Vrije Universiteit Brussel - Belgium

Faculty of Sciences

In Collaboration with École des Mines de Nantes - France

and

Universidad Nacional de la Plata - Argentina

2002

A Conceptual Object-Oriented Model for Complex Geographical Entities
– the DDCO model –

A Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
(Thesis research conducted in the EMOOSE exchange)

By: Emmanuel Labbe

Promotor: Prof. Théo D'Hondt (Vrije Universiteit Brussel)
Co-Promotor: Dr. Silvia Gordillo (Universidad Nacional de la Plata)

Abstract

The modeling of complex entities or phenomena in geographical information systems has to face numerous problems due to the highly variable attributes they are composed of. Moreover, each particular type of GIS objects to model seems quite different from the others, and finding a global design approach has hardly succeeded until today.

The object-oriented paradigm has proven itself to be a good modeling approach, and coupled with an appropriate design methodology it can facilitate the modeling of such entities. This work defines a new way to represent geographical objects by abstracting them in a common and uniform conceptual model. To cope with the complexity of GIS objects, this work tries not to focus on the dimensions objects or phenomena could be composed of, but on the dependencies that relate these dimensions.

This abstraction brings uniformity in the development of a geographical application. The same mechanisms are used to model every GIS objects. Therefore, a kind of methodology design approach logically comes from the model abstractions. It gives the bases for future object-oriented frameworks that would enable to easily conceive geographical application manipulating complex objects or phenomena.

Acknowledgements

I would like to take the opportunity to thank all the people who contributed in a way or another to the achievement of this thesis. Among others:

- Silvia and Gustavo, who gave me a warm welcome to Argentina, Silvia also for having been a pleasant advisor.
- people I met in Argentina, who contributed to make time going faster, sometimes a bit too fast.
- my EMOOSE classmates, for a great 6-month period in Nantes.
- I would like to give Jessie my apologizes for having been a so difficult roommate in Argentina. I guess, however, we spent quite a good time.
- Agustina, for giving me light when shadows appear.
- and last but not least my family, for being. . . my family.

“ *Le Temps nous égare*
Le Temps nous étreint
Le Temps nous est gare
Le Temps nous est train ”

– Jacques Prévert.

Contents

Chapter 1

Introduction

Before presenting the aim of this research, this chapter introduces the notion of Geographical Information Systems (GIS), describing briefly their main characteristics[1].

1.1 What are Geographical Information Systems?

1.1.1 Definitions

A Geographical Information System (GIS) has been defined by the Federal Interagency Coordinating Committee for Digital Cartography (FICCDC) in 1988 as a "*System of computer hardware, software, and procedures designed to support the capture, management, manipulation, analysis, modeling, and display of spatially referenced data for solving complex planning and management problems*".

In [Did90], Michel Didier describes a GIS as a set of spatially referenced data, organized in a way to easily extract synthetic information useful for decision-making process ("*Ensemble de données repérées dans l'espace, structuré de façon à pouvoir en extraire commodément des synthèses utiles à la décision*").

These two definitions both agree on the fact that a GIS handles – in some ways – geographical data. They put in prospective, however, different aspects of GIS.

[1]For a more complete introduction on GIS, the reader can refer to [Aro89]

Geographical data

Geographical data are information related to objects or phenomena of the world that we can perceive. They are usually composed of four major components:

- **Attributes, type, appearance** of the object or phenomenon considered. For example, attributes of a forest could be its density, the species of the trees it is composed of and their average height.
- **Relationships** with others entities. A building may belong, for instance, to a city.
- **Time**. The object or phenomenon considered – with particular attributes, relationships, etc – only exists at a given time, or during a given period of time.
- **Location**. This is the most important component, the one that justifies the essence of geographical data. Every object or phenomenon has a location, described in a concrete reference system.

What differentiates a GIS from other information systems is the manipulation of spatially referenced data, which implies that a GIS must deal with spatial relationships among the objects and phenomena it handles. This spatial analysis is unusual in other classical information systems.

Processes and functionalities

The FICCDC centered its definition on the processes and functionalities a GIS should provide. In order to "*solve complex planning and management problems*", the first step of a GIS is to capture data from the real world. The captured data can come for example from measurements on the field, from sets of data recorded by captors or similar devices, from interpreted data – which means data already processed by a human thought, for instance the results of surveys –.

The data are introduced into the system, in order to be processed. The system must provide a way to store, organize, and give access to the information. This is known as data management.

The relevant data for solving the problem are extracted from the system and analyzed. The results are communicated to the user, usually by means of maps, graphics and schemata.

Analysis and decision support tool

The definition of Michel Didier clarifies the final purpose of a GIS: decision-making. GIS provide a way to capture geographical information, to select and analyze the relevant ones, and to draw conclusions that will be the bases on which decisions can be taken. These decisions will lead to actions, which might change the real world from where the initial data were captured. Figure 1.1 summarizes this global cycle.

