CHAPTER 4

OBJECT VERSIONING IN XML DATA MANAGEMENT

4.1 INTRODUCTION

Basic object-oriented concepts like objects, class, inheritance and aggregation provide rich modeling power to solve many complex engineering applications like Computer Aided Design / Computer Aided Manufacturing (CAD/CAM), Computer Aided Software Engineering (CASE), Multimedia and other collaborative design applications. The collaborative design and cooperative works are going to be the key feature for the next generation applications. The collaborative work approach allows many people to work on shared understanding of planning, assumption, project goals, design themes and other needs of everyone.

Engineering design is an evolutionary process. The design of a product goes to several cycles before getting matured as the final design. The versions of a design component in collaborative design need to be captured and manipulated. In the design process, it is essential to provide a mechanism to capture the evolution of the design process. In recent days, XML is widely used in internet applications, collaborative design, software configurations and platform independent way to represent and exchange data. Design parameters and configuration details specified in the XML form can be easily exchanged between collaborative environment workstations.
Version management is one of the challenging research areas in XML data management. In software configuration management and software installation, version manipulation techniques are essential to manage different versions of XML data. In this section, a novel technique such as object versioning is proposed by combining the features of objects and XML data management. The configuration details and design details in the form of XML are mapped into Java objects and Java objects to XML data file using Java Architecture for XML Binding (JAXB). Versions of an object are created whenever an object is modified. The proposed technique simplifies the task of managing versions of XML data.

Engineering design is a complex process, which consists of management of complex design components. The design of a component consists of many sub components, which in turn may consist of many low level components. All these components evolve as the design progresses. These evolved individual design components are called versions. The higher level individual component may use different versions of low level components. A data model should provide adequate support to keep track of these versions.

Conventional database management systems like RDBMS do not have the rich data modeling power to support engineering applications such as CAD / CAM, CASE and Collaborative design applications (Chou et al 1986; Chen et al 2005; Li et al 2007). An attempt has been made to extend the core object model to support collaborative design on a network of workstations (Janakairam et al 1997). But this extended model does not take into account of the evolutionary nature of design products. As the design of products evolve, different versions of product are created. This object versioning in XML databases for collaborative design and software configuration management is an attempt to integrate the version semantics.
proposed in (Ramakrishnan and Janakairam 1996) with collaborative design (Chou et al. 1986; Chen et al. 2005; Li et al. 2007) to model the evolution of design products. The semantics of version are captured with component class and component version class. The configuration file used in the software installation environment and design details of a pillar drilling machine is taken as an example to create and manage the versions. In the design of a pillar drilling machine consists of components like head assembly, base assembly and pillar assembly. Chief designer assigns the design work of a component to individual design engineers. The individual components are designed and submitted to the chief designer. All the components are assembled at the chief designer’s level. If a new version of the machine is to be created, then the chief designer sends messages to the appropriate individual designer. New versions of components are generated by the collaborative activity of the individual designer.

4.2 VERSIONING CONCEPTS

Versioning is an important function that needs to be included in database management system as well as in integrated CAD and software configuration management. Version control is the ability to manage the relationship between different instances into a meaningful structure and support traversal and other operations on those structures. Version control has long been considered as an important issue for many development tools such as software engineering, software configuration management, CAD, technical documentation and authoring. The primary challenges for versioning in this area are to support opportunistic, the open ended design process requiring the preservation of the earlier design details identified in the design process, the reuse of the existing designs and searching for alternatives. The major aspects in version management are semantics of version creation, manipulation, version naming, change propagation and name binding.
Many commercial database applications require the ability to create and access multiple versions of a component. Examples of these are identified in historical database, such as those used widely in business and commerce for maintaining multiple versions of legal or financial documents. Other important uses of versioning are found in databases and used in the software configuration. It is widely recognized that version control is one of the most important tasks in an environment in which users need to generate and experiment with multiple versions of a component before selecting one that fulfills their needs. The basic requirements of a system that support versioning are

- All persistent objects can have versions and there should be no limit on the number of versions an object should have.

