th>Level</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor</td>
<td>0%</td>
</tr>
<tr>
<td>Average</td>
<td>10%</td>
</tr>
<tr>
<td>Good</td>
<td>85%</td>
</tr>
<tr>
<td>Very Good</td>
<td>0%</td>
</tr>
</tbody>
</table>

![Repeatability](image3)

**Repeatability**

<table>
<thead>
<tr>
<th>Level</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor</td>
<td>5%</td>
</tr>
<tr>
<td>Average</td>
<td>20%</td>
</tr>
<tr>
<td>Good</td>
<td>50%</td>
</tr>
<tr>
<td>Very Good</td>
<td>25%</td>
</tr>
</tbody>
</table>

![Efficiency](image4)

**Efficiency**

<table>
<thead>
<tr>
<th>Level</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor</td>
<td>0%</td>
</tr>
<tr>
<td>Average</td>
<td>10%</td>
</tr>
<tr>
<td>Good</td>
<td>40%</td>
</tr>
<tr>
<td>Very Good</td>
<td>50%</td>
</tr>
</tbody>
</table>
Similarly the aspects related to repeatability (Fig. 4.17(e)) and efficiency (Fig. 4.17(f)) were tested. Repeatability gave very good results with 95% of the users responding with “very good” as the system gave identical results when any command was repeated. For the efficiency parameter 10% gave average, 40% good and 50% very good.

![Scalability](image1)

**Fig. 4.18(a) Scalability**

![Manageability](image2)

**Fig. 4.18(b) Manageability**

The scalability (Fig. 4.18(a)) and manageability (Fig. 4.18(a)) aspects gave very good results. More data was entered and the performance was found to be the same. Respondents were satisfied after adding new tables when tested for the manageable parameter.

![Availability](image3)

**Fig. 4.18(c) Availability**

![Flexibility](image4)

**Fig. 4.18(d) Flexibility**
The availability (Fig. 4.18(c)) feature received some setback as the backup was not available for longer period when electricity went off. Also, there was no alternative available in the event of the system crash. It was suggested to make alternative arrangements to improve this aspect so that the system is available 24 hours a day. The users expressed satisfaction with the flexibility (Fig. 4.18(d)) aspect that new features could be incremented into the system when desired.

![Interoperability](image)

Fig. 4.18(e) Interoperability.

The interoperability (Fig. 4.18(e)) aspect also received good response from the users with 75% saying “very good” and 15% very good. Data in different formats was tested for the same.

**Table 4.4(b) Testing results for computer literate audience**

<table>
<thead>
<tr>
<th></th>
<th>Poor</th>
<th>Average</th>
<th>Good</th>
<th>Very Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>Manageability</td>
<td>-</td>
<td>2</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>Availability</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Flexibility</td>
<td>-</td>
<td>1</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>Interoperability</td>
<td>-</td>
<td>2</td>
<td>15</td>
<td>3</td>
</tr>
</tbody>
</table>
4.11.2 Testing of computer illiterate audience

Table 4.5 shows the data for the computer illiterate or the mushroom growers who are not conversant with computers. While a majority of them were satisfied with the content, navigation, some were uneasy with the “ease of use” and “memorability”. This was because many of them had worked on a computer for the first time. Also, a majority of them were not comfortable with English language. They desired the information in Hindi. The data of table has also been shown in the form of pie charts in figures 4.19(a-e).

Table 4.5 Testing results for computer illiterate audience

<table>
<thead>
<tr>
<th></th>
<th>Poor</th>
<th>Average</th>
<th>Good</th>
<th>Very Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>-</td>
<td>5</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>Ease of use</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>Navigation</td>
<td>-</td>
<td>-</td>
<td>9</td>
<td>26</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>-</td>
<td>10</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>Learnability</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>Memorability</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig. 4.19(a) Content

Fig. 4.19(b) Ease of use
The results show that the farmers were satisfied with the content. Those having some doubt desired that marketing related information be also included in the system. Regarding “ease of use”, Fig. 4.19 (b) shows that only 20% responded with “good” because they were not conversant with the computers.

Farmers were satisfied with the navigation (Fig. 4.19 (c)) as it was a menu driven experience for them. Overall they were satisfied. The satisfaction ranged from average to very good. They were excited that something was being done to give them quick information in time of need.
Regarding Learnability (Fig. 4.19(e) and Memorability (4.19 (f)), the response ranged from poor to very good. Only 20% gave very good for learnability and 10% for memorability because most of them didn’t have much experience with computers.

Overall, the farmers were satisfied and when asked whether they would subscribe to the services, once the AgDIS was fully implemented, the farmers were very enthusiastic and said that it depended upon what information they would be getting and in what form.

Regarding the specific tools an end user must use, such as web browsers or XML Editors, most of the commercial products tested were found to provide acceptable performance. The overall analysis of the results of the study showed that the farmers can acquire high level of fluency with the technology. The appropriation of technology will empower them to access the information they desire with ease.