In the case of an epidemic for instance, studying the distribution of the contaminated persons can help determining the epicenter of the disease and thus its root causes, which will enable to take the decisions to stop the epidemic.

Note that the manipulation of geographical data to support decision-making is anterior to the appearance of computer technics. A strategic map is an example of tool which handles geographical information and is used to take decisions. Besides, maps are often used by GIS as a way to represent the data. Nevertheless, the digitalization of the information and the digital manipulation allowed to treat a greater amount of data, in a faster and easier way.

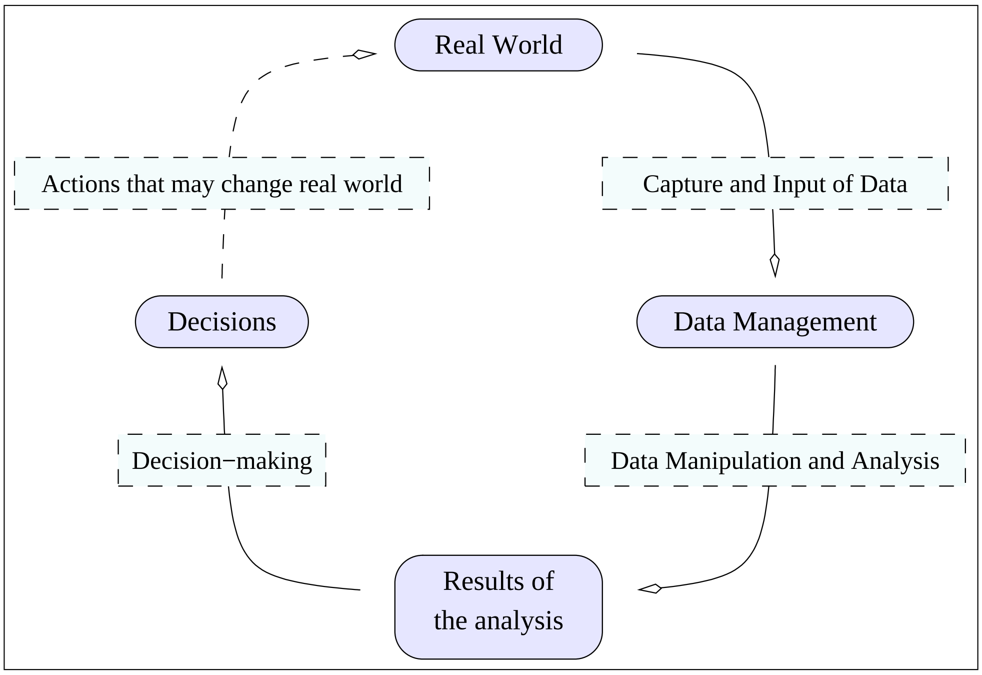

Real World
Actions that may change real world
Capture and Input of Data
Decisions
Data Management
Decision−making
Data Manipulation and Analysis
Results of the analysis

Figure 1.1: Geographical information processing

1.1.2 GIS classical spatial data models

The main information contained in geographical data are the spatial localizations of the objects or phenomena. We understand by localization of an object both its shape and position. With the computerization of GIS, two main approaches – shown in figure 1.2 – grew in parallel to digitally describe the localizations: the raster and the vector data model.

