Ekonomická fakulta
Faculty of Economics
Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Geographic Information Systems 1
Lecture 5: Hybrid data model, object-oriented data models

Renata Klufová
University of South Bohemia, Faculty of Economics

March 2021

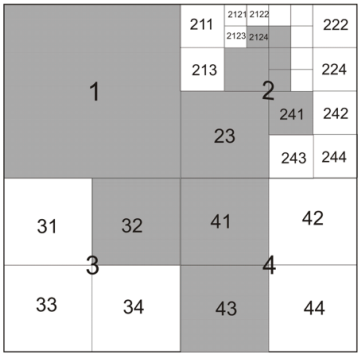
Advantages of two basic data representations

vector	**raster**
more compact data structure than raster	simple data structure
more efficient coding topology $\Rightarrow$ more efficient implementation of operations	easy superposition and combination of data with remote sensing data
accurate graphics	easy types of spatial analyzes
searching, updating and generalizing graphics possible	cheap and intensively developed technology

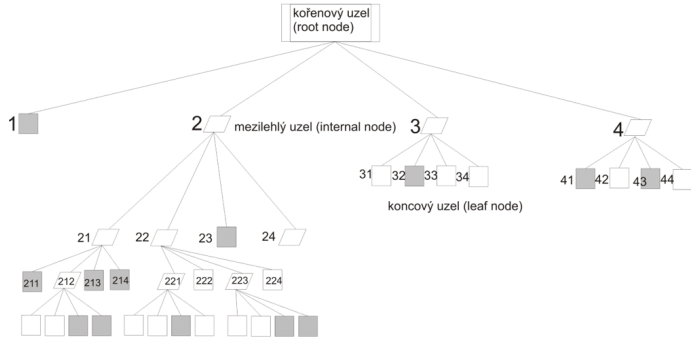
Disadvantages of two basic data representations

vector	**raster**
complex data structures	large volumes of graphic data
difficult superposition of several vector polygon data or polygon and raster data	topological links are difficult to show
each unit has a different topological form $\Rightarrow$ difficult modeling	jagged graphics output - can be reduced by increasing the resolution, but increasing the file size
display and graphic output can be difficult to adjust	transformation of cartographic representations - - time consuming, special algorithms
expensive technology - complex software and technical equipment	difficult modeling of network links

based on the gradual regular division of a square area into quadrants and subquadrants until either the value of the attribute is the same in the whole quadrant or the smallest possible quadrant size (pixel size) is reached

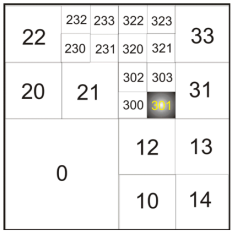


- the quadtree structure can be most easily represented by a graph (tree),
- **depth** of the tree ... the number of edges lying on the longest path from the root to any leaf - the notation *n* - indicates the quadtree resolution.



representation in computer memory - e.g.:

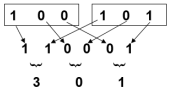
- the oldest - explicit record of all quadtree nodes, edges represented by pointers to child nodes - very cumbersome structure,
- **linear quadtree** - only end nodes (sheets) are recorded - each of them marked with a special numeric key from which the position of the node in the quadtree can be derived - allows to derive spatial relations - the most used key type... **Morton's key**.



prokládání bitů:

sloupcový index = $5_{10} = 101_2$

řádkový index = $4_{10} = 100_2$



... it arose from the need for joint uniform processing of vector and raster data.

Deren a Jianya (1992): **Unified Data Model**:

- combines the advantages of vector and raster model and thus enables simultaneous storage and processing of data from GIS, DMT and remote sensing,
- is based on a linear quadtree - the geometric description of geoelements is expressed using Morton keys instead of x and y coordinates,
- to increase the accuracy of point position representation, the quadtree base cell is further divided by a fine grid (again represented by a quadtree) - the location of the point is then represented by two Morton keys - the first indicates the position of the point in the basic quadtree and the second in the finely divided grid.

