Chapter 7

Implementation

In the present chapter, an outline of the implementation of DPW architecture using the state-of-the art technology is discussed. Only the outline of an implementation is given here. No implementation has been done as a part of the present work. Implementation of the system has not been done, because of the following reasons:

- The system is complex. Even a prototype implementation will take many man hours.

- Our focus is more on academic interests, mainly identification of issues and logical solutions. Implementation is beyond our focus.

- An office automation software consists not only of DPW, it also consists of legacy software for financial accounting, database, e-mail, conferencing etc. Therefore, the DPW software is to be integrated as a component with other commercial office automation software, like Lotus Notes. A DPW software itself cannot survive as an alternative to Lotus Notes. The integration can only be done as a joint venture with the authority of the software.
• Tools like workflow, XML etc. required for DPW were in a primitive stage when the work was started. Vendors like Oracle, only recently incorporated workflow and XML in their new products. But the technology as a whole is still in a flux, and is yet to settle down as accepted standards. In such a situation, an implementation may not be long lasting.

• The security algorithms and protocols are proprietary in nature. There are restrictions in using them.

• Acceptability and performance of such a system is meaningful when we get feedback from real life offices using such a system, which again is not possible with the present setup of ours.

In fact a part of DPW architecture was implemented in a project in IIT, Guwahati, based on our papers [36, 37]. Here the security and context part was not fully implemented. The prototype implementation is working fine and the performance is reported as satisfactory [33].

7.1 An Outline of an Implementation

Java introduces a new model of client/server interaction for the Web. A small component like program called applet can be written in Java and can be downloaded into a browser that is Java compatible. An applet is portable, can be executable inside a java-enabled browser on any computing platform. Java applet provides, on its own, a secure environment within a browser. Web and Java provides a solution for distributed computing but the solution is not complete, particularly in workflow automation environment. HTTP is a stateless protocol and the interaction is initiated by the client (client pull).
The problem lies in the basic dichotomy between the Web paradigm and the workflow paradigm. The Web is composed of a collection of non-persistent state objects, whereas workflow, in particular office document production workflow, requires a degree of persistence of state among its objects to enable the completion of the assigned task. Moreover, Workflow very often needs server initiated interaction (server push). Of course, Netscape and Microsoft have announced support for push technology for scheduled broadcast communication. This needs to be enhanced for an event-based push.

The architecture for the DPW can be implemented using Web technology. The client environment may be a general Java enabled Web browser, where the client is a Java applet. Agents of the client logic layer may be implemented as Java objects in the applet. The interfaces of the view module can also be implemented in the applets. The protocols can be implemented on the top of the HTTP protocol. Since server push mechanisms are not yet matured, we can modify the protocols, where necessary, with client pull only. The manager modules of the server logic can be implemented as Java servelets. Of course, server push can be simulated using the Java RMI mechanism. In DPW, server push is required in a scenario like, a reviewer is online and a case is received by the arbiter for the reviewer, the arbiter updates the inlog of the reviewer in the storage and the same updated inlog is to be sent to the online reviewer. Here the communication is initiated by the arbiter.

Alternatively, the DPW architecture can be realized in a distributed object model. We have a matured de facto as well as de jury open standard for distributed object infrastructure, that is Common Object Request Broker Architecture (CORBA). CORBA objects can exist anywhere on a network, and their location is completely transparent. Details such as the language in which an object is written or the operating system on which it currently
runs are also hidden to clients. The implementation of an object is of no concern to an invoking object; the interface is the only consideration a client must make when selecting a serving object. CORBA provides an open environment built for integration, change and evolution, and is suitable for office automation software. The standard CORBA objects are grouped as CORBA services, CORBA facilities and CORBA domains. The CORBA services provide system-level services which may be used to design the objects for other groups. The CORBA services provided in CORBA 2.0 are: Concurrency, Events, Externalization, Licencing, Lifecycles, Naming, Security, Time, Trader, Start-up, Persistence, Properties Query, Relationship, Transactions, Collections etc. CORBA facilities, which provide horizontal and vertical application frameworks used by business objects, include User Interface, Information Management, System Management, Task Management etc. The CORBA domains provide object solutions to some standard domains like Financial services, Health-care, Telecommunications etc. Application objects can be designed from these standard objects through inheritance and that is what exactly needs to be done in realizing the DPW architecture. Another advantage of CORBA is that legacy software can be integrated in a CORBA-based architecture using CORBA wrappers. Details are available in [23]. CORBA 2.0 also incorporates a binary protocol for communication between ORBs, called the Internet Inter-ORB Protocol (IIOP), through which two ORBs can talk to each other. IIOP and HTTP can coexist in the same setup.