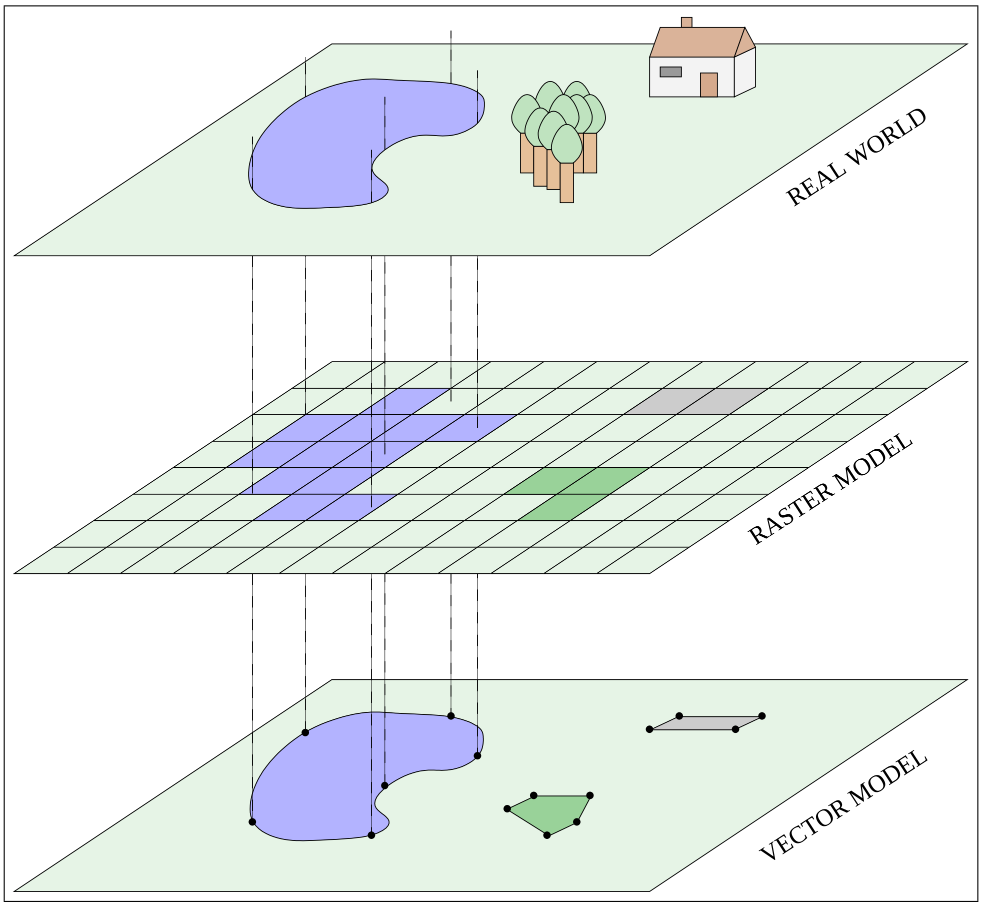


Figure 1.2: Spatial data models

Raster data model

In raster data models, space is regularly subdivided into cells. Each cell represents a surface of the real world. The area of the surface defines the resolution of the model. The cells are linked with sets of values which define for each what are the phenomena or objects that take place on the surface they define.

Usually the cells are rectangular, but in fact, it works with any tessellation. This model is often used when manipulating digital images, for instance when the information is coming from pictures or is aimed for output devices like common screens.

The raster data model is a simple data structure. Nevertheless, as mentioned in [Aro89], "*the units of the raster model do not correspond to the spatial entities they represent in the real world*". The objects or phenomena positions only exist as collections of cells. Thus, topological relationships between them are difficult to extract from the model.

Vector data model

The geometry of objects or phenomena is described, in the vector data model, by points belonging to their geometrical boundaries. Points can be connected together by a line segment or any mathematically defined curve.

In this model, each object or phenomenon is represented by a particular entity: the location of a thermometer can be described by a point, a road by a line segment, a forest by a polygon corresponding to its boundaries, etc. Spatial relationships between objects of the real world can be 'translated' into relationships between their representations. If two houses are neighbors, we can equivalently declare the entities modeling their geometry as neighbors. The model makes it easy to add some spatial relationships knowledge, to be used for topological analysis.

1.1.3 Temporal GIS

Until now, most of the GIS emphasized the spatial aspects of geographical data – localization and spatial relationships between objects – and let temporal aspects be just an attribute in the relational side of the models. Such GIS are usually able to answer complex queries in regard to the spatial point of view. On the other hand, they hardly take into consideration time for the queries.

This emphasis of static representations of reality constraints GIS capabilities for representing dynamic information, necessary for modeling objects or phenomena that may change in space and time for instance.

As most of the objects and phenomena in nature are dynamic ones, temporal GIS gained a lot of consideration in the past ten years, and research is investigating since then how to combine temporal and spatial aspects. Nevertheless, this field is still at an early stage of development, and most has to be done especially for the data representation.

1.2 Aim of this research

1.2.1 Goals and orientations

The goal of this work is to set up the bases, focusing on data representation, for an object-oriented framework that would enable to easily conceive GIS manipulating complex object or phenomena, like spatiotemporal ones. This work tries to go beyond the already existing projects by reconsidering what are geographical data.

Usually, particular models are available to deal with particular types of objects or phenomena. For instance, models exist for representing continuous fields [Gor01]. A GIS will have to implement a different model for each type of objects or phenomena it wants to feature. Thus, it will be limited in its development and its use by and to the models it chose.

On the other hand, some more generic models also exist, generally in the form of design support tools for data model [PSZ99, RSKH01]. Nevertheless, they often force the user to adapt to their own views with regard to the integration between space, time and all the other dimensions a GIS could be interested in.

This work tries to open a new path for Temporal GIS and complex objects or phenomena modeling, by observing that in fact most of them could be represented by the same kind of structure. A field of temperature can be seen as a temperature depending on a position, while a moving point can be seen as a position depending on a time. This notion of dependency is the same in both cases: a unique temperature exists for a given position in the first case, while a unique position exists for a given time in the second case. Other examples not involving time or space can be thought of, for instance the appearance of an object depending on a scale parameter.

The main idea is not to focus on the dimensions that objects or phenomena could be composed of, but on the dependencies that relate these dimensions, offering a more uniform and wider range of modeling possibilities.

1.2.2 Organization of the thesis

This thesis is organized in four parts. In the first one, chapter 2, a survey of existing modeling approaches for spatiotemporal data models in GIS is given. Chapter 3 then presents a new object-oriented based approach for modeling complex phenomena or entities in GIS. An example of use of the model of chapter 3 is presented in chapter 4, and finally, a quick analysis of its suitability for GIS data modeling is given in chapter 5.