- lines - represented by a set of Morton keys, which contains not only the position of the start and end node and all vertices, but all the way through the basic quadtree,
- areas - coded similarly - boundary lines + cells of the boundary quadtree lying in this area.

reasons:

- effort to increase the functionality and efficiency of GIS,
- growth of the amount of available data - problems with storage and processing,
- analyzes - transition from spatially oriented analyzes to object-oriented analyzes,
- today's data structures are optimized with respect to data storage and manipulation without taking into account an accurate representation of the real world,
- geo-elements are reduced to points, lines and areas, resp. pixels,
- data structures emphasize primarily the localization of geoelements, store thematic data completely separately and thus significantly complicate the implementation of spatial analyzes,
- neither the vector nor the raster model exist in the real world - roads are not lines, cities are not points, pixels represent completely arbitrary places in space.

OOGIS ... object oriented GISes:

- work with objects that correspond to specific real geoelements,
- objects describe both spatially and thematically, temporal, relational and functional ⇒ multidimensional entities, which allow solving problems using earlier data structures unsolvable - such as overlapping geoelements (river and valley) or geoelements with unclear spatial boundaries (fuzzy constraint - valley and adjacent ridge).

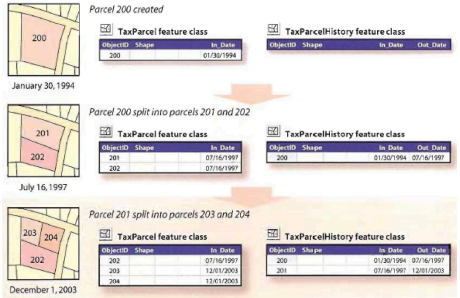
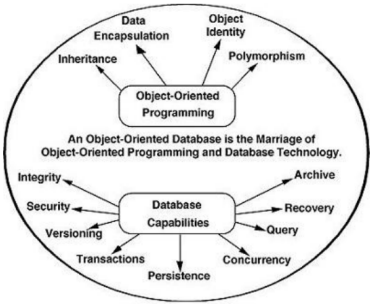


Figure 2.10 ESRI parcel model

OODBMS ... object oriented systems of database management:

- program objects correspond directly to real objects,
- objects can capture any order of complexity ⇒ can store various data types and their complex relationships,
- objects can contain other objects ⇒ can affect any complex structure.



OOP ... Object Oriented Programming:

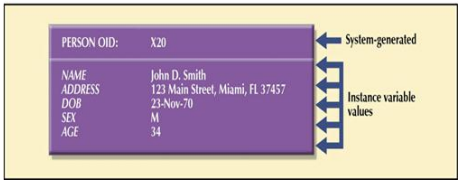
- **Objects** - the individual elements of the modeled reality in the program are grouped into entities called objects. Objects remember their **state** and externally provide **operations** (accessible as call methods).
- **Abstraction** - aims to hide complexity from the users and show them only the relevant information. For example, if you want to drive a car, you don't need to know about its internal workings. You can hide internal implementation details by using abstract classes or interfaces. On the abstract level, you only need to define the method signatures (name and parameter list) and let each class implement them in their own way.
- **Encapsulation** - allows us to protect the data stored in a class from system-wide access. As its name suggests, it safeguards the internal contents of a class like a real-life capsule. You can implement encapsulation in Java by keeping the fields (class variables) private and providing public getter and setter methods to each of them.
- **Association** - the act of establishing a relationship between two unrelated classes. For example, when you declare two fields of different types (e.g. Car and Bicycle) within the same class and make them interact with each other, you have performed association.

OOP ... Object Oriented Programming:

- **Aggregation** - narrower kind of association. It occurs when there's a one-way (HAS-A) relationship between the two classes you associate through their objects. For example, every Passenger has a Car but a Car doesn't necessarily have a Passenger. When you declare the Passenger class, you can create a field of the Car type that shows which car the passenger belongs to. Then, when you instantiate a new Passenger object, you can access the data stored in the related Car as well.
- **Inheritance** - makes it possible to create a child class that inherits the fields and methods of the parent class. The child class can override the values and methods of the parent class, however it's not necessary. It can also add new data and functionality to its parent. Parent classes are also called superclasses or base classes, while child classes are known as subclasses or derived classes as well.
- **Polymorphism** - refers to the ability to perform a certain action in different ways. In Java, polymorphism can take two forms: method overloading and method overriding. Method overloading happens when various methods with the same name are present in a class. When they are called they are differentiated by the number, order, and types of their parameters. Method overriding occurs when the child class overrides a method of its parent.

