4.0 Object-Oriented gateway to CIS:

4.1 Introduction:

Data about spatial objects (spatial and non-spatial data) is stored in spatial database, which is the central element of CIS. Spatial database must be an useful abstraction of reality and it must support operators for the management and analysis of database. The data in a GIS comes from various sources viz. satellites, aerial photographs, ground observation etc.. New applications require customised data types and interfaces which the conventional GIS is lacking, because they are static and inflexible. Present day applications uses complex structured geometric entities, which inherit features from other entities also. So, need has been felt to develop a system which support complex spatial objects. Hence, modelling involves structural objects.

Due to various reasons stated , need has been felt to develop an object-oriented GIS, which uses class hierarchy and structural object orientation which provide efficient means to model complex geographical objects. Behavioral object orientation feature in object-oriented data model allows user to create specific data types and operators for a particular application. Object-Oriented data model (OODM) permitts spatial and non-spatial characteristics of one class of feature to be inherited by related feature types. All spatial and non-spatial data related to a class of feature are stored together using object-oriented approach, which will help in creation of complex data models.
CIS user is interested to access data (i.e., external aspects) irrespective of its internal representation which is hidden from the user. To improve performance, fix a bug etc., it is essential to change the implementation of spatial object. The feature of encapsulation (information hiding) of object-oriented data model can be made use of in developing a CIS. Need has been felt to call a variety of functions i.e., to have specific behavior at run time using same interface. This can't be achieved in traditional CIS. So, feature of polymorphism of object-oriented data model can be used in CIS.

The various sophisticated tools available with object-oriented data model has motivated in developing an object-oriented CIS, which will enable in creating complex real world spatial phenomena with ease. As, developing new CIS using object-oriented concepts is costly, an object-oriented frontend (which is application independent) to existing CIS is designed. In order to facilitate system to be user friendly, graphical user interface (GUI) is to be provided, and this is also incorporated. So, object-oriented gateway to CIS consists of object-oriented frontend to CIS, and GUI modules.

4.2 Theoretical basis:

The terms/features used are briefly explained below [16],

4.2.1 Object:

It is a convenient collection of data that represents a meaningful entity in application. It represents an individual, identifiable item, unit or entity, either real or abstract, with a
well defined role in the problem domain. Objects serve two purposes. One, they promote understanding of the real world and two, they provide a practical basis for computer implementation. Objects may be any thing existing, like book, person, polygon, forest etc., and not existing physically but well defined, such as symbol table, binary tree, a chemical equation. An object has a state, behavior, and identity in object-oriented data model. The structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable.

The state of an object encompasses all the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties. A property is one inherent or distinctive characteristic, quality or feature that contributes to making an object uniquely that object. All properties have some value. This value might be a simple quality or it might denote another object. Behavior describes how an object acts and reacts in terms of its state changes and message passing. The behavior of an object is completely defined by its actions.

4.2.2 Object identity:

Each object should have a unique identity to denote the object independently of its behavior or state. This implies that all objects have identity and are distinguishable by their existence and not by descriptive properties that may have. Most programming and database languages use variable names to distinguish temporary objects. Most database systems use identifier keys (primary keys) to distinguish persistent objects, mixing data value and identity.
4.2.3 Class:

An object class describes a group of objects with similar properties (attributes), common behavior (operation), common relationships to other objects, and common semantics. Person, cycle and process are all object classes. The abbreviation class is often used instead of object class. Most objects derive their individuality from differences in their attribute values and relationships to other objects. However, objects with identical attribute values and relationships are possible.

We can abstract a problem by grouping objects into classes. Common definitions such as class name and attribute names are stored once per class. All objects of a class can benefit from the code reuse by writing all operations once for each class.

4.2.4 Complex object:

It is used to represent objects that are composed of other objects as parts (components) and/or containing other objects as values/reference to their attributes. Two types of reference semantics exist between a complex object and its components at different levels. The first type, which is called 'ownership semantics' applies when the sub-objects of a complex object are encapsulated within the complex objects and are hence considered part of the complex object. This is referred to as the 'is-part-of' or 'is-component-of' relationship. The second type which we call 'reference semantics' applies when components of the complex object are independent objects but sometimes may be considered as part of complex object. This is called
'is-associated-with' relationship, since it describes an equal association between two independent objects. The desire of current database users is to abstract these complex objects without disturbing the structure (in natural way),

4.2.5 Encapsulation:

Encapsulation (also known as information hiding) consists of separating external aspects of an object, which are accessible to other objects, from the internal implementation details of the object, which are hidden from others. Encapsulation prevents a program from being so interdependent that a small change has massive ripple effects. The implementation of an object can be changed without affecting the applications that use it. One may want to change the implementation of an object to improve performance, fix a bug, consolidate code, or for porting.

Encapsulation is not unique to object-oriented languages, but the ability to combine data structure and behavior in a single entity makes encapsulation clearer or more powerful than in conventional languages they separate data structure and behavior.

4.2.6 Hierarchy and inheritance:

Another important characteristic of object-oriented system is that they allow type or class hierarchies and inheritance. These two are interrelated. Type hierarchies and class hierarchies are conceptually different, but because in the end result they often lead to mean the same thing. Hierarchy is often explained in terms of class. Classes may form a hierarchy (with super class -
sub class relationship) such that objects in any given class are automatically members of its super class. All methods and attributes that apply to a class, apply to its sub classes as well and this is called inheritance.

Inheritance is the sharing of attributes and operations among classes based on a hierarchical relationship. Each sub class incorporates, or inherits, all of the properties of its super class and adds its own unique properties. Class hierarchy is the relationship among classes where one class shares the structure or behavior defined in one (single inheritance) or more (multiple inheritance) other classes. This inheritance defines a kind of hierarchy among classes in which a sub class inherits from one or more super classes, a sub class typically augments or redefines the existing structure and behavior of its super classes.

An object class a inherits the attributes and methods of an ancestor class b if and only if they do not conflict with the attributes and methods that have been defined for a directly or for any ancestor class of c that lies between a and b.