Chapter 2

Existing spatiotemporal GIS data models

Modeling complex objects or phenomena in GIS has been investigated by research focusing on spatiotemporal aspects mainly. This chapters gives an overview of this research in spatiotemporal data models.

The main research carried out on spatiotemporal concepts for GIS took place in the 90's. It has been inspired, and even limited sometimes, by the similar research in the field of databases. The global trend for modeling spatiotemporal data in GIS went from the space-dominant view of time-stamping layers to a more time-dominant view of time-stamping events and processes. This evolution is quite similar to the move from relation tables to objects in databases.

This chapter will first give an overview of space-dominant and time-dominant models. It will then describe other approaches and discussions that have been carried out on the theme of spatiotemporal data models and applications.

2.1 Space-dominant view models

Monica Wachowicz presents in [Wac99] space-dominant models main characteristics. In such models, space is viewed as a container. Geographical elements are arranged in space according to their position. Layer-based (raster) or vector models are usually used to describe the spatial arrangement. Time is introduced by associating a time value to geographical elements. It could be a 'time point' or a 'time period', to describe the state of the geographical entity respectively at a given instant or during a given period of time. Existing space-dominant models mainly differ one from the other in what they time-stamp.

2.1.1 Time-stamping layers

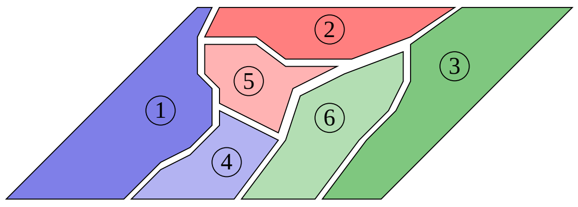
Armstrong in [Arm88] has investigated the introduction of temporal aspects in GIS from the perspective of spatial databases. In his model, geographical elements are organized in layers. A layer stores data that are thematically and temporally homogeneous. Thus, for a given theme, the states of geographical entities at different times will be represented by different 'snapshots'. Note that this model does not involve any explicit temporal relationships between the snapshots.

This simple model has inspired some others. Beller and al. [BGLS91] implemented a prototype temporal GIS based on the Temporal Map Set (TMS) object. A TMS object is defined as a collection of GIS maps representing the same area and theme at different instants. The model also introduced the notion of event thanks to separate binary TMS. They are composed of binary maps that determine whether a geographical entity belongs to the event or not.

The main draw back of this approach relies on the redundancy of the data. Having two snapshots of a given theme at different instants does not imply changes between them. Moreover, there is no constraint from one layer to another, leading to an increased risk of having inconsistencies between the layers.

2.1.2 Time-stamping attributes

In [LC88] Langran and Chrisman describe a model – space-time composite – which conceptually represents the changes of spatial entities in time. A space-time composite object is a single-attribute unit. The world is represented as a layer composed of spatially uniform space-time composites, each of them having its own set of attribute modifications in time. Thought from the perspective of cartography, this model can capture changes occurring to spatially static objects. Figure 2.1 shows an example of the use of the space-time composite model. The world is made of an aggregation of regions. A region has an attribute that determines if it is covered by water, sand or forest. The regions are fixed in space, but the space-time composites record the coverage evolution of each region.



Time	①	②	③	④	⑤	⑥
	L	L	F	L	L	L
	L	S	F	L	L	S
	L	S	F	L	S	F
	L	S	F	S	S	F

(L: Lake, S: Sand, F: Forest)

Figure 2.1: Space-time composite model

On the other hand, the concepts behind the model prevent it from capturing temporality across space. It can not be used to capture motion of geographical objects for example.

Furthermore, an update in the geometry of the space-time composite objects or in topological relationships among them involve a reconstruction of the database in order to reorganize the space-time composite units and their attributes tables. For instance, in the example shown in figure 2.1, if the region ⑥ has to be split as it is now covered half by sand, the database will have to be remodeled so that the region ⑥ does not exist anymore, replaced by a region ⑥$_1$ and a region ⑥$_2$ and their respective attributes tables.

2.1.3 Time-stamping spatial objects

Time-stamping spatial objects, like [Wor94] with the spatiotemporal object model, has been the third approach for space-dominant models. Spatiotemporal objects live in a 3-dimension world composed of a 2-dimension space and a 1-dimension time. A spatiotemporal object is composed of spatiotemporal atoms, each of them holding a set of fixed properties of the object, among them a geometry and time. This model emphasizes the notion of individual entities in a space and time frame. The model enables to capture changes of these entities in both time and space, by projecting the spatiotemporal atoms they are composed of in the spatial and time dimensions of the world.