- represents a real entity,
- contains information that describes its properties and behavior,
- an individual object is distinguished by elements of information that distinguish it from other objects.

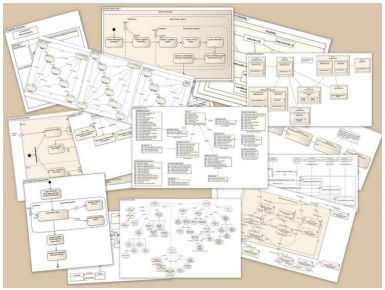
FIGURE 11.16 STATE OF A PERSON OBJECT INSTANCE



Object state = state (form) of the object, ie values of its properties at a given moment in time. Changing the state of an object over time takes place through the object's internal action or the object's interaction with the environment.

= graphical language for visualization, specification, design and documentation of software systems - supports OOP
The UML standard is defined by the **Object Management Group** (OMG) standardization group.
history:

- Grady Booch, James Rumbaugh, Ivar Jacobson - methodology of object-oriented analysis - 80s and early 90s,
- Rational Software co. - 1995 - first UML design - version 0.8.
- Jacobson - implementation of the **Objectory method** into UML - approach to OOSE (Object Oriented Software Engineering) standards,
- UML consortium - large SW companies (Digital Equipment Corp., Hewlett-Packard, IBM, Microsoft, Oracle ...),
- 1997 - standardization, OMG control.

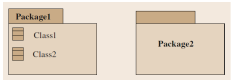


ways of use:

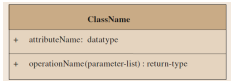
- **Draft Drawing** - In this application, UML is a support tool for communication between developers and for recording ideas and suggestions. Only the things that are essential for the graphical representation of the design, parts of the design, are drawn in the diagrams before programming begins. Comprehensibility, speed of drawing and ease of change or design of solution alternatives are important.
- **Drawing detailed designs** - complete design or complete implementation. When drawing a design, the analyst should include all the elements so that the programmer is able to create a program without much thought over the subject matter (the programmer should not need to consult with the user). When drawing detailed designs, specialized programs (CASE) are usually used, which are able to share information between individual models and check the consistency of the design.
- **UML as a programming language** - the developer draws UML diagrams, from which directly executable code is generated. This requires specialized tools and very accurate expression in UML diagrams. In this context, the term **Model Driven Architecture (MDA)** is very often used, another OMG standard that seeks to standardize the use of UML as a programming language.
- **Metamodel** - this view is used by UML authors and CASE tool authors - they do not look at UML as diagrams, for them the UML metamodel is the basis (diagrams are only a graphical representation of the metamodel). In this approach, the term model is often used instead of diagram, eg the term class model is used instead of class diagram. The metamodel is described using **Meta-Object-Facility (MOF)** - an abstract language for specifying, creating and managing metamodels (another OMG standard). XMI - an XML-based standard (part of the UML standard) is used to exchange metamodels.

Conceptual information modeling mainly uses structural diagrams, especially package diagrams and class diagrams that use elements derived from the UML kernel.

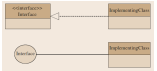
- **Package** - space for a group of elements contained in a package.



- **Classifier** - describes a set of objects that have common properties, each such object is an instance of the class - types:
  - **Class** - a set of objects that share the same set of semantics, properties, and constraint specifications.



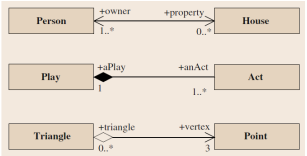
  - **Interface** - describes the service offered by instances of any class that implements this interface.