- Applications should be able to access either the current version (which should be the default) or any specific version. Only the current version of an object can be updated. Access to earlier versions should be read only normally.

### 4.3 VERSIONING REQUIREMENTS FOR ENGINEERING DESIGN

Engineering design applications involve the management of complex and evolutionary design of engineering component such as aircrafts, ships and automobiles. These components are characterized by numerous sub components they constitute. These components are to be efficiently stored and managed by the system. The design task is characterized by a set of rules of thump, specifications and techniques. The designers use these rules to design the various components of the artifacts. The designer often uses the knowledge of an earlier design to solve problems, referred to as design reuse. Routine design involves modifications of the existing design to suit additional
requirements and specifications. Due to this nature of the design process, the system should provide adequate support for managing the different versions of a component corresponding to its different stages of creation and evolution.

4.3.1 Existence of Product Versions

Generally in the industry, the design department is involved in creating an improved design of its product. The improvement may be in terms of better performance or additional facilities, or suit different requirements. Hence, for any industry there will be several versions of its products. For instance, in drilling machine manufacturing industry, there exists several series of drilling machines. In each series there may be different versions to suit the customer requirements. Each series is derived from an existing series with suitable modifications incorporating the latest advances in technology. Further different versions of the drilling machine may use the same versions of a component.

4.3.2 Versioning in Design Process

A complex artifact consists of many components. For instance, a pillar drilling machine consists of a pillar, head and base. These components in turn consist of lower level components. For example, the head consists of spindle, quill, motor, hand lever, etc. Each component is normally designed by a design engineer. The designer may use an existing version of the component retrieved from the database if he finds that it satisfies his requirements. Else he modifies an existing version to suit his requirements. There may be several versions of components, which would satisfy the requirements. Hence the designer may need to experiment with multiple versions of the design. When designer retrieves a version he must be able to understand the necessity that leads to its creation, its characteristics, the
specifications that govern the use, the differences from other versions and the lower level component versions used. This knowledge must be captured as a part of the versioning mechanism to give full support to the design process and also meaning to different versions of the component.

4.3.3 Interdependencies

Interdependencies between the versions of different components exist in a product. For example, changing the design of a head of pillar drilling machine may require that the pillar be modified or replaced by another version. This interdependency between head and pillar needs to be captured adequately by the version management mechanism to ensure correct design.

4.4 VERSIONING IN OBJECT ORIENTED DATABASES

Object oriented databases are more suited for complex engineering design applications. The core object model consists of classes, objects, messages and inheritance. The state of the entity is represented using the attributes and its behavior using the methods in the class definition. Schema refers to the hierarchical organization of all classes in the database and their interrelationships. The class hierarchy represents the generalization or specialization of relationships among classes in terms of the super class / subclass. The composite class hierarchy represents the aggregation relationships between classes. The complex component can be represented as a set of components using the part-of relationships.

4.4.1 Class and Object Versioning

The evolutionary nature of the design or development process, changes to the schema is considered as a rule than exception. As the design
progresses, the schema evolves by adding changes to the class definition, to class hierarchy and to the composite class hierarchy. These design process evolutions have led to class, schema and object versioning (Monk and Sommerville 1993). Schema versioning refers to the creation of a new schema version based on changes to the hierarchical organization of classes. Class versioning refers to the creation of a new version of an existing class definition whenever a modification to the class definition such as addition of attribute or methods. An object is called as versioned when it changes its state, i.e the attribute values are modified. The version derivation hierarchy can be represented in a version hierarchy graph.