4.2.7 Binding:

Binding is the act of associating name to a type. It is generally seen in terms of time when the action takes place. Static binding or early binding means that the behavior of an object is fixed at the time of compilation. Dynamic binding or late binding means that the actual behavior of an object is known only at run time. So, binding is not made until the object designated by name is created.
4.2.8 Polymorphism:

The binding of specific behavior, among the different implementations, at run time is known as polymorphism. It is also known as overloading. It can be facilitated to both objects and operators. For objects, it represents a concept where a single name may denote objects of many different classes, that are related by some common super class. It can be termed as object polymorphism. For operators, termed as operation polymorphism, same operator name or symbol (i.e., an operator or a function) is bound to more than one different implementations, depending on the type of object or the type of the parameters passed. Polymorphism is the result of interaction of inheritance and dynamic binding. It is one of the most powerful features of object-oriented programming (OOP) and distinguishes OOP from traditional programming.

4.2.9 Extensibility:

It assembles the pre written modules and classes into new tailor made systems. The technological improvements can be quickly incorporated into the reusable modules and thus the system can always remain up to date. The development of new systems become rapid and economical. New techniques can be evaluated without making significant modifications.

4.2.10 Object-Oriented DBMS [20] [52];

This is a DBMS with an object-oriented data model (OODM). object-oriented database management system (OODB) apart from
supporting the object-oriented features explained above (4.2.1 to 4.2.9) should also provide DBMS features such as persistence, secondary storage management, transaction management, concurrency control, recovery and query along with structural object-orientation (support of composite objects) and support user defined data types.

4.3 Need for object-oriented CIS (OOGIS) [41]:

The conventional GISs are static and inflexible with regard to new applications that may require customised data types and interfaces. Geometric entities which are complex structured in nature, posses inherited features of other entities. Also there is a need to support complex objects. Modelling of structured objects, adoption of emerging standards and integration of efficient access methods for large spatial databases are falling into the challenging areas in development of any GIS. In conjunction with RDBMS, proprietary GIS databases are also to be used which resulted to use OODBMS for GIS databases [31].

So, keeping this in view, in object-oriented GIS, class hierarchy and structural object orientation provide efficient means to model complex geographical objects. In addition, behavioral object orientation (support of user defined types and operators) allows to facilitate support of specific data types and operators for particular application. Moreover the object-oriented approach implements a system where all spatial and non-spatial data relating to a class of feature are stored together permitting the creation of complex data models. OODB permits spatial and
non-spatial characteristics of one class of feature to be inherited by related feature types. Object-Oriented approach also encapsulates all applicable operations with in each class of features so that operations on the feature are manifested equally in the spatial and attribute data. All topological relationships can be stored for each instance. Object-Oriented approach is having clear advantage in comparison with the fixed set of data types and operators that is typical for most commercial GISs that exist. So, using object-oriented database design which offers most sophisticated tools, real world models of spatial phenomena in a CIS can be created with ease at present.

4.4 Object-Oriented GIS (OOGIS) vs CIS:

Following are the comparisons between GIS and OOGIS.

4.4.1 Generic concepts:

The generic concepts of an object-oriented database design are object, classification, relationship, inheritance relationship and aggregation. Inheritance relationships among the various objects are embedded components of geographic entities. Complex real world entities can not fit in relational database, so artificial decomposition into several relational tables becomes necessary with common attributes (foreign keys) for linking. By decomposing efficiency is decreased and there is a possibility of losing the semantics associated with data. But in OOGIS, the user is able to manipulate the complex objects as if they were single units. So, the generic types mentioned above help the user to model/abstract the complex entities in natural way. In traditional
CIS, a complex object must be decomposed over different relations.

4.4.2 Adhoc query facility:

Adhoc query facility is not possible with CIS, but provide menu driven queries and restricted class of queries. Programming language interface to CIS provide user to query in adhoc manner and interactive interface to CIS is best used for pointing a geographic object on the display and put query centered on that object. CIS can offer more flexibility when it can handle searches both in spatial and object centered ways.

Object extensions to SQL (O-SQL) are proposed in TIGRIS object-oriented data model for CIS [18]. These extensions include:

The ability to invoke any method on any object directly from the query language for use either in predicates in selects, in relational operators or for updates.

The ability to define class sensitive language macros that interpret query statements in accordance to class context (a query over loading operators)

The ability to group query statements together in single units of execution, with inter query communication via temporary relations managed by the system.

4.4.3 Concurrency:

Most of the applications require operating in multi user environment. Database systems usually provide mechanisms to facilitate controlled sharing of data and resources. This controlled sharing means that certain data items can only be read
or updated by a selected group of users. Only authorised users who have ownership permissions are allowed to change the data. Due to data encapsulation and multiple inheritance features of OOGIS, the situation is less complex to support controlled sharing.

4.4.4 Distribution:

Large amount of data from different sources is common in GIS applications. The performance of the system can be improved if the data can be stored in different locations without affecting the integrity of the database. Without losing the integrity and consistency the updates/retrivals are to be performed and the distribution of data is transparent to the user. Distributed data management has made significant progress since the introduction of relational model and object-oriented models. Traditional GIS data models, on the other hand, do not lend themselves easily to the distributed data management. Also complex geographical objects and user defined data types will be supported in more natural way in OOGIS.

4.5 Need for cooperative environment [2]:

For any application of GIS, data is generated by multiple sources (digitized, satellites, ground observation, weather station, aerial photographs etc..) and this data is accessed, processed and transformed by many users and available for use to other users also. Lack of coordination among all these different users of data may render large amounts of work useless.
Activities related to geographic applications involve modelling geographic phenomena (source of modelling reality). Modelling takes place by applying successive transformations to some input data, which results in creation of derived data. This derived data can not be interpreted correctly (probably by experts in other domains who may have to use this derived data) without the knowledge about how data was created and global model of which it is part. Furthermore, the complex models used require the collaboration of several people/experts and they may have to use the model which is designed by others. One more problem added to current situation is the model itself may undergo changes as more knowledge and precise data becomes available in future. Hence, there is need for considering cooperative environment in the design of more sophisticated CIS.

The main aspect of cooperative modelling is that, multiple agents interact with each other to create complex structures/models. Another aspect is temporal cooperation, the data sets may be used and accessed years after they are created. This raises the issue of defining cooperation between actions executed may be years apart. Existing CIS do not provide any support for the cooperative work. So, there is a need to facilitate cooperative environment, and still allowing users to take advantage of CIS.

4.6 Need for graphical user interface (GUI):

The performance and efficiency of any system can be looked and felt by user, only when proper interface is provided. In case
of information systems, interface in interactive manner or through programming languages avail user to build their application and to perform queries. Graphical user interface (GUI) is, no doubt, so user friendly and becomes must in case of CIS. The spatial analysis system involves large volumes of data, display of geographical objects like maps necessitates the option of GUI.