  - **Datatype**

Feature class

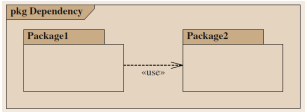
- **Attribute** - characteristic (property) common to subjects of class,
- **Operation** - activities that can be performed with the object,
- **Association** - specifies the connections (relationships) between elements and classes,



- **Multiplicity** - specifies the number of elements of the class that can be associated with the class on the other side of the association: 1 (number) - a specific value (here exactly 1), * - any number (ie 0), 1 .. * (interval) - using 2 dots we can denote the interval;
- **Navigability** - describes the ability of one element to use information contained in another element.

Feature class

- **Aggregation** - represents a whole - part relationship.
- **Composition** - similar to aggregation, but represents a stronger relationship. The entity of the part makes no sense at all.
- **Association class** - mediates the relationship between two entities.
- **Generalization** - taxonomic relationship between more general and more specific elements.
- **Stereotype** - they extend the semantics of already existing elements, but do not affect their structure.
- **Note** - text information.
- **Constraint**
- **Dependency** - indicates that the implementation or operation of one or more elements requires the existence of one or more other elements.

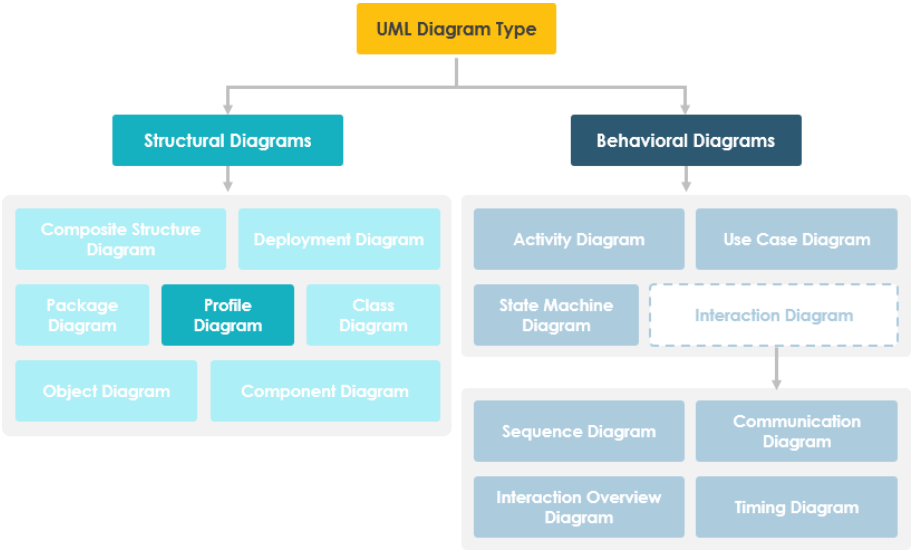


advantages of UML:

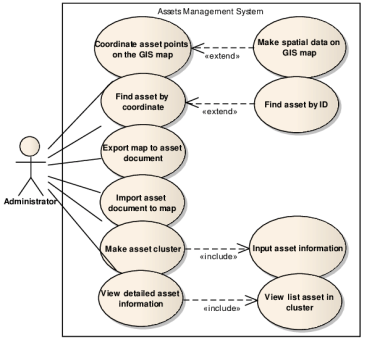
- Simplifies complexity.
- Keeps communication open.
- Automates software and process development.
- Helps resolve persistent architecture issues.
- Improves the quality of work.
- Reduces costs and time to market.

types of diagrams:

- behavioral diagrams,
- structure diagrams.


UML Diagram Type
Structural Diagrams
Behavioral Diagrams
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Class Diagram
Object Diagram
Component Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

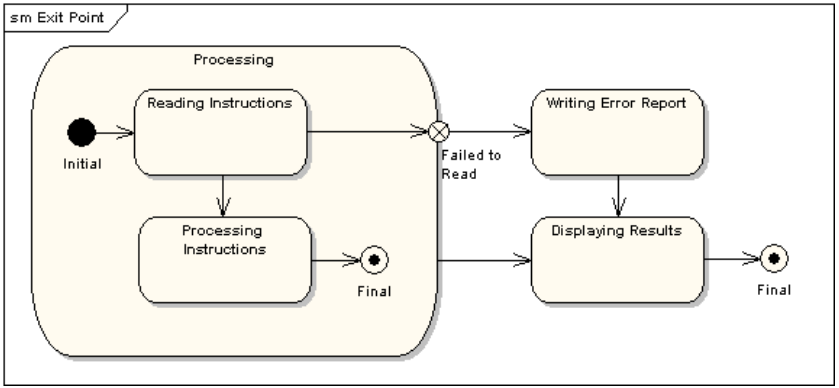
Use case diagram - give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.