A design object is an aggregation of design data, which is treated as a coherent unit by the designer. A design object is usually a complex object obtained by bringing together a number of component objects. The design object consists of design and non design attributes, and is stored in design database (Hatz 1990). Design attribute value changes affect the functionality and behavior of the object. There are no changes in functionality or behavior when changes are made to the non design attributes. Whenever changes are made to design attributes, a new version is created and changes to the non design attributes result in equivalences (Ramakrishnan and Janakiram 1996). Version history explicitly records the ancestor / descendant interrelationship among versions and distinguishes is derived from the relationship. When the concepts of version history and component hierarchy are combined, the result is configuration (Hatz and Chang 1988). Classes in the schema are organized as inheritance or aggregation hierarchy. Inheritance hierarchy exhibits Is-a relationship as shown in Figure 4.1 Aggregation hierarchy shows the Part-of relationship as depicted in Figure 4.2.
4.4.2 Inadequacy of the Existing Versioning Mechanism

Any component has certain characteristic attributes associated with it. These key attributes play an important role in the design of components. For example, the design of the boiler may be considered. An element used in the fabrication of boiler, for example, steel has many properties associated with it such as Young’s modulus, Poisson’s ratio, yield stress, melting point, and thermal conductivity. Depending on the design characteristics, only a few of the properties are considered as key factors to the design. Changes to the
values of these key attributes will result in a new design. Depending on the
design characteristics, object version may or may not lead to a new version of
the design. Hence it is seen that the existing versioning mechanism fails to
capture the design process as a concept. This versioning mechanism should
seamlessly be integrated with the design process. It should create new design
versions based on the functionality and behavior of the component rather than
merely based on the changing class definition or attribute values.

Whether the design of a component is changed, the design of
another component may have to be altered. For instance changing the design
of a head of pillar drilling machine may require a change in the base as well
as in the pillar. This is due to the interdependency between head and base.
These interdependencies need to be captured in order to ensure the correct
design of machine. The versioning process should also capture the semantics
of each version i.e. information regarding the necessity for the creation of that
version, a specification that governs the use of that version and differences
from other versions etc.

4.5 VERSION MANAGEMENT

The version management has been used to record the evolution or
modification of data. Modification of data or code often introduces new bugs,
and in such a case it is very essential to know exactly what was changed. The
version management mechanism helps a group of users working on a set of
objects and at the same time can modify the objects without the interference
from others. The version supported systems can help in managing and tracing
parallel development process. The ability to create and maintain multiple
versions of the same object is very important, especially in an environment
where the specification of objects tend to change or evolve over time. This is
a common principle in design systems, where the designs are developed in an
iterative process. Version control allows collaborative designers to express their different design ideas. The critical issues in version management are

- Usability
- Technology supported
- Extensions required
- Environmental parameters

4.5.1 Usability

This category contains issues that affect the way in which the version related services in a collaborative activity environment are used. Either end user can utilize the version facilities of a particular application or developer who is incorporating the version functionality of the surrounding environment into the application. The overall goal for addressing issues in this category is to develop solutions that are powerful enough to meet the user needs, which are not too difficult to use or confining for the user. The major sub issues associated with the user interference are

- Overhead
- Granularity
- Version selection
- Variants
- Propagation

4.5.2 Technology Supported

Version control techniques can serve as a basis from which new computer supported cooperative work related technologies can be developed and the existing one can be improved. These category version control issues
can exploit to support and enable computer supported cooperative work technologies. The various issues associated with this category are

- Transition between different modes of collaborative work
- Group awareness
- Work process
- Transaction models

### 4.5.3 Extensions Required

This type of issues related to areas in which additional functionality for versioning concepts is required to be useful in a collaborative environment. Effective mechanisms to address these issues will increase the usability of various support mechanisms within a computer supported cooperative work environment. The issues associated with this are

- Access control
- Merging

### 4.5.4 Environmental Parameters

Issues placed in this category are those, which arise from and relate to the circumstances of the collaborative environment and system architecture. They include issues that impact both the design and implementation strategies employed for version related services. Design related issues, such as integration with existing tools and single or multiple system architecture, influence the shape of versioning services by placing requirements on the functionality which must be provided. Implementation related considerations, such as interoperability, have more impact on the way a design is implemented within a particular environment.
• Integration with existing systems
• Single or Multiple System Architecture
• Interoperability

4.6 COLLABORATIVE DESIGN

Object databases have been recognized as providing rich modeling capabilities for next generation applications especially CAD/CAM, CASE, multimedia and software engineering. The core object model consisting of objects, aggregations, associations, messages and inheritance, however it is not adequate to tackle all the requirements of these applications. Since collaborative activity is one of the key features for the next generation of applications, it is essential that the core object model can be extended to support collaborative activity.