Main use of GIS is performing spatial queries in different ways before coming to conclusion for any decision making application. To support spatial query, GUI is one not to leave. So, GUI is also an important issue of consideration for the success of OOGIS.

4.7 Aim of object-oriented gateway to GIS:

It is much better if existing GIS is converted to OOGIS, as it is very costly to develop new OOGIS.

So, the aim of object-oriented gateway to GIS is two fold, one is to develop an object-oriented frontend to existing GIS, such that the flavor of object-orientation and the power of handling spatial data/objects of GIS are blended together to evolve a fully functional, flexible, application independent, user friendly GIS. Secondly, to provide GUI, schema building facility, automated change propagation, and version control in existing GIS. It is also possible to achieve cooperation between users working in two different underlying GIS, developing two sub models of a global application.
4.8 Object-Oriented data model (OODM):

Data model can be seen as a collection of conceptual tools for describing data relationships, data semantics, consistency constraints and operations. A data model describes on the abstract level objects and their behavior. Data structure is a specific implementation of data model and it fixes performance aspects such as storage utilisation and response time. Object-Oriented data models are proposed for taking care of characteristics which differ from those of traditional business applications such as complex data structures, longer duration transactions, newer data types for storing images or large textual items and need to define non-standard application specific operations for applications like CAD/CAM, image and graphic databases, CIS and multimedia database.

Fig 4.1 below shows the concepts drawn from different areas towards the development of an object-oriented data model (OODM).
As shown in Fig 4.1, various concepts from traditional database systems, semantic data models, object-oriented programming are put together to form an object-oriented data model. The database features viz., persistence, sharing etc., are required for any data model. The object-oriented data model originates from the tradition of semantic data modelling, but it takes view of data that is closely aligned with object-oriented programming languages.

The entities in an object-oriented data models uses single modelling concepts: the objects. The Object-Oriented paradigm is based on five fundamental concepts [4].

1. Object is used to model every real world entity and is associated with a unique identifier.
2. Each object has a set of instance attributes (instance variables) and methods. The value of an attribute can be an object or a set of objects. This characteristic permitts arbitrarily complex objects to be defined as an aggregation of other objects. The set of attributes of an object and set of methods represents the object structure and behavior respectively.
3. Object's status is represented by the attribute values and can be accessed or modified by sending messages to the objects to invoke the corresponding methods. Methods define the operations to manipulate or return state of an object. Objects can also communicate with one another by sending messages. For each message understood by an object, there is a corresponding method that executes the message.
4. Objects sharing the same structure and behavior are grouped into classes. A class represents a template for a set of similar objects.

5. A class can be defined as a specialisation of one or more classes. A class defined as a specialisation is called a sub class and inherits attributes and methods from its super class.

Aggregation and generalisation mechanisms of semantic data models in addition let the user represent relationships among objects and among object collections.

4.8.1 Semantic data models:

The OODM uses semantic data models to represent complex objects because traditional relational models are inadequate such as limited expressive power (semantic content) and number of problems can not be naturally expressible in terms of relations. Spatial systems are a case where the limitations become clear [53]. In a typical spatial system a polygon is decomposed into rings, which are again decomposed into set of chains. The polygon will have attributes polygon ID, chain ID, ring ID, ring seq. Ring will have attributes ring ID, chain ID, chain seq. The attributes of chain are chain ID, start node, end node, left polygon, right polygon and attributes of node are node ID, point ID.

The attributes of point are point ID, x coordinate and y coordinate. Each of these is a relation. This model of polygon as a set of relations, though complete, is low level and depends on user's capability of decomposing the complex objects or entities into relations. Semantic models aim to provide more facilities for
the representation of the user's view of systems. These models (ex: entity-relationship model, functional data model, semantic data model) as well, decouple these representations from the physical implementation of the databases. These provide more powerful abstractions for database schema than the relational, hierarchical, and network models can support. Aggregation, generalisation, specialisation and association are various abstraction constructs with the first two being proposed by Smith & Smith (1977).

4.8.2 Aggregation and generalisation:

Aggregation refers to an abstraction in which a relationship between objects is regarded as a high level object. Aggregates are abstract entities that contain heterogeneous components. In relational terms, an aggregated tuple has attributes that are themselves tuples of different relations. This relationship can be nested to any depth. Aggregation in one way of representing hierarchical structure among tuples of different relations. The classes or types are constructed by attributes or components and there is likely same attribute belongs to two classes but corresponding objects do not share .the instance of common attribute.

Generalisation refers to an abstraction which enables a class of individual objects to be thought of generally as a single named object. Elements of a class can be specialised and grouped into sub classes. A sub class inherits all the properties of it's parent class. Elements of a sub class also belong to their parent
class. This sub class mechanism is called generalisation. Generalisation views a set of objects or a set of types/classes as one generic type/class. In case of objects we call it as 'classification' (i.e., 'class' in OODM) and in case of class we call it as 'type-type' generalisation or simply generalisation. All properties of the generalised type can be inherited downward to the constituent types. The arrows indicate the direction of generalisation.

![Diagram of generalisation hierarchy](image)

Fig 4.2 Generalisation hierarchy

In fig 4.2, employee and student together constitute higher level type person. Student is a generalisation of CIS and Physics. The attributes of person are inherited by each constituent (e.g., employee and student). These abstractions can be modelled in OODB with the concepts hierarchy and inheritance.

4.8.3 Class hierarchy and inheritance:

The class hierarchy specifies the 'is-a' relationship among classes. If two classes $c_1$, $c_2$ are related with 'is-a' relation, i.e $c_1$ 'is-a' $c_2$ then all properties (including attributes and methods) defined on $c_2$ will also be defined on $c_1$. $c_1$ is called a
sub class of $c_2$ and $c_2$ is a super class of $c_1$. The inheritance concept says that the properties specified for a class are inherited by all it's sub classes in the hierarchy. This is similar to the generalisation construct in semantic data modelling.

### 4.8.4 Class lattice and multiple inheritance:

If a class can have only one super class, the class hierarchy forms a tree. Sometimes, it is useful for a class to have multiple super classes. This generalises the class hierarchy tree to a directed acyclic graph (DAG), simply called a lattice. In a class lattice, a class can inherit properties from multiple super classes. This feature is known as multiple inheritance.

### 4.8.5 Additional features of OODM:

The following additional features, an OODM can support in addition to the basic concepts.