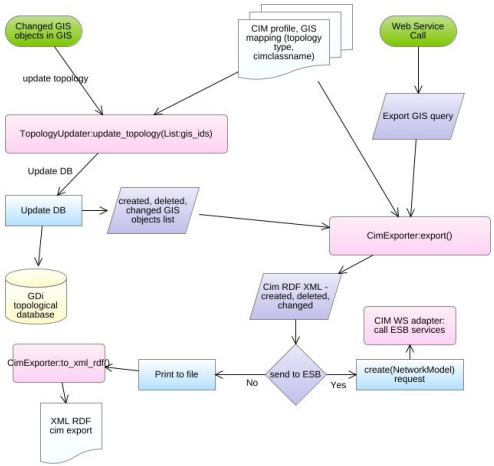
basic relationships:

1. **Association** between actor and use case,
2. **Extend** between two use cases,
3. **Include** between two use cases,
4. **Generalization** of an actor,
5. **Generalization** of a use case.

State diagram - sequence of states of a given object

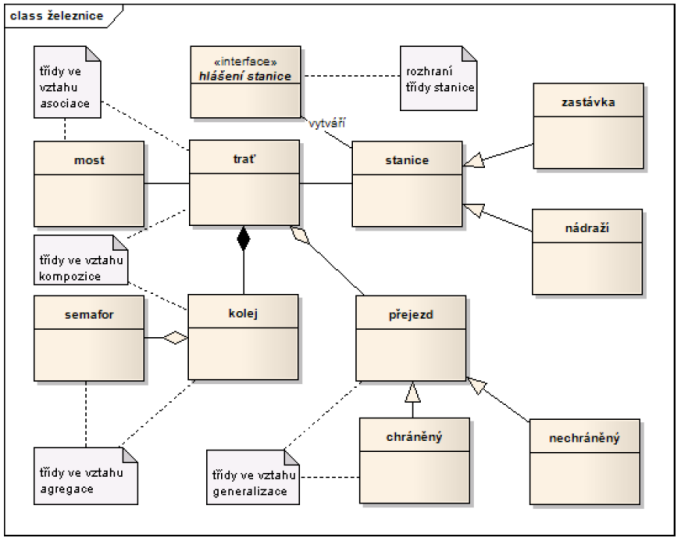


Activity diagram - represent workflows in a graphical way. They can be used to describe the business workflow or the operational workflow of any component in a system.



Class diagram - main building block of any object-oriented solution - it shows the classes in a system, attributes, and operations of each class and the relationship between each class.:

- **Association** – relationship between one or more classes - eg track and station. The relationship of the association means that there can be communication between these classes,
- **Generalization** – generalization is a relationship where one superclass contains several other subclasses that inherit some of its properties. E.g. the superclass crossing has two subclasses protected and unprotected,
- **Composition** – a composition relationship occurs between the track and track class if the "track" class represents the whole and the "track" part of it. If there is no track, there is no track,
- **Aggregation** – there is an aggregation relationship between the track and crossing class. The crossing is always on a certain track, but not on every track there must be a crossing. The traffic light and the track are in the same relationship, because the track can also exist without a traffic light, but the traffic light must always belong to a certain track,
- **Interface realization** – there is an implementation relationship between the station class and the station reporting interface, this relationship expresses that the station reporting implements specific functions based on the properties and signals from the station class. The interface

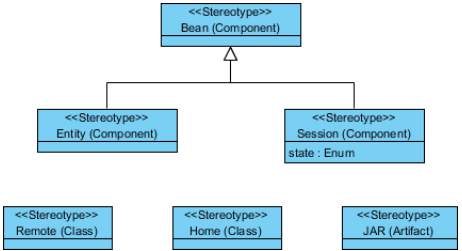

class železnice
třídy ve vztahu asociace
«interface»
hlášení stanice
rozhraní třídy stanice
vytváří
zastávka
most
trať
stanice
nádraží
třídy ve vztahu kompozice
semafor
kolej
přejezd
chráněný
nechráněný
třídy ve vztahu agregace
třídy ve vztahu generalizace

other diagrams:

- **Object diagram** - describes the behavior and relationships of individual instances of classes - objects. They usually describe how objects work together to solve a specific problem (use case).
- **Component diagram** - component = an executable part of the code that provides a summary of the services in the form of a so-called "black box". Nowadays, software is increasingly being created from components.
- **Composite Structure diagram** - shows the internal structure of more complex elements of the model (use cases, classes, components) and shows the cooperation of this element or classifier with other elements in the system.
- **Package diagram** - combines elements of UML diagrams (e.g. classes or use cases) into so-called packages and shows dependencies between these packages. The packages themselves can be part of class or use case diagrams, and can lighten and streamline diagrams that work with a large number of elements.

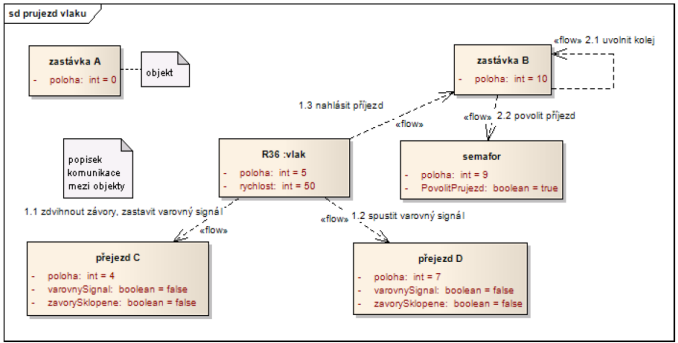
other diagrams:

- **Deployment diagram** - expresses the layout of individual system components and their relationship to the hardware.
- **Profile diagram** - expresses the relationships between stereotypes, constraights, and metaclasses that are used in models.



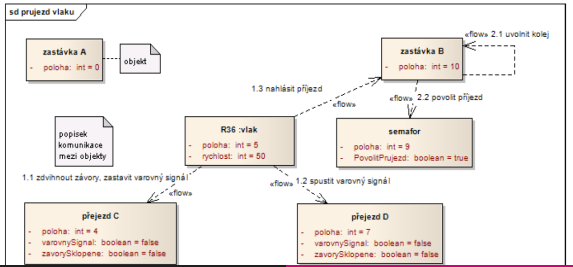
other diagrams:

- **Communication diagram** - shows the method of communication between objects at a certain moment - eg train R36 in a certain part of the line between stops A and B and crossings C and D. The diagram describes the order and meaning of messages sent by object train R36 at a given time to objects stop B, crossing C , crossing D and further messages which are transmitted within the stop B and to the traffic light building.



other diagrams:

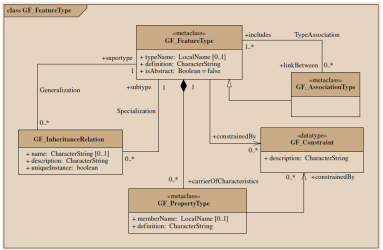
- **Sequence diagram** - shows the behavior and cooperation of individual objects within one use case.
- **Interaction diagram** - captures a scenario of a certain activity in the system = a variant of the activity diagram, which shows the concurrency of activities of individual classes and communication between them.
- **Timing diagram** - shows changes to the state, condition of an object, or role object; in relation to time.



ISO/TC 211: GFM = metamodel for representation of elements in the application schema - metamodel elements can be represented as elements of the UML model of the application schema

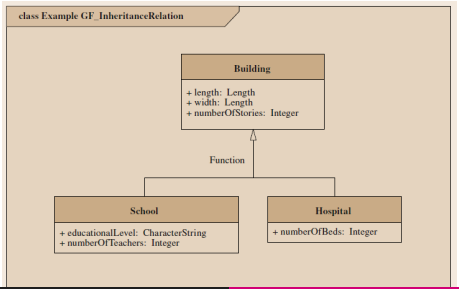
basic building blocks:

- **GF_FeatureType** - metaclass - 3 attributes that are expected to be included in each element:
  - *typeName* - contains the name of the specific type of elements represented by the created class,
  - *definition* - contains an element type definition and is "carried" as a class attribute,
  - *isAbstract* - contains a Boolean value indicating whether or not the created class is abstract. If the value is TRUE, the UML class name is displayed in italics to indicate an abstract class.