Collaborative design is a form of Computer Supported Cooperative Work (CSCW) used in CAD applications. CSCW has emerged as a research discipline during the last decade. In this research domain, human-to-human communication is improved by using computers and communication devices. The growing interest in CSCW reflects the demands of industry for improved tools to aid the coordination and control of group activity. CSCW systems are primarily concerned with supporting a number of users cooperating to address a particular problem. People can cooperate synchronously or asynchronously. In a collaborative design, the nature of interaction is asynchronous. It is important to understand the nature of collaboration before it can be modeled as a system. Most of the industrial design projects are increasingly done by a team of designers and require expertise in various areas. Recent developments in computer networks and communication show a lot of potential for supporting the team design activities, possibly in a distributed situation. Many computer applications have been developed to aid designers in visualizing
design objects and solving analysis problems in accordance with the increased performance of hardware and software technologies. However, the use of these techniques largely remains in a single user environment.

4.6.1 Requirements of Collaborative Design

Collaboration is fundamental to group work. Collaboration occurring in different environments may vary. The collaborative step establishes a solid framework for efficient project management and decision making by establishing clear communication channels, define roles and responsibilities, and also provides potent quality control mechanism. Conflict occurring among team members working on a design problem, software project or document is resolved through collaboration. It is essential to understand the nature of collaboration occurring in a particular domain before it can model into a system. The nature of collaboration occurring in the design process can be summarized as follows

- **Solving conflicts of viewpoint and conflicts of interest**

  Many design problems are ill defined in the beginning. Different team members of a design group can have conflicting views of the problem. The design problem is defined in increasing detail in progressive steps by identifying the constraints on the solution. Conflicts of viewpoint can be seen as conflicting constraints on the design problem being solved in a collaborative fashion. Conflicts of interest arise when the optimizing functions of group members or different teams working on the problem are different.
Asynchronous collaboration

Collaboration among the members of design team occurs in an asynchronous fashion. Each designer works in his own design space consisting of a set of design objects at his convenient pace. A single design space may span from several hours to days and hence any collaboration in design environment should support long duration transactions. A practical approach is required to capture dependencies among design spaces of different designers and propagate necessary changes from one design space to the other as the design evolves.

During the course of collaborative design development, the information generated by the members of the design team must be readily available to other team members, so that they can respond to propose developments as they occur. Many significant questions arise in the support of asynchronous design collaboration. One is the issue of version maintenance. What happens to old information as it is incrementally updated? Other issues concern establishing protocols for the kinds of interaction between team partners and the team partners are permitted to have information developed by a particular designer, when significant changes have been implemented by an individual member. Only limited subsets of these issues were addressed in establishing a method for support of asynchronous design communications within the collaborative design.

Design annotations

Design annotations often convey the designer’s mind behind his decisions. Design annotations are critical to the understanding of the design. Substantive meta level communication occurs among designers through design annotations. Support for recording design annotations in the database may be essential in a collaborative environment. Attribute annotations, which
record the designer’s annotations when attribute values are changed, can be introduced. It is possible for other designers to examine the annotations.

- **Ability to undo individual actions**

  The need for undoing selectively individual actions when several members work simultaneously on a shared document is important. Such facility for selectively undoing the earlier changes without affecting other designer’s changes is an essential ingredient of the design environment when a designer employs trail and error techniques.

- **Different roles of members / teams / groups**

  Members play different roles in design teams. Often the role of design critic is to point out the weak point in a design. The role of a product manager can be seen as one of the arbitrators between groups or members. Any collaborative systems should provide role modeling including authorizations for accessing relevant design objects.