One of the major drawback of these models is their lack in time related knowledge due to their space-dominant view of the world. For all of them, time is represented as an independent and linear structure. There is no link between the snapshots or the spatiotemporal atoms. The notions of transitions or processes do not exist. Moreover some changes can remain uncaptured, if happening between two snapshots for instance.

This kind of models leads analysis to be based on the spatial similarities or differences between the 'layers' at different instants. Time can hardly be taken into account for making the queries, and the analysis will hardly reveal the relationships linking the different geographical entities, for instance cause/consequence relationships between events, which are useful for the decision support role of GIS.

2.2 Time-dominant view models

A second global approach for modeling spatiotemporal data is to have a time-dominant view of the world. The main difference with the previous models is the notion of events. Changes are no longer just associated to a time parameter: events lead to changes of the geographical objects. Time information are usually kept within the notion of events and processes, and time exists as time sequences or time objects. While geographical elements were arranged in space according to their position in the space-dominant view, geographical elements are here generally organized according to the events they belong to.

2.2.1 Events-dominant approach

The approach of Peuquet and Duan in [PD95] is similar to the Temporal Map Set approach of [BGLS91] as it records the evolution of a thematic layer, grouping the different states of an evolution caused by a particular event together. Peuquet and Duan, however, focused their model on a temporally-based representation as, in their opinion, it is a well-suited form for "*analyzing overall temporal relationships of events and patterns of events throughout a geographical area*". Their goal was to facilitate the analysis of temporal relationship and change patterns through time, while keeping a relatively efficient data structure in term of size.

The ESTDM (Event-based Spatio-Temporal Data Model) records a base map (about a particular geographical theme) that corresponds to the initial state of the world. A sequence of time-stamped events is associated to the base map. An event is composed of sets of changes that describe where and how the event altered the world. Thus, instead of recording a new global snapshot whenever a change occurs as it was the case in the TMS model, the model only records the changes that took place between two states. Moreover, the explicit association of each change to particular events enables to "*represent the spatiotemporal manifestation of some process*".

If the model has proven itself with temporal and spatial queries, it still lacks the notion of geographical entities. Well suited for describing the evolution of thematic maps – distribution of the population in a country for instance –, it can hardly represent interactions between geographical entities. Modeling a cadastral problem would be done in an unnatural way, loosing the notion of land unity, affecting the queries that could be done to the system.

2.2.2 Processes, part of geographical objects

In [RL95], Raper and Livingstone tackled the problem of spatiotemporal modeling for GIS by proposing "*the creation of new integrated object-oriented modeling environments*". Instead of suggesting a rigid model for geographical objects, they described *OOgeomorph* – an object-oriented spatial modeling system for geomorphology –. *OOgeomorph* acts like an environment in which phenomena can be modeled according to user-defined criteria.

A geomorph_system is composed of several forms, processes and materials classes, implemented in function of the domain targeted by the system. A phenomenon is then modeled by composing instances of these categories classes. These forms, processes and materials objects are composed of attributes objects, which each refer to a 3 dimensional position and 1 dimension time.

Note that Raper and Livingstone wanted to separate their model from low level storage concerns. They considered a geomorph_system more as an organized view of some data stored in an already existing GIS or spatial database. A geomorph_info object plays the role of interface and translator between both of them.

The *OOgeomorph* model has some similarities with the spatiotemporal object model of [Wor94]. They both represent their geographical entities as collections of 'atoms'. Nevertheless, the *OOgeomorph* model stresses with the notion of forms, processes and materials, the importance of physical considerations.

This model tries to give a non-rigid approach to the spatiotemporal data modeling. One basic assumption, however, is the reference by the 'categories objects' to a 3 dimensional space and 1 dimensional time. This rigid feature allowed to develop formal operations in order to manipulate the phenomena objects, in the case of queries for example. But if it is well suited for representing point information, topological relationships or area information are difficult to model with this choice.

2.2.3 Separation of domains

The previous models have a tendency to group everything in one object. On the contrary, Yuan in [Yua96] and Claramunt and Theriault in [CT95] proposed an approach based on the 'separation of the domains'.

In these two models, semantical – or thematic –, spatial and temporal information are split in three different domain models. Time is no more an attribute of a location or part of a spatiotemporal object. Geographical entities are built by dynamically link elements of the three domains. This allows among other to view reality from location-centered, entity-centered, and time-centered perspectives. There is no predefined arrangement, and it allows to model a wide variety of change types, as shown in table 2.1.

Change Type	Description of scenario	Fixed	Controlled	Measured
1. Fixed	Duration of event or attribute	Location	Attribute	Time
2. Category	For a given point in time certain phenomenon may change its characteristics from site to site	Time	Attribute	Location
3. Static	For a given period of time where attributes may change from site to site through time	Time	Location	Attribute
4. Transitional	For a given event where its characteristics or processes may change at sites through time	Attribute	Location	Time
5. Mutation	For a given area where attributes may change from site to site and from time to time	Location	Time	Attribute
6. Movement	For a given event where its location may change from time to time	Attribute	Time	Location

Table 2.1: Spatiotemporal changes, from [Yua96]

By integrating the concepts of events, sequences, or actions, time-dominant view models are usually more suited to detect causal relationships. Thus, such models will increase the decision support potential of GIS. As a drawback, spatial queries might suffer from the time dominance organization of the data.