The DPW architecture can be implemented within the framework of CORBA so that it can be a component of an open system and can be integrated easily with existing software packages. With the CORBA object bus, ORB, in both the client as well as server side, a Java object can directly
talk to the server or vice-versa. The client environment is the standard web browser. The client is a Java Applet which will run in the browser. The agents of client logic module and corresponding interfaces are designed as Java objects in the applet. A Java based lightweight CORBA ORB, called Java ORBlet, is loaded in the client side. The objects of the client applet can communicate among themselves or with remote server objects through the interfaces and stubs of the ORB. The applet can be initially loaded using a HTML page and HTTP protocol. The client authentication can be implemented by authenticating the applet during loading, using Sun Microsystem's *applet certification scheme*. Initial synchronization of clock of a client with that of the arbiter can be done by using the CORBA service called *Time*. The Crypto Agent of the client can be designed as a derived object which inherits the *Security* and *Time* objects of the system level CORBA services. The Security service of CORBA provides authentication, confidentiality, content integrity, non-repudiation etc. The crypto can get time attributes for timestamping by inheriting from Time service. The User agent also inherits some traits from Security service. The other services like Naming and Event etc. can be automatically used by ORB.

On the server side, the managers: Production Manager, Workflow Manager, Storage Manager and User Managers of the architecture can be implemented as CORBA server applications. CORBA server applications are regular CORBA objects. The CORBA servers are plugged to an ORB object bus. If the HTTP server is CORBA compatible then it can be directly plugged to ORB otherwise it can be wrapped with a CORBA wrapper to make it CORBA compatible. Similarly a standard workflow engine can be wrapped to make it a workflow manager. Of course, CORBA incorporated a workflow service in its recent version. In that case, the workflow manager
can be implemented inheriting this service. The storage manager can be implemented by inheriting from the Persistence and Query services of CORBA services. The Persistence service provides a single interface for storing objects persistently on a variety of storages- including Object Databases, Relational Databases and simple files. PSQL can be implemented inheriting traits from Query services.

7.2 Implementation of Page Cube

The data storage model for the DPW is Page Cube, discussed in Chapters 3, 5 and 6. The Page Cube can be implemented in different ways. One of the following approaches can be taken.

7.2.1 Relational Database Approach

The Page Cube model can be implemented using relations. It is useful to consider these relations as part of a multi-dimensional data cube model used in data warehousing. The relations are then regarded as dimension tables and the hierarchical relationship can be represented by either a star schema or the snowflake schema [28]. In the star schema, the dimension tables are not normalized. Where as in the snowflake schema, they are normalized. Based on the complexity of the dimensions, we can choose one of the two alternatives. The advantage of a star schema is that it is easy to define hierarchies and it reduces the number of joins. It is a common practice in multi-dimensional modelling that when the dimensions are simple in nature, such dimensions are incorporated into the central fact tables as attributes as there is no justification of maintaining a separate dimension table [29]. The advantage is that expensive join operations between the central fact table
and those dimension tables can be avoided thereby improving the efficiency. Of course, there is no clear rule for such justification. It is a design decision. In the schema of the page cube, some of the dimensions can be incorporated in the central fact tables. *Time, Category, Class* and *Status* are such dimensions which can be incorporated in the central tables without any loss of generality. Time is a significant dimension and needs a special treatment for it. Once time is incorporated as an attribute of the central tables then there is no need for a separate timeId. Since, in our scheme, the central arbiter will provide the time, a unique time can be assigned by the arbiter to a page. Therefore, it can also be used as a unique identifier for a page, the pageId. According to this modification, the arbiter ensures that at a particular time one and only one page will be created. A representative simple star schema of PC is as follows:

**Dimension Tables:**

- **Type** (typeId, typeDesc, parent........)
- **Topic** (topicId, topicDesc, parent.......)
- **User** (userId, userName, address, .......)
- **Domain** (domainId, domainDesc, parent...)
- **Role** (RoleId, RoleDesc, parent...)  
- **Dpw** (dpwId, dpwDesc, ........)

**Other Central Tables:**

- **Production** (pageId, topicId, typeId, category, class, userId, roleId, domainId, dpwId, ......)
Storage(pageId, location, size, signature, state, pageText,...)
Flow(pageId, senderId, senderRoleId, sentTime, receivedTime, ReceiverId, ReceiverRoleId)

Relations to represent graphs: DGs and IDGs
Citation(citingPageId, citedPageId, plugTime, unplugTime, citeCount)
DocumentPart(documentId, partId, plugTime, unplugTime)
ContextDocument(contextId, documentId, plugTime, unplugTime)
CaseContext(caseId, contextId, plugTime, unplugTime)
PartPortion(partId, portionId, plugTime, unplugTime, kindOfEdge)
UserRole(userId, RoleId, plugTime, unplugTime)
RoleDpw(RoleId, dpwId, position, plugTime, unplugTime)