- **Support for weak and strong consistencies on design objects**

  It is often important to note that a designer allows some inconsistencies in his design into the process of evolution. Often, as the design evolves in the rough design stage, constraints are dynamically identified. Systems providing strict consistency checks may prohibit inconsistent data. This is a contradiction to the basic nature of the design process, which probably requires more sophisticated and tunable consistency mechanisms on design objects. As the design evolves in the rough design phase, constraints are dynamically identified. In collaborative design, the designer dynamically identifies his constraints space with that of the collaborating team members.
**Synchronous collaboration**

Synchronous technologies support real time, face to face interaction between parties in spatially remote locations. The recent trends in synchronous communication technologies point to interesting directions for support of design communications. However, the bandwidth or speed of transmission available for communication between computers limits interaction with the applications. Limited bandwidth results in a remote view of the application, which lags behind the actual operation of the application. During synchronous communication, an ISDN digital telephone network provided a dedicated, steady stream of communication between the collaborators. Application sharing technology allowed the design database and embedded applications to be synchronously shared among collaborators, operating remotely located machines.

**4.6.2 Basic Collaborative Design**

Design via networks, local area network or internet is also known as collaborative design. Research in this area focuses on doing in designing an environment that supports design teams operating in physically independent locations, simultaneously or asynchronously. Standard equipment for collaborative design includes white board or equivalent software that supports the simultaneous viewing of design details, audio contact, video contact, procedural or administrative documentation facilities.

The core object model provides a number of concepts such as object, unique object identifier, class, class hierarchy, message passing, versioning and schema evolution. The core model allows complex objects and navigational access, which are very useful in design databases. The core object model does not provide support for collaborative work. Often designers work on their workstations, which are connected by a network. The
basic nature of work of the designers involves collaborating with other members of the team. In such a design environment, it is often necessary for the model to support collaborative activity among designers working on a network of workstations.

The basic collaborative design system should meet the following requirements.

i. It should provide enough flexibility for individual designers to work on design space and at the same time provide a means for cooperation with other designers.

ii. The collaborative activity should not unduly interfere with the design process such that it slows down the design process itself.

In collaborative design systems, designers work on their own workstations, which are connected by a network. The task of each designer is to design a component allotted for him and passing messages to others whenever the component design affect other components. The chief designer of the design team passes the detailed specifications of the component to the corresponding designer.

![Figure 4.3 Pillar drilling machine design activity](image)
For example, to design a pillar drilling machine in collaborative design, the pillar drilling machine is divided into components such as head assembly, base assembly and pillar assembly. A different member of the design team on workstation design each of the components. On completion of the head assembly, base assembly and pillar assembly designs, the chief designer integrates them to form the final design of the pillar drilling machine. The design activity is represented in Figure 4.3.

The CAD industry agrees collaborative design and engineering based on increased sharing of information between different disciplines across geographical boundaries. The continuing development of object oriented design software and the growing use of internet is now allowing interoperability and data sharing to happen, leading to the growth of true collaborative environment throughout the whole design process. Information technology, which aims to improve collaborative designs, interoperability and the design, will make substantial differences in the success of a project including reduced costs and greatly improved productivity. This has led to the development of object based industry foundation classes, a series of standard software object libraries covering all aspects of the design department.

4.7 VERSION SUPPORTED COLLABORATIVE DESIGN

An emerging trend in product design is the de-centralization of the design team. Critical to this enterprise-engineering environment is the availability if the design information is through shared databases. Such access can also be significant during the redesign of a product or the original design of a similar product. The proposed system is by merging the processes of collaborating design and versioning to capture the evolution of the design process. The design of pillar drilling machine is taken as an example in this design activity because it supports group activity.
The primary challenge in sharing design information across functional workgroups lies in reducing the complex expression of association between design elements. Collaborative design systems have addressed this problem from the perception of formalizing a shared ontology or product model. The design model and ontology are an expression of the meaning of the design and provide a means by which information sharing in design may be achieved. However in many design cases, formalizing ontology before the design begins, establishes the knowledge sharing agreements or mapping out the design hierarchy. And it is potentially more expensive than the design itself. The association between the design and representation is formalized by basing the representation on terminological pattern in the design test.