2.3 Other approaches and discussions

2.3.1 Design tools emergence

Because of the increasing number of geographical applications and their need for spatiotemporal data, design tools to support modeling of spatiotemporal database are appearing.

MADS [PSZ99] and DISTILL [RSKH01] are two examples of such tools. They are quite similar and rely on the same concept of having orthogonality between objects, space and time domains. They encourage to model first the objects involved in the system to model. Then, some spatial and/or temporal concerns can be added to the objects, their attributes or their relationships, generally speaking to all the conceptual modeling abstractions of the tools. It allows to model a wide range of systems involving spatiotemporal features – they are particularly well suited for modeling moving objects –, and to incorporate them into existing GIS applications or database management systems by means of automatic translations provided by the tools. Thus, even if the tools provide real improvements as regards the simplicity of modeling, the final concrete model generated will be limited by the current GIS and database management systems, in particular for the post management of the data (visualization, queries, etc).

2.3.2 Discrete models

The world we are perceiving is globally continuous. Unfortunately, computing only enables us to implement discrete models. Most of the currently proposed models for spatiotemporal data do not take into account the information loss during the discretization.

Continuous evolution is expressed by highly variable data [YdC95]. The value is potentially different for every quantum of time. We have the choice to take a snapshot of the value for every quantum of time, but then we exceed the computer technology (in space and time), or to only record some states of the value. Then, because of the loss of the continuity information of the data evolution, problems arise when querying the data.

For instance, snapshots models could 'miss' important changes of the world if they occur between two snapshots. But it can lead to more pernicious situations. Let us consider the example of [YdC95] illustrated on figure 2.2. A flock of sheep and a storm are moving. How can we answer to the question "did the sheep got wet by the storm?" if we have only the information on the positions in time T_1 and T_2? In fact, it depends on what happened meanwhile, and how it happened. In other words, it depends on the evolution of the flock of sheep and the storm between T_1 and T_2. Few are the models taking this problem into account while it should be a major concern of spatiotemporal data modeling in order to carry out correct analysis on the data. For instance [YdC95] suggests to add information on the evolution of the entities at each discretization step by means of behavioral functions.

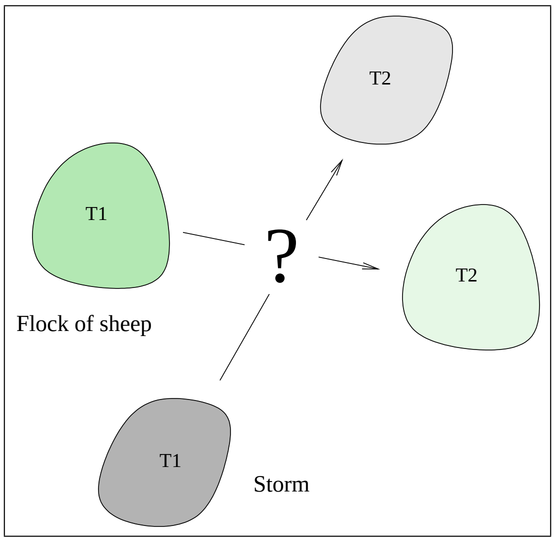


Figure 2.2: Discretization and information loss

2.3.3 Fields vs Entities

Yuan describes in [Yua01] a problem that arises when trying to model complex phenomena like a wild fire or a storm. The majority of the models proposed for representing spatiotemporal data make an assumption on the type of objects they will have to handle: whether fields or entities. And thus, they are generally unappropriate for the other type. But a complex phenomenon can incorporate the two types. A storm for instance is something we can perceive as an entity. It has clear boundaries and position. So, it can be represented with an entity view of geographical objects. On the other hand, what is happening inside the storm – rain, wind – are to be represented with a field approach.

2.3.4 Formal models

Other more formal approaches exist, like [GBE+00] where Güting and al. provide a semantic foundation for handling time dependent geometries. Nevertheless, should their abstract spatiotemporal data type be very well suited for moving objects representation – as they based their framework on them –, it is not flexible enough to deal with the wide variety of types of geographical entities, as most of the others current models. Moreover, as a space-dominant view model, it is less suited for adapted GIS queries.

2.4 Conclusion

There is still a lack in models to efficiently capture dynamic phenomena and objects. New spatiotemporal data models should be flexible enough to adapt to all geographical objects. They must also take into account the discrete view of continuity to provide good bases for analysis. And at last, they should take into account events and evidences, which are critical to ensure data quality and enable good causal relationships detection analysis, in addition to the states and changes that are the main concern in temporal representation.