#### 4.8.5.1 Composite object [21]:

Many applications find it useful to provide an 'is-part-of' relation between an object and the object it references in addition to 'is-a' relationship. This is similar to aggregation concept in semantic data modelling. A composite object can be defined as an object with a hierarchy of exclusive objects that form a tree structure. However, it can also form a DAG structure, if component objects can be shared. The composite objects are often treated as units in data storage, retrieval, and integrity enforcement.
4.8.5.2 Version control:

Version control captures the record of evolution for data objects. Users in CAD/CAM, CIS etc., environments often desirous to work with multiple versions of an object, before selecting a right one that satisfies as their requirements. An object version can have only one previous version. Two properties, previous version, and next version, may be used to express the appropriate temporal relationships among object versions. Next version property can be multi valued, thereby allowing a given object version to have multiple successors and this version history generally forms a tree structure.

4.8.5.3 Equivalent objects:

In some applications such as design databases, an object can have different representations that are equivalent, the purpose of which is to impose constraints on the databases. OODM uses generic objects to represent the semantics of equivalent objects explicitly. A generic object contains attributes that identify different representations of the same object. Modifications in one representation are reflected in others by adding constraints to the definition of generic objects.

4.8.6 Data operations in OODM:

So far we have seen conceptual schema part of OODM, i.e., how the real world is represented as objects and relationship between these objects. The second part 'data operation' in OODM include schema definition, database creation, data retrieval and data update.
The schema definition operations are high level operators/used to describe application schema structure, various class definitions and relationship among the classes (class hierarchy). In CIS, CAD/CAM etc., applications the requirements of the user, source data, environment changes with time, hence the definition of schema is liable to be changed, which generally does not change in business applications. In order to modify the schema definition the operators similar to 'change class', 'add class', 'delete class', and 'change class hierarchy' are to be provided.

The operations for creating database have the form,
Create object (object name, object definition) or
Add object (object name, object definition), are used to input the data in the required form (structuring the data).

The operation for data retrieval is of the form
retrieve (object name, list of attributes, list of conditions)'
'List of attributes' specify the attribute names of interest specified as [all], [all but atbl], [atb3, atb4] etc.
'List-of-conditions' has the form [atbl logical operator value], [default] etc..

The operation of data update is of the form
update (object name, object value).

All operations stated above can be performed by making use of an interface language.

4.8.7 Object-Oriented databases:
An object-oriented database system (OODB) is a DBMS with an object-oriented data model (OODM), and the key feature of OODB is
the power they give the designer to specify both the structure of complex objects and the various operations which can be applied to these objects. Developers have selected one of the approaches mentioned here under:

Extending a relational system to support the concepts of objects

Extending an object-oriented programming language to include persistence, sharing etc., database features.

Many OODBs have been developed to meet the requirements of various applications viz., CAD/CAM, CASE, document processing etc., by making use of one of the above approaches. Knowledgebase representation schemes are to be incorporated in OODB to support AI applications. Orion developed by Mcc and Iris developed by HP will fall into the category of extending a relational system. Gemstone (by servo logic), Encore (by Brown univ), and object store (by object design) are the OODBs as an extension of object-oriented programming languages. The first one was developed using 'small talk' and next two were built on C. Although many OODBs have been proposed using these approaches, they differ in their interpretation of OODM. All OODBs (proposed/developed) support the concept of complex objects and object encapsulation at different levels. They can be categorised on the level of object orientation.

4.8.8 Structural object-orientation:

A data model supporting the construction of composite objects is called structurally object-oriented. So the object (tuple) do
not have to be atomic (as in the case of relational model), but may be composite in turn. A system that supports composite objects must provide specific operators, such as, duplication and deletion of objects with their sub structures, or navigation through an object structure. This is sometimes called operationally object orientation. It should provide atomic types (eg: integer, character) and certain type constructs (eg: tuple, set, list) which will enable users to construct application specific complex data structures. Structural object orientation can be used in a natural way to model built in objects (i.e., objects embedded in other objects in the sense of part of hierarchy) or to specify shared objects which are included in several other objects simultaneously. 'Class' construct is generally used to attain structural object-orientation in OODBs. To maintain reference between the objects and component objects object ID plays a vital role.

4.8.9 Behavior object-orientation:

A data model is called behaviorally object-oriented if it supports user defined types and the definition of operators (methods) that are applied to these types. Once the user defined type and its associated operators are specified, this new type can be used like a system defined type. Operators or methods are defined by an interface (name, input parameters, output parameters) and a program to compute it. The OODM will take care off operators not applying to inappropriate objects. To, apply an operator, one only needs to know its interface; no knowledge is required about its implementation.
In order to increase portability of an application, it should be generic and independent of the specific hardware configuration irrespective of the way the data model is defined and the success of object-oriented system design.

4.8.10 Modelling real world phenomena [11]:

The empirically real world verifiable facts referring geographical reality which constitutes a spatial information system is to be modelled. This data model is a limited representation of reality since the facts may not be certain, and is constrained by the finite, discrete nature of computing devices. The spatial databases representing real world phenomena is composed with logical units, spatially referenced entities must be abstracted, generalised or approximated in the process of creating the database. In spatial databases, modelling plays a major role and controls the view of the world which the user ultimately receives.

From the user's point of view, it is suitable to start with the real world. The world consists of fully defined and incompletely defined entities/objects. The incompletely defined spatial objects set is the source of the problem and includes spatial objects from different sources depending on the needs of any user. Euclidean geometry to reason about spatial arrangements and network topology when plan a trip or navigate any automobile are the mostly used methods to conceptualise space by which most of the applications can be addressed. It is not that reality changes, but the concepts used to structure perception of the
situation differs. In order to cope with the complexity of the realisation, we have to abstract details and concentrate on the aspects that are important for the task at hand.

The environment where the nature of the variations, the objects that can be identified and the identification may vary with the discipline, spatial scale and personal opinion gives few problems for exact models of real world phenomena or reality. The better model for complex real world objects is to regard them as some type of complex continua and because they can not be seen in their entirely, they can only be described by sampling and reconstructed by interpolation. A CIS will be successful only if it can present the user, an accurate view of the world, and to do so requires efficient access to a database, and the use of accurate data models.

4.9 Different approaches to OOGIS:

The object-oriented term has received attention recently in the CIS literature as many of the computer science concepts of object-oriented programming and databases have stimulated discussions in the spatial context. The object-oriented notion of object identity is clearly more compatible with the object view of reality than the field view, and the systems currently being marked as object-oriented, rather than layered, seem to be aimed at those applications in which the object view is more acceptable.