4.7.1 Object Versioning in Collaborative Design

In version supported collaborative design, a new version of a component can be designed from the existing one. Each designer retrieves the component of the design from the database, makes changes and puts in the database as a new version of the component. In this collaborative design model, each component to be designed is represented in terms of the component class. The component class in turn may contain sub components.

```
Class component {
    attributes
    sub components
    methods
};
```

In order to capture the version derivation information for each component separately, component_version class is used. This component_version class contains design, non design and ancestor attributes.
class component_version
{
    design attributes;
    non design attributes;
    ancestors;
}

The constraints associated with each component are specified in the component_specification class. This component_specification class captures the dependent and independent relationship with the other component class.

Class component_specification
{
    attributes;
        specifications;
        dependent relations
    methods:
        message to other components;
        message from other components;
}

4.7.2 Design of Pillar Drilling Machine

Drilling machine is one of the simplest, moderate and accurate machine tools used in production shop and tool room. It consists of a spindle which imparts rotary motion to the drilling tool, a mechanism for feeding the tool into work, a table on which the work rests and a frame. Drilling is the process of making holes or enlarging a hole in an object by forcing a rotating tool called a drill. Drilling machines are classified according to the construction and work performed.
Portable drilling machine
Sensitive drilling machine
Upright drilling machine
Radial drilling machine
Gang drilling machine
Multi spindle drilling machine

Pillar drilling machines are specified in different sizes according to their drilling capacity such as 12mm, 25mm, 32mm and 50mm.

Pillar drilling machine consists of components like head assembly, pillar assembly, and base assembly. The various parameters associated with the design of pillar drilling machine are drilling capacity, spindle diameter, spindle length, squill inner diameter, squill outer diameter, drive pulley speed, driven pulley speed, motor speed, base length, base height, base width, area of force, pillar flange diameter, pillar diameter, pillar height, table diameter, table working force, table arm distance etc. The important design parameter of pillar drilling machine is drilling capacity, based on which the parameters of the various components are computed. Whenever the drilling capacity parameter is changed, new versions of various components have to be created. The changes to the non-design attributes such as motor speed lead to the equivalent version. If the drilling capacity of the machine is 12mm and the motor speed is 1500 Rotations Per Minute (RPM), the change of design attributes i.e. drilling capacity from 12mm to 25 mm leads to the creation of new design versions, whereas the change of non design attribute i.e. motor speed from 1500 RPM to 3000 RPM leads to equivalent design version.

The sample class representation of the head assembly of the pillar drilling machine is given as
Whenever the component associated with the class is modified, the dependent components receive the message from that component. The dependent components are modified according to the message received, so that the entire product is inconsistent state.

According to the proposed model, whenever a new version of product is required, the message is passed to individual members of the design team from the chief designer. After receiving the message, the individual designer retrieves his earlier component from the database and makes necessary modifications according to the new requirements and store this as a new version of the original retrieved component. In this case no message is passed between individual designers. If the chief designer needs a new version of a particular component then he sends a message only to the
particular designer who designs that component. After receiving the message, the designer creates a new version of that component. If this new version of the component affects other components then the message must be sent to the dependent component and versions of the dependent components are also created.

Let us assume that a pillar drilling machine of 12mm size has already been designed and available in the database. If the design for a pillar drilling machine of 25mm size is needed, then the individual design members retrieve their components designed for 12mm and modify it for 25mm. The modified components are stored as versions in the database. The version derived relationship is presented in Figure 4.4 The workstation framework model is shown in Figure 4.5.

Figure 4.4 Versions of pillar drilling machine
Figure 4.5 Object versioning workstation framework model

4.8 OBJECT VERSIONING IN CONFIGURATION MANAGEMENT

In version supported software configuration management, configuration details are represented in the form of objects. A new version of an object can be created from the existing one. The configuration details are stored in a flat file in the form of objects that consists of both essential and optional attributes. The modification of essential attributes causes the creation of a new version of an object. During software maintenance, the configuration details are retrieved and modified according to the local environment. After modification the details are put into the file as a new version of the configuration. In this proposed system the following classes are needed to manage the configuration details represented as objects and its versions.