Chapter 3

Modeling complex objects and phenomena in GIS

This chapter introduces objects for modeling GIS applications, and presents a new approach, based on a new abstraction of what are geographical objects, for modeling complex objects and phenomena in GIS.

Geographical Information Systems intend to model the world we perceive. But this world comprises a wide variety of phenomena and entities, and GIS data models often lack abstractions. They are usually targeted to precise types of geographical objects, and therefore have limited modeling capabilities. Moreover, GIS data models have to deal with the incomplete knowledge we catch from our observations, and few models really provide a flexible structure to cope with the discrete modeling of the world continuity for instance.

Time and space are two elements one can deal with in everyday life. They can be associated to any object or phenomenon. Therefore, they have been given a great – and justified – importance in GIS data modeling. Giving them so much importance, however, restricts the way to observe the world. In current data models, only seems to matter the way to integrate together time and space.

This chapter will show that most of the entities or phenomena we can perceive could be generalized in a common abstract model. It starts with a short introduction to object-oriented programming and its benefits for GIS. It then presents the Model/View/Controller framework, commonly used while developing object-oriented graphical user interfaces, from which we will reuse some principles; before studying how we would model some typical complex phenomena or objects of GIS using an existing object-oriented model. Finally, pointing out some conceptual problems in the example model, it will present a new approach for such modeling.

3.1 Object-oriented approach in GIS

3.1.1 Object-oriented programming

Paradigm

From a philosophical point of view a paradigm is a set of assumptions, concepts, values, and practices that constitute a way of viewing reality for a community that shares them, especially in an intellectual discipline. A programming paradigm can be defined as a conceptual framework which provides a set of tools and concepts to describe a problem domain and to solve it.

The paradigm of object-oriented programming can be formulated as "a set of objects which collaborate by sending messages to each other". There are only objects and they can only send and receive messages.

An object can be defined as an abstraction of an entity of a problem domain. An object presents a well determined behavior – what does the object do –, an implementation for that behavior – how does it do it – and an identity. The behavior of an object is specified through the set of messages it can receive, and its implementation through a set of "methods" and "collaborators". A method is a set of collaboration with other objects (by sending messages).

Each object has a state: its set of internal collaborators. To program in the object-oriented paradigm means to change the state of the objects, that is to say changing the collaborators.

Usual extensions of the paradigm

All the object-oriented languages add something else to the paradigm: the way to create objects. Two main approaches exist:

- Prototype-based approach: any object can create others by performing a kind of cloning. A new object will only need to specify the differences it has with its prototype, and will delegate all their common part to it.

- Class-based approach: one object called "class" will contain all the information about the structure of its "instance" objects. An instance of a class is created using the template definitions of the class. If the instances represent entities in the domain reality, classes usually represent the concepts. Moreover, classes are generally ordered in a class hierarchy: they can inherit from other classes, which means that they will merge the structure information they contain with the one of their superclasses. This relationship can be seen as an "is-a" relationship: a class represents something that "is-a" something represented by its superclass.

The prototype-based approach tends to grab the general concepts by creating particular entities and studying their similarities, while class-based approach requires to tackle the problem by first modeling the concepts to be able to create particular entities.

The semantic gap between object-oriented languages and the reality is narrow: if an industrial process has to be simulated, then there should be an object for each entity involved in the process. Programming in the object-oriented paradigm requires to think in terms of a "hierarchy" of objects and "properties" owned by the objects. It focuses on the data, the objects and the collaborations required to satisfy a particular task.

3.1.2 Using objects in GIS

When observing the world and trying to reason about it, human being is naturally using some categorization mechanisms to represent the knowledge. Looking at a forest, one will get the concept of it as a set of trees, the concept of a tree as a combination of a trunk, branches and leafs, and so on.

This natural process of categorization is very similar to what is happening in the object-oriented approach. Considering a class-based approach, the concept of tree can be modeled as a class. Then, each type of tree adds some particularities to the original concept, but remains however a tree. They can thus be modeled as subclasses of the class Tree. Finally, a particular tree in the real world can be represented by a particular instance of the class in our system. The world we can perceive – and GIS try to model – is full of concepts and entities ready to be used in the object-oriented paradigm.

Working with other paradigms can obviously lead to think in the concepts. But sooner or later, the programmer will think in how to implement a particular behavior for a particular concept, bypassing the discovery process of concepts. It usually results in a system with repeated information or behavior, and mixed concepts, which make the overall system harder to maintain and upgrade. Let us have a look to an example:

- For the implementation of a given GIS, the concept of a forest has been properly discovered and defined. A forest covers a particular surface, and it is a legitimate question to ask whether or not a point belongs to the forest. Thus the programmer of the system implements a procedure – or equivalent – in the scope of the forest code to answer such a query.

- For the same system, the concept of a country has also been pointed out. Asking if a particular city belongs to a country is another legitimate question, and our programmer will implement another procedure – or equivalent – in the scope of the country code to answer the query.