Following are different approaches to OOGIS, for which some prototypes have been developed as an attempt to use
object-oriented model in CIS technology. These approaches can also be used, to develop a non object-oriented CIS model.

1. The extension of OODB functionality to existing CIS:

   In this approach capture, manipulation, analysis, presentation capabilities of the spatial data are availed by the CIS, and the storage, management and other object-oriented features are to be enhanced over the existing CIS. One example in this line is the commercial CIS SICAD of Siemens NIXDROF with its data management component GDB, offers numerous database functionality.

2. The extensions of an existing DBMS by geometric and geographical functionality: For this approach, OODB seems to be an interesting option. A kernel consisting of geographical and geometric building blocks can be built on underlying OODB as base classes. These building blocks are classes and methods for representation and management of spatial information. Other CIS functionality can easily be build with the help of kernel. The first prototypes have been developed in this line are based on O2 and Postgres Godot is the OOGIS developed based on this paradigm.

3. The coupling of GIS with existing DBMS, is a hybrid approach, in which some or all geographical information are stored in specialised structures, and attribute information is stored in DBMS.
4.10 Approach followed for OOGIS:

As explained already, there are many advantages if CIS is developed on object-oriented concepts. The following are steps followed in achieving this goal.

4.10.1 An object-oriented frontend to any CIS:

This is an extension to an existing GIS. In this there is a clear separation between the underlying GIS and the extension part and this separation plays an important or vital role in achieving the portability, extensibility and (cooperative) modelling capability.

![Block diagram of object-oriented gateway to GIS](image)

Fig 4.3 Block diagram of object-oriented gateway to GIS

For the functionality, such as spatial data management, data storage and spatial query processing, we have to rely on underlying GIS. Object-Oriented frontend implements the modelling capabilities, the underlying system lacks. The underlying GIS can be heterogeneous GISs developed in different approaches. Portability can be achieved, as the frontend is the upper layer
for different CIS. One has to rewrite the parser, which is responsible for communication between frontend part and underlying GIS. This frontend is not restricted to an interface layer between/ it allows users to model complex objects and for building schema. The frontend will augment the underlying GIS with features like, object identity, structural object orientation, and behavioral object orientation, which in turn may perform multiple transactions or multiple calls to underlying GIS. To attain extensibility, inheritance and reusability are helpful. In order to facilitate cooperative modelling, the frontend supports version control, multiple views of modelling and automated change propagation.

The system may be a tool, with which users abstract and model the data of their interest or the system is the outcome of modelling of tools which manipulate the data of user's interest. The information system developed in first approach gives overall modelling authority to the user, thus it is flexible and support varities of applications, where as the one developed with, later approach suffers from certain limitations such as inflexibility, static and gives narrow view to the user, but it may be simple to use, because user merely follows the existing model for which tools are provided.

User has to be tuned to model or some tools it supports to use an information system, which is developed in latter approach. User finds greater difficulty when his need, application context, or data changes. User can not build any schema for application. The user has to build his own schema for each application, which
is a difficult task. The limitation can be overcome by providing an easy environment to define the schema and GUI. It reduces much of the key in work as user need not deliver many commands to build schema. Thus the former is superior to later.

Here, the main effect is to convert an already developed GIS without facilities for creating schema into a GIS which is flexible and allows the user to design his own model of the real world phenomena or geographical reality. The GUI makes use of graphics on a bit mapped video display. Graphics provides better utilisation real estate, a visually rich environment for conveying information, and the possibility of what you see is what you get video display of graphics and formatted text prepared for a printed document. In GUI, the video display, no longer confined to echo text, but becomes a source of input showing various graphical objects in the form of icons and input devices such as buttons and scroll bars. The user can directly manipulate these objects on the screen using a pointing device mouse or keyboard. Graphic objects can be dragged, buttons can be pushed, and scroll bars can be scrolled. The user can directly interact with objects on the display, rather than one way cycle of information from the keyboard to the program and to the video display.

4.11 Design:

4.11.1 Introduction:

The object-oriented approach views a system as a set of objects, each object having a well defined operations, and a
transformation function that transforms the objects by performing operations on the objects.

The object-oriented design methodology consists of three parts:

Define the problem

Develop an informal strategy

Formalise the strategy

The third step has, further, four phases

1. Identify the objects and their attributes
2. Identify the operations on the objects
3. Establish an interface and
4. Implement the operations

These three steps can be merged and then divided into four stages, which follow software engineering life cycle. The soul of the design will be found in the control concerns of object-oriented design. The stages are

Analysis

System design

Object design

Implementation

This object-oriented methodology is proposed by Rumbaugh et al. The methodology consists of building a model of an application domain and then adding implementation details to it during the design of the system. They called this approach as object modelling technique (OMT). Since the design follows the same approach as OMT, it is necessary to brief about this methodology.
1. **Analysis:**

It is concerned with understanding and modelling the application and the domain within which it operates. The conceptual view of the proposed system is arrived based on the initial input to this phase: the problem statement. In this stage model will be built of the real world situation showing its properties. The analysis model is a coarse abstraction of what the desired system must do, not how it will be done. The objects in the model should be application domain concepts and not computer implementation concepts such as data structures. The output from analysis is a formal model that describes the objects and their relationships, the dynamic flow of control, and the functional transformation of data subject to constraint.

2. **System Design:**

The overall structure is determined in this phase. During system design, the target system organised into sub systems based on both analysis structure and proposed architecture. The system designer must decide what performance characteristics to optimize, choose strategy of attacking the problem, and make tentative resource allocations.

3. **Object design:**

During this phase, design model is built based on analysis model but containing implementation details. The designer adds details in accordance with the strategy established during systems design. The focus of object design is data structures and algorithms needed to implement each class. Both application domain objects and the computer domain objects are described
using the same object-oriented concepts. During object design, practical designs are produced by elaborating, refining and then optimizing the analysis models. During the object design stage the shift in emphasis is from application concepts to computer concepts. The basic algorithm is chosen first to implement each major function of the system and object model is optimized for efficient implementation and this model structure is then optimized for efficient implementation.