Class configuration
{
    Attributes
    Methods;
}
In order to capture the version derivation information and constraints associated with each specification for each configuration, a separate configuration version class is used. This configuration version class contains event update method and attributes and captures the dependent and independent relationship with another class.

Class configuration_version
{
  Ancestor;
  Specifications;
  Dependent relations;
  Update ();
}

The focus is given to manage the XML software configuration file. During the software installation the configuration details stored in the file are fetched and modified according to the current environment. Software maintenance can be considered as a critical task. During this it is essential to keep track of changes made to the configurations. Some parameters are essential and other parameters are optional. Based on this the user has to specify the parameter values for essential elements and keep the default values for optional elements. The major steps in creating and managing versions of XML configuration data are

i. Input XML configuration file.

ii. Mapping the XML file to XML schema and XML schema to Java classes using Java Architecture for XML Binding (JAXB).
iii. All of the generated classes, source files, and application code must be compiled.

iv. Mapping the current XML data into objects.

v. Create versions based on the modifications of the object.

vi. Maintain the version history using the version graph.

Figure 4.6 shows the architecture model of the proposed object versioning system used in configuration management. The major components are Java Architecture for XML Binding (JAXB), User Interfaces, Object Version Module and Flat File System to store objects.

Figure 4.6 Object versioning system model in configuration management

4.9 PROTOTYPE MODEL OF OBJECT VERSIONING

The prototype implementation of the proposed object versioning system consists of following modules.

- Design Module
- Conversion Module
- Versioning Module
4.9.1 Design Module

Designers create the component design using Auto CAD. The component design is transformed into an MS Excel file using the procedure in the Autocad window,

- Select the "Insert" tab at the top of the screen.
- Locate the "Linking and Extraction" section, then click the "Extract Data" button. Save our .dwg file when prompted.
- Select the "Create a new data extraction" option, then click the "Next" button.
- Type a name for the extraction file, click the "Save" button, then click the "Next" button.
- Select the text that we wish to extract from the "Objects" column, then click the "Next" button twice.
- Adjust the layout of the extracted text, then click the "Next" button.
- Click the "Output data to external file" option, click the "Next" button, then click the "Finish" button.

The design module and the corresponding wheel object design using AutoCAD are shown in Figure 4.7 and 4.8 respectively. The sequence of steps followed for mapping an AutoCAD object into an Excel data file is shown in the Figures 4.9 A, 4.9 B, 4.9 C, 4.9 D, 4.9 E, 4.9 F and 4.9 G respectively.
Figure 4.7 Design module

Figure 4.8 Design of wheel object
Figure 4.9 Mapping autocad object to excel data
Figure 4.9 (Continued)
Figure 4.9 (Continued)
4.9.2 Conversion Module

This conversion module transforms the excel data into Java object using a Java package com.excel2objects.common.excel.engine. The mapped objects are stored in a file system for persistence. The mapping process is shown in Figure 4.10. Figure 4.11 shows the values wheel object attributes.
Figure 4.10 Conversion module

Figure 4.11 Wheel object data
4.9.3 Object Versioning Module

The task of object version module involves creation and maintenance of Java objects. Whenever critical attributes are changed, the existing object/document is replaced with the new one. The existing / created objects store it in a file for repository. Finally java objects are converted into XML file.
Figure 4.13 Object versioning module

Figure 4.14 Object versioning interface
Figure 4.15 Object versioning output

Figure 4.16 XML representation of java object
4.9.4 Marshalling and Unmarshalling

XML and Java technology are recognized as ideal building blocks for developing Web services and applications that access Web services. A new Java API called Java Architecture for XML Binding (JAXB) can make it easier to access XML documents from applications written in the Java programming language.

JAXB allows Java developers to map Java classes to XML representations. JAXB provides two main features: the ability to marshal Java objects into XML and the inverse, i.e. to unmarshal XML back into Java objects. In other words, JAXB allows storing and retrieving data in the memory in any XML format, without the need to implement a specific set of XML loading and saving routines for the program's class structure. It is similar to xsd.exe and XmlSerializer in the .NET Framework. JAXB is particularly useful when the specification is complex and changing. In such a case, regularly changing the XML Schema definitions to keep them synchronized with the Java definitions can be time consuming and error prone.

Unmarshalling is the process of transforming the storage or transmission representation of an object back into its original memory representation. It is the inverse of marshalling. Unmarshalling an XML document means creating a tree of content objects that represents the content and organization of the document. The content tree is not a DOM-based tree. In fact, content trees produced through JAXB can be more efficient in terms of memory use than DOM-based trees.

A JAXBContext object has been created. This object provides the entry point to the JAXB API. When the object is created a context path needs to be specified.
This is a list of one or more package names that contain interfaces generated by the binding compiler. By allowing multiple package names in the context path, JAXB allows us to unmarshall a combination of XML data elements that correspond to different schemas.

jaxbMarshall.unmarshal () is used to convert XML content into java object,

The wheel object for the XML file consists of,

- Noarms: 16
- Inrad: 546
- Outrad: 730
- Centerx: 3069
- Centery: 2092
- Tposx: 1355
- Tposy: 3602
- Tcolor: blue
- Wcolor: blue
Marshalling is the process of transforming the memory representation of an object to a data format suitable for storage or transmission, and it is typically used when data must be moved between different parts of a computer program or from one program to another. Marshalling is similar to serialization and is used to communicate to remote objects with an object and in this case a serialized object. It simplifies complex communication, using custom/complex objects to communicate instead of primitives. The opposite, or reverse, of marshalling is called unmarshalling (or demarshalling, similar to deserialization).

jaxbMarshall.marshal () is used to convert the java object into XML file,

The XML file for wheel object,

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wheelob tcolor="blue" tposx="1355" tposy="3602" wcolor="blue">
    <noarms>16</noarms>
    <centerx>3069</centerx>
    <centery>2092</centery>
    <inrad>546</inrad>
    <outrad>730</outrad>
</wheelob>
```

JAXB simplifies access to an XML document from a Java program by presenting the XML document to the program in a Java format. The first step in this process is to bind the schema for the XML document into a set of Java classes that represents the schema.
A Schema is an XML specification that governs the allowable components of an XML document and the relationships between the components. For example, a schema identifies the elements that can appear in an XML document, in what order they must appear, what attributes they can have, and which elements are subordinate (that is, are child elements) to other elements. An XML document does not have to have a schema, but if it does, it must conform to that schema to be a valid XML document. JAXB requires that the XML document to be accessed has a schema, and that schema is written in the W3C XML Schema Language.

Wheel.xml document has a schema that is written in the W3C XML Schema Language. This schema defines a <Wheelob> as an element that has a complex type. This means that it has child elements,

- <noarms>
- <centerx>
- <centery>
- <inrad>
- <outrad>

4.9.5 Use-case Model

The Unified Modeling Language is a modeling language used in object oriented system development. Use-case diagram used for object versioning system prototype model is shown in Figure 4.18. The class diagram for the wheel object and object version module is shown in Figure 4.19.
Figure 4.18 Use-case diagram

Figure 4.19 Class diagram
4.10 CONCLUSION

Customers are demanding better products, services and prices. This is particularly true in the highly fragmented design and engineering industry and software development where there are inefficiencies in a data sharing among design teams and development teams throughout the different industry disciplines. The collaborative design and object oriented technology, which not only meets the needs of its customer, it also reduces the costs. The proposed model simplifies the task of managing various versions of design details and configuration details in the form of XML data. This model takes a single XML file and manages the versions of those files as objects. The prototype model of the proposed system is being developed in JAVA. The proposed model can be further enhanced by the inclusion of multiple configuration and files and their relations.

The next chapter discusses in detail the performance analysis of public-key encryption algorithm used in XML data security requirements.