⇒ This is an example of repeated behavior, as well as a lack of concepts. In fact, the region covered by a geographical entity is a concept of its own. Both a forest and a country have the property of having a region, and they should delegate the query of whether a point is inside their boundaries or not to the region concept itself. It allows a more uniform representation of the world. The forest and the country share the concept of a region; they do not implement it in their own and potentially different way.

Another advantage of object-oriented approaches for modeling GIS is the notion of identity. Each object has its own identity. A particular instance of a tree represents a particular tree in the real world. If the tree is cut and used in an industrial process, the object representing the forest and the one representing the industrial process will both collaborate with the same tree instance. This enables the system to naturally know that the tree used in the process is coming from the forest. In the same way for example, if topology is modeled using a vector-based approach, the notion of crosscutting between two paths can be implicitly stored if they share a common point.

Last, but not least, objects enable to encapsulate together data and the operations attached to these data, leading to a strong modularization of the concepts and their implementation. Moreover, concerns about how is actually implemented a concept is no matter for any outside entity. Each object comes with its well defined interface – its behavior –, and the use of this interface to communicate enable to change the implementation without affecting the whole system. An object appears for another as an intelligent and independent entity able to perform the tasks declared in its interface, the apparent intelligence coming from the collaborations performed by the object while dealing with the reception of a message.

With the emphasis put modeling concepts in object-oriented approaches, we are required to pay more attention to the concepts of the problem domain so as to make clear distinctions between them. A Forest is a concept, the Region it covers another, the Trees it is composed of one more, etc. Concepts are less tangled together than compared to others approaches, which allows easier abstractions and thus better reuse, better modularity and models closer to our view of reality.

3.2 The Model/View/Controller model

This section is not directly related to our concerns in modeling complex objects or phenomena in GIS. Nevertheless, the Model/View/Controller approach for graphical user interfaces is an example of a good object-oriented model, and we will be able to reuse some of its concepts for our own problem domain. The Model/View/Controller model (MVC) has been heavily used for the user interface of Smalltalk-80 [KP88]. Should it be a relevant example for describing Design Patterns (because of its use of Composite, Factory, Observer, Strategy patterns) [GHJV94], it could also be seen as a good design practice of its own. Let us consider an example:

Problem: we want two types of graphical user interfaces that enable us to display and modify a set of numbers. The first one should present the numbers in the form of a table of percentages. The user can change the numbers by selecting them, and entering new values. The second user interface should show a bar chart, using the values of the set of numbers. The user can update the numbers by selecting a bar and changing its size.

Naïve solution: before the MVC approach, a solution might have looked like the figure 3.1. An object TableUI would have represented the concept of the GUI for the set of numbers in the form of a table. The object collaborates with a set of percentages. It can display a table containing the values using display(graphicsContext). If the user enters a new value, the object updates the data through the call of the message onValueEntered(at, value). Finally, it also gives third party objects access, by the message getData(), to the data entered by the user.

Another object BarChartUI would have represented the concept of the GUI in the form of a bar chart. The object collaborates with a set of numbers representing the length of the bars. It can display the bar chart through the call of display(graphicsContext). When the user interacts with the GUI using the mouse, the object updates his data thanks to the message onMousePressed(x, y).

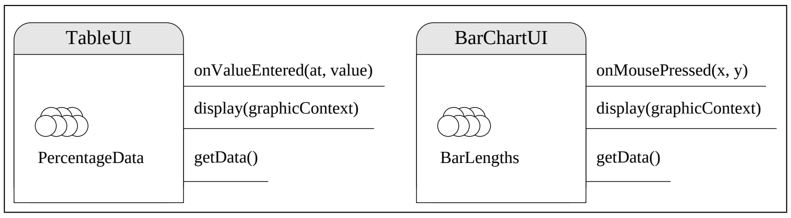


Figure 3.1: Naïve solution for a set of numbers GUI

This solution would work, but some defects can be highlighted. For instance, both GUIs are intended to enable the user to interact on a set of numbers. If we want to allow the user to choose at run-time his favorite GUI, we would need to convert the data from one object to the other, each time the type of interface to display changes.

Better, yet naïve, solution: figure 3.2 tries to solve the fact that both GUIs are applicable on a set of numbers. It abstracts this behavior in a superclass SetOfNumbersUI, responsible for keeping the data and operation on them. The subclasses TableUI and BarChartUI now concentrate their behaviors on the way to display the data and interact with the user.

Unfortunalty, this better solution suffers equivalent problems as the naïve one: coupling of concepts. For instance, how to change the response of a GUI to the user inputs? What if two different GUIs adopt the same policy for treating the user inputs? This concept of how managing the user interactions should be taken apart from the current GUI objects. From this simple example, we can extract the concepts behind a graphical user interface:

- Data: a graphical user interface is intended to visually share with the user some data of the system. These data should not be part of the view concept as their existence does not depend on whether or not we want to show them. The data are encapsulated in the concept of Model in the MVC approach.