4 Implementation:

The object classes and relationships developed during object design are finally translated into a particular programming language, database, or hardware implementation. Programming should be a relatively minor because all the hard decisions should be made during design. In this implementation phase, it is important to follow good software engineering practice so that the design is straightforward, and the implemented system remains flexible and extensible. Some initial implementation was carried-out in [56],

Through the system development cycle i.e. analysis through design to implementation, object-oriented concepts can be applied. In all the above phases, the OMT methodology uses three kinds of models to describe the system:

a) the object model, describing static structure of the objects, in a system, and their relationships

b) the dynamic models describing the interactions among the objects of the system and
c) the functional model, describing the data transformation of the system.

The three models are orthogonal part of the description of a complete system and are cross linked and each model is amplified and acquires implementation details as development progress.

In functional decomposition technique which is most direct way of implementation, consisting of dividing the system into subprograms, transferring control between sub programs concentrate upon the algorithmic abstractions. The nature of the abstractions that may be conveniently achieved through the use of subroutines is limited.

The functional development methods suffer from the following limitations:

- do not effectively address data abstractions and information hiding;
- generally inadequate for problem domains with natural concurrency;

often not responsive to problem changes in space and hence the system can be fragile.

Massive restructuring by decomposing functionally is required if the requirement changes. In object-oriented approach focus is on identifying objects first from the application domain and then on filling procedures around them.
4.12 Analysis:

Analysis of the system, starts with the problem statement, which is to develop an object-oriented frontend to an existing GIS in order to enhance it as fully functional, flexible, application independent, user friendly GIS. Providing GUI is also part of the problem statement under consideration. These two can be combinedly grouped under the title "Object-Oriented gateway to GIS".

The analysis phase emphasizes building real word models, using object-oriented view of the world. The main task is to examine the needs from the perspective of the classes and objects found in the vocabulary of the problem domain. Apart from providing "object-oriented frontend to existing GIS" , by using object-oriented concepts, the system must take of the transparent line of separation between frontend and the underlying GIS. The two main objectives are to provide object-oriented concepts and GUI.

The OODB described previously is aimed at the description of object-oriented concepts that are incorporated in frontend. The high level abstraction, "Schema for an application" is the main objective, to achieve, which we decided to provide the user with the facility to create schema for each application.

The objective of providing GUI is to minimize the training overhead to users as many existing GISs force the users to undergo training in order to use the system, and make the system so user friendly, even for novice users. The primary function of GIS Viz., capture the data (spatially referenced attributes), analysis,
query and display are to be provided in an enhanced or object-oriented way.

In CIS can be made fully functional and flexible, if it allows the user to build the application with high level abstraction "schema".

While developing the system, the following tasks are to be considered and analyzed.

1). Incorporating object-oriented concepts discussed already (OODM/OODB)

2). Establishing GUI for user friendliness

3). Frontend should act as the gateway to the underlying GIS (i.e./ capture, analysis, query and display in spatial context)

4). Providing an effective communication between underlying GIS and frontend

5). To have high level abstraction for an application by providing "building the schema" facility.

4.12.1. Schema:

It is the logical organization of entities and definition of their properties for a particular application. In object-oriented database systems, each entity in the real world is represented as an object.

The concept of 'class' is proposed to classify the objects, which have same characteristics or some abstraction level i.e., characteristics of objects are defined once per class but not for each object. Operations which can be applied to each object of a class are also defined in the class. In the schema classes that
are part of an application along with the hierarchical relationship among classes are defined. While defining the class, user has to give class description as follows.

NAME: Name of the class
CLASS RELATIONSHIP: The relationships with other classes are defined, to specify class hierarchy.
SUPER CLASSES: List of super classes
SUB CLASSES: List of sub classes
STRUCTURE: The aggregation hierarchy which represents the structure of objects in the class is specified.
REFERENCE NAME: The name which is to refer the component object
DOMAIN: The name of the class to which the component must belong
METHOD: The operations which are applicable for every object in the class are specified. Two kinds of methods are available here:
Daemon Method: This method is invoked when a specified object is created or modified. It can be specified in the form
Class name: Action
Operational method: This method is used to manipulate a class or object. It can be specified in the form
Message: Action

Fig 4.4 Class description
Classes are managed to form a hierarchy with reference to their abstraction level or their relations, and the descriptions are inherited according to the hierarchy. In order to facilitate the abstraction constructs Viz., generalization, aggregation user is allowed to define two types of hierarchies in the schema. The first one is class hierarchy and the second one is aggregation hierarchy. Schema definitions consists of the definition of classes and class hierarchy.

Class hierarchy is defined as the super class-sub class relationship among the classes in the schema. It provides the user a way to generalize one or more classes as a single class. This aids user in specifying the relation between classes (abstraction at class level) i.e., all objects of a class are related to all objects of another class. The sub class inherits all the attributes and methods of its super class. Class may inherit properties from multiple super classes also. If there is any conflict among attributes of different super classes, the immediate super class attributes have higher precedence. Structural correctness of class hierarchy is to be maintained while user is defining class hierarchy.

We define a class hierarchy as correct if the graph representing the class hierarchy meets the following two conditions.

Condition 1

Either the condition \((a)\) or \((b)\) holds for any pair of classes connected, \((A, B)\), in the class hierarchy graph, but both of them are not satisfied.
a) These exists one direct path from A to B
b) If there are multiple paths from A to B, each passes through at least one node (class)

**Condition - 2**

For any class A, there is no cyclic path that starts from A and reaches A.

Aggregation hierarchy is defined in class description which is used to represent the complex objects (composite objects). Complex object classes are defined using references to other classes. The referencing instance variables (attributes or component objects) have non atomic values. Their domains are the sets of all instances of the referenced classes. The user may be given a set of primitive (int, real, char, boolean etc.) and non primitive (string, point, line, polygon etc.) predefined classes to use as references in the definition of complex object classes. However, user can also use his own defined classes as references in the description of class. It is abstraction at object level. The relation is defined between an object of a class and an object of another class. The relationship may be "part-of" or "contained-in" explained below is used to reflect the modification in CIS by allowing the user to modify the schema.

4.12.2 Schema modification [49]:

In this section, operations for modifying a schema and their effect are described. These operations are for class hierarchy, aggregation hierarchy, method, and as well as changing a class name.
There is no name conflict in the hierarchy:

No situation involving definition of names or methods which correspond to the same message in the multiple super classes. All of them are used commonly for schema definition and schema modification.

4.12.2.1 Operations for class hierarchy:

The following operations are applicable only for the class which classifies as the "root object" in the aggregation hierarchy.