The authorization base and revoked authorization base relations are discussed Chapter 6 and are not incorporated here again. The Rule-Base for authorization discussed in Chapter 6 can be implemented as active rules of active database systems. Authorization as active rules is also discussed in [13] and the rule structures can be taken from this work. The names of the relations and of the attributes are chosen in such a way that they are self explanatory. For brevity, a data dictionary for the schema is not provided. A page may be on more than one topic, may also belong to more than one class. Therefore, the normalized Production relation will have more than one tuple with the same pageId. Therefore, pageId alone cannot be the primary key of the relation. Without any loss of generality, if we consider multiple values of topicId as well as of class as a single atom, where values are delimited somehow, then the given denormalized relations serve the purpose. The content of a page is considered as an attribute, pageText, in the relation Storage. But
in practical implementation each page can be stored as a separate file, with pagId as the file name, and only the location of the file in the file system is stored in the location attribute.

7.2.2 XML Database Approach

The present trend for digital document encoding and interchange is to use descriptive markup. In the light of the state-of-the-art technology, XML is the best tool for describing contents of office documents. Therefore, it can be a suitable tool for implementing the page cube model. An XML page cube is a collection of XML pages. The root tag of a page is `<page>`. The `<page>` tag contains three tags `<profile>`, `<body>` and `<tail>`. The `<profile>` describes the dimensions of the page cube, `<body>` describes the content of a page. DGs and IDGs are implemented by nesting of elements in the `<body>` or with external references as links. The `<tail>` section contains external reverse links of different graphs. The structure of a sample XML page is as follows:

```xml
<?xml version="1.0" ?>
<!DOCTYPE page SYSTEM "pageCube.dtd">
<page id="2001.12.30.13.56.45">
  <profile>
    <dimension>
      <type id="">message.officeOrder.notice</type>
      <topic id="">leave.casual.special</topic>
      <category>document</category>
      <class>case</class>
    </dimension>
  </profile>
</page>
```

133
The dimension hierarchies can be implemented as nesting of tags in `<profile>`.
Each dimension may have a separate page with a distinct page ID or all dimension hierarchies may be defined in a single page. In the dimension page(s) the `<body>` and `<tail>` section may be naturally null. For checking the validity of a page belonging to the XML page cube a Document Type Definition (DTD) is to be designed, which specifies the grammar of elements (tags), attributes of tags and their relationships. A representative XML DTD, named `pageCube.dtd`, is as follows:

```xml
<!DOCTYPE page [ 
<!ELEMENT page (profile, body, tail)> 
<!ELEMENT profile (dimension)> 
<!ELEMENT dimension (type?, topic?, time?, category?, class?, 
    user?, domain?, dpw?, state?)> 
<!ELEMENT type (type*, name)> 
<!ELEMENT topic (topic*, name)> 
<!ELEMENT time (year, month, day, hour, minute, second, msec)> 
<!ELEMENT category (#PCDATA context | document | part)> 
<!ELEMENT class (#PCDATA rule | case | support)> 
<!ELEMENT user (name, address)> 
```
7.3 Discussion

In the present chapter, an outline of the implementation of a DPW system is discussed. The integrated environment of the Web, Java and CORBA provides a secure and reliable platform to implement a DPW system. Implementation of each tier of the three-tier architecture has been for DPW
discussed in detail. The implementation of the client as a Java applet pro-
vides platform independence of the process, because a Java applet can run
in any Java enabled browser. The independence is at the cost of perfor-
ance. Since Java applets are interpreter based, it is slower than compiled
programs. As a part of Java security policy an applet is not allowed to write
in the local file system of the client. This may pose a problem in real life im-
plementations, because local caching may be required. Of course, this can be
handled by presently available solutions to address the problem, like Jamie’s
file system. So far as the implementation of the page cube is concerned,
relational model is matured and mathematically well defined. Moreover, the
relational query languages are powerful. But this approach may be possibly
inefficient, because the representation of such irregular dynamic structure in
flat relations is complex and retrieval of information may involve many join
operations. Performance degradation and lack of openness is obvious. On
the other hand, XML approach may be more efficient but it is too early to
say, because XML tools are less available. Platform independence of data
can be provided by XML, because an XML data file is an ASCII file with
simple encoding. But XML is verbose, as a result increase in file size is a
problem. It can be addressed in XML compression techniques. The product
XMill is an example of an XML compressor. But since there is a possibility
of a part being split, the content of a page needs to be marked up somehow.
XML may provide a meaningful markup scheme for office documents. There
are different XML query languages like Xpath, XML-QL, Quilt, Xquery etc.
It is to be seen whether PSQL can be implemented on top of one of these
existing query languages. Only a guideline of implementation is given here.
During actual implementation, new issues may prop up and the guideline
itself may need to be modified if necessary.