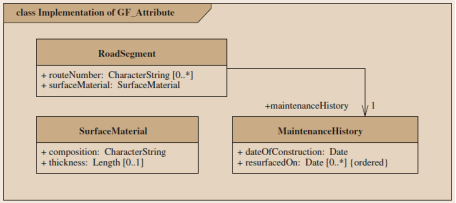
elementary building blocks:

- **GF_InheritanceRelation** - the **GF_FeatureType** metaclass is associated with the **GF_InheritanceRelation** class by two generalization and specialization associations. This structure shows that two elements of the GF_FeatureType class can be related to each other as a superclass and a subclass in a generalization hierarchy.
  - = generalization UML relationship,
  - e.g.: the element type Building (*Building*) has two subclasses: school (*School*) and hospital (*Hospital*). The Building element type has *three attributes*: length, width *, and *numberOfStories*, which are inherited by both of its subclasses. Each of the subclasses has specific attributes: for schools: educational level (*educationalLevel*) and *numberOfTeachers*; for hospital: number of beds (*numberOfBeds*). The two generalization relationships belong to a generalization set named *Function*.



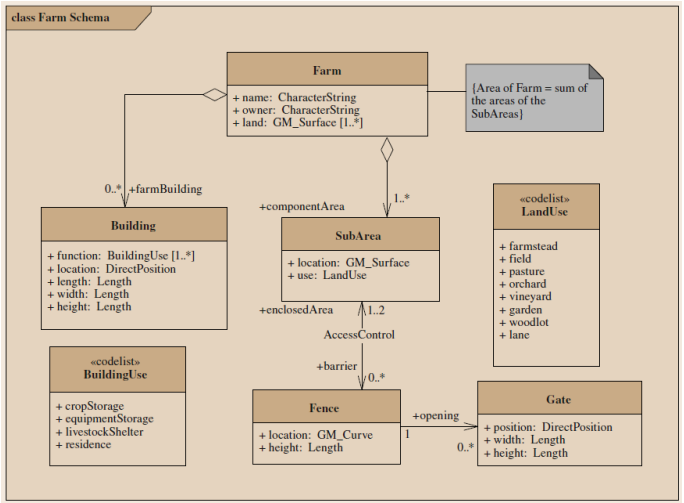
elementary building blocks:

- **GF_AssociationType** - relationships between GF_FeatureType and GF_AssociationType;
- **GF_Constraint** - a description of the restriction that may be applied to the case of any associated metaclass - e.g. a road on which only vehicles with a speed higher than 60 km/h are allowed to move;
- **GF_PropertyType**,
- **GF_AssociationRole** - describes the role that an element type instance can have in association with another instance of the same or another type;
- **GF_AttributeType** - describes the type of attribute that belongs to a particular element type.



- **GF_Operation** - describes operations associated with an element type.

application scheme for the description of the spatial characteristics of one farm:



advantages:

hybrid model	**object model**
easy option to modify	more suitable for modeling complex objects
easy integration of attribute data with other relational systems and applications	whole object - no separate attributes from spatial data
easy to use	rasters and vectors in one database
elaborate theor. basics for rel. database	less disk space
standardization in RDBMS (SQL)	easy editing of object properties
	one object can have multiple representations
	support for other types of data (multimedia, ...)

disadvantages:

hybrid model	**object model**
insufficient possibilities of time data processing	various OODBMS still incompatible
no prevention of integrity violation in DBMS	object identification is often difficult (especially for continuous rasters)
slow polling, especially for complex objects	less experience than with hybrid systems
insufficient support for "long transactions"	more complex database design - def. and method
rel. poor adaptability to requirements of a concrete application	OODBMS demanding on HW

Than you for your attention.