Add a class:

This operation is used to add a class to existing schema. The newly added class will be, automatically, put in class hierarchy according to specification in class hierarchy in class description.

Delete a class:

Existing class can be deleted from the schema using this operation. Before deletion, the class is removed from the class hierarchy and all sub classes of class being deleted are redefined as sub classes to its super classes. However root classes in class hierarchy graph can not be removed.

Hierarchy Change:

This is used for changing super/sub class relationship between a class and its sub class. The conditions to be satisfied are given below.

- Adding super/sub relationship:

  This is to add a super/sub relationship between two
existing classes. This operation is successful only if the resultant class hierarchy graph satisfies the conditions specified for structural correctness.

- Deleting super/sub relationship:

  To delete a super/sub relationship between a class and its sub class this operation is used. In case a class with only one sub class, this operation can not be applied.

4.12.2.2. Operations for aggregation hierarchy:

In order to change aggregation hierarchy, the following operations can be used. The user is allowed to modify the aggregation hierarchy defined in a class, but inherited structures of object defined in other classes cannot be modified. These will affect the structure of class in class description.

- Add attribute:

  This operation is used to add new attribute to an existing class. The attribute may have its domain either from primitive classes or user defined classes. In case an attribute is defined in with its domain as user defined class, then that class should not be super class/sub class to the class for which attribute is being added.

- Delete attribute:

  An existing attribute can be deleted simply by specifying the reference name of that attribute.

- Change name:

  The name of the schema or any class of schema can be changed. This can be done by specifying the current name and 'new name'. The new name should not be conflicted with schema name or other
class names.

**Add/delete a method:**

Operations are to be provided to add a method to a class or to delete any method from class. However modification of method is not permitted.

4.12.2.3 Propagation of changes:

As already explained user is allowed to modify schema. The system should take care of semantics and integrity of objects that have been created earlier to modification. This can be achieved by propagating changes to classes and/or objects.

Propagation of changes to class definition:

Whenever a class definition is changed it is to be propagated to other classes in the class hierarchy. These includes:

**Addition of new instance variable (attribute);** When an attribute is added to a class, then it should be propagated to all its sub classes, because the newly added attribute has to be inherited by the sub class. If the newly added attribute is defined in any of its super class then the corresponding inherited attribute is suppressed.

**Deletion of an instance variable (attribute);** If there is any attribute, with same name, defined in any of the super classes then it is to be inherited by the current class. Also messages have to be passed to all its sub classes so that they change their structure accordingly.

**Addition/deletion of class;** Here all the super classes and sub classes of current classes have to change their class relationship accordingly.
**Propagation of changes to object instances:**

Whenever schema is modified, changes are propagated to other classes first so that they change their structure and/or class hierarchy. After that all the classes involved in the change, either directly by user or propagation of change by other classes, are to propagate these changes to existing object instances. For instance, if a class is deleted, this is propagated to all of its sub classes. Then all the objects of all sub classes are to be modified by removing attribute values that are due to inheritance property from the class being deleted. This is also valid in case of changes in class hierarchy (i.e. super class/sub class relationship).

Example: Class hierarchy:

![Class hierarchy diagram]

Method `area()` is defined to find out area of an object of administration, will be applied to each object of each descendent of class Administration.

4.12.3 In between object-oriented frontend and CIS:

The entire system can be viewed as two layers one over the other. The upper layer is the object-oriented frontend and the
The lower layer is an existing CIS with former used for building application schema. The user interact with the frontend only. The frontend's responsibilities are availing CIS functions to user and providing object-oriented data modeling capability to user. The underlying CIS and frontend must have effective communication. The two communications are frontend requesting the underlying GIS for some task (e.g., digitize, store, overlay), and the other is the communication/feedback from GIS to frontend. Acknowledgements and information regarding tasks performed by GIS as a consequence of frontend's request are communicated. So frontend is entirely dependent upon GIS for its functionality. The required task from frontend has to be converted into a request or operation which the underlying GIS knows. This task is accomplished using parser or interpreter.

4.13 System design:

The basic approach to solving the problem is selected in system design stage. While in analysis, the focus is on what needs to be done, in system design the overall structure and style are decided. The object-oriented data modeling capabilities is distributed among the four sub systems viz., schema manager, class manager, object manager, and system manager.

The frontend is under the control of system manager. System manager has the responsibility of correlating the GIS and schema (upper layer). The sub systems are described below.

4.13.1 Schema manager:

'Schema creating facility is provided by schema manager. The structure and behavior of schema are explained in object-oriented
The major responsibilities of schema manager are given below.

Fig 4.5 Object-Oriented gateway architecture

The major responsibilities of schema manager are given below.
Schema creation for each application:

The schema manager creates schema after checking for the correctness of all the information. The user is required to give schema name, structure of schema and class description of classes in order to create a schema.

Schema listing and view schema:

Existing schema is maintained by schema manager to know existing schema to enable users to create new schema. The schema structure (Partially or completely defined schema) and classes of that schema are shown to the user. Before modifying the existing schema, user may wish to view schema structure.

Resolving conflicts:

Conflicts like "duplication of names", 'referencing a class or object which is not pertaining to this schema are resolved by schema manager only.

Class hierarchy and aggregation hierarchy maintaining in a schema:

Schema manager maintains these two hierarchies in each schema. These two hierarchies can be defined or changed.

Modifying the schema, and propagation of schema changes to class manager:

Schema database has to be reflected with schema changes whenever the schema is modified. These changes are to be propagated to class manager and object manager.

Managing Schema database:

Schema database constitute schema structure, class descriptions and objects information of each class. Storing*
retrieving and modifying the schema information is taken care by schema manager.

4.13.2 Class manager:

Main function of class manager is to cooperate with schema manager in order to maintain the schema. The main tasks of class manager are listed below.

Class creation;

All classes defined in a schema are to be created. Using the class description, class is created by the class manager after checking its correctness.

View class or class list;

The class structure (its relationship on the class hierarchy), and attributes along with their domain class are shown to the user. Before any user modifies the class, user may wish to view the class structure. In a single schema, duplication of class names are prevented by the list of classes.

Modifying the class and propagation changes to object manager;

Adding an attribute, deleting an existing attribute, adding method and changing class hierarchy are included in class modification. After modification of the changes, they are propagated to object manager in order to change the objects of that class, already created.

Responding messages from schema manager:

Messages imply the changes that are propagated from schema manager. The schema manager requests class manager to change the occurrences of a class in all other classes that is being deleted from schema. Now the class definitions of all classes which are
having references to the deleted class are modified.

4.13.3 Object manager:

After the schema is created, the user then concentrates mainly on creating and manipulating the objects. The structure of the object i.e., class structure to which the object belongs, identifies, references to components and primitive attribute values are stored in a database maintained by frontend part. But the spatial data, pertaining to spatially referenced entities are stored by the underlying CIS. The references to such data are maintained by object manager. The important functions of object manager are as follows.

Creating objects:

Data from different input devices in different formats are captured for creation of an object. An object with different attributes is created by keying attribute values from key-board, or digitize maps, or specify data (remotely sensed, transformation of existing data) in some files. The underlying capture routines are to be invoked for this purpose by the object manager in coordination with systems manager which in turn sends requests to CIS.

Modification and retrieval of objects:

The changes incorporated in class definition or in class hierarchy are from class manager to object manager. The objects are suitably modified according to the changes made in the class. Objects can be explicitly modified by the user. The important task of object manager is to retrieve the objects. In
order to answer queries and to display objects, the required objects are retrieved by object manager and passed to either to display routines or system manager.

4.13.4 Transaction and map modules:

Transaction in CIS applications is totally different from that in ordinary database systems, because it includes operations over spatial objects. The analysis functions which the system provides are used to perform operations on spatially referenced entities. Map is a tool necessary to enable the user to perform transaction over the spatially referenced objects. A special kind of object class is a map which contains other geographical objects with respect to a reference to a system. Map objects can be derived by specifying map boundaries and objects that constitutes a map. Various analysis functions such as overlaying of maps, add maps, cut or zoom an existing map etc., which are provided by CIS, are used to manipulate maps to generate new maps.

So, transaction which is the main part of frontend relies on underlying CIS. Transaction consists of analysis and query processing. The underlying CIS is invoked to accomplish these two through system manager. The transaction is suitably fragmented into tasks and a request is sent to CIS for each task.

4.13.5 System manager:

System manager's responsibility is to coordinate all the managers and it has overall control on frontend. The requests from object manager, schema manager, and transaction part are
transfered to the CIS by Invoking parser. System manager process the reply or feedback given by the underlying CIS and the result is communicated to schema manager and object manager. The result may be a data file name or reference to some component object or output of some analysis function.

4.13.6 Parser:
The request delivered by system manager is converted by parser into requests or operations or commands which the underlying GIS can understand. So, design is independent of GIS.

4.14 Object design:
Object design is a mapping process, and keeps everything ready for implementation. This is called mapping process in the sense that, the ideas, strategies, and models developed earlier are mapped onto objects, data structures and algorithms. The analysis model is extended with specific implementation decisions and additional internal classes, attributes and operations by object design. During analysis, the discovered objects serve as the skeleton of the design, but object designer must choose among different ways to implement them, depending on the language selected (C can be used as it supports directly object-oriented programming) for implementation. The details of object design are given in the general object-oriented notion of class diagrams and object diagrams proposed by Booch [5],

The existence of classes and their relationships in the logical view of the system are shown in a class diagram. The class
structure of the system can be viewed in a single class diagram. The class diagram captures the structure of classes that form system architecture. The objects existence and their relationships in the logical design of the system are shown in an object diagram. Object diagrams are prototypical, in the sense that, each one represents the interactions or structural relationships that may occur among a given set of class instance, no matter what specially named objects participate in the actual implementation.

Class diagram is shown above and it shows the major classes and their interaction with other classes. The classes have vital contribution in the system development. Each class is intended to do specific task identified during system design.

CIS-schema, CIS-class, CIS-object are classes corresponding to schema manager, class manager, object manager respectively. The gateway control utilities which is responsible for integration of
classes in the system is the central component of class diagram. The whole system is monitored by this and it acts as a bridge between frontend and underlying CIS. The parser is invoiced whenever it has to communicate with the underlying CIS. This keeps track of user actions with GUI environment. The GUI part displays the graphical windows (widgets) requested by various classes on the screen and communicates the user actions to system manager or other classes. The object table is to maintain the objects and their offsets of each class (user defined class and not to be confused with classes designed) in each schema. The various object diagrams which correspond to the classes in the class diagram are given below.

Object diagrams:

The class design i.e its fields, member functions, functions used that are not members is clearly shown in object diagram. Just by looking at the object diagram which contains the sub class also, it becomes easy to understand the complete set of member functions of the sub classes. The object diagrams for the classes viz., GIS-schema, GIS-class, GIS-object and other supporting classes are described below.

1. GIS-schema:

The instances of GIS-class are present in GIS-schema. Each instance of GIS-schema corresponds to a schema defined for an application. This contains all the classes of that schema and schema database. To perform tasks like creating schema, modifying schema, deleting schema, various methods are shown in object diagram. It uses the routines 'hierarchy validation' for checking and maintaining the hierarchy class in a schema.
2. CIS-class: The CIS-class contains the class structure in terms of its super classes, sub classes and attributes. The methods defined in CIS-class create class, view class, and modify class hierarchy, relationship.
3. CIS-object:

CIS-Object maintains objects of each class. It responds to the propagation of changes in schema (i.e., whenever class definition is changed it modifies the objects) and it is part of object manager. New object, old object, the two classes inherit the structure of CIS-object and are responsible for creation of...
the objects. CIS-object points (with next pointer) to another object in the schema. Display of objects on the screen in GUI environment is done by the member function view().

Fig 4.9 Object diagram: CIS-object
4. Object table:

It is the hash table maintained by the system manager and used by CIS-object. The list of objects and corresponding offsets pertaining to a class are available in object table. It also points to another object of object table so that hashing list of all classes of schema are maintained in linked list of object table. Append(), setoffset(), delete are methods defined to create, and modify the hash table. For knowing the existence of a particular object, the method exists() is used.
5. Draw schema structure:

It provides graphical user view of a schema. When the user wants to view the loaded schema, the schema is shown as a directed acyclic graph (DAG). This structure is generated and maintained by this class. It also takes care of the subsequent modifications of the schema.

![Object Diagram](image)

**Fig 4.11 Object diagram:**

**Draw schema structure**

Complete design is explained so far. We started with brief description of object-oriented design technique. Various phases of general object-oriented design are explained. Major tasks are identified during problem analysis phase. The proposed architecture is clearly described. At the end major classes, class diagram and object diagrams are identified and role of each class in the system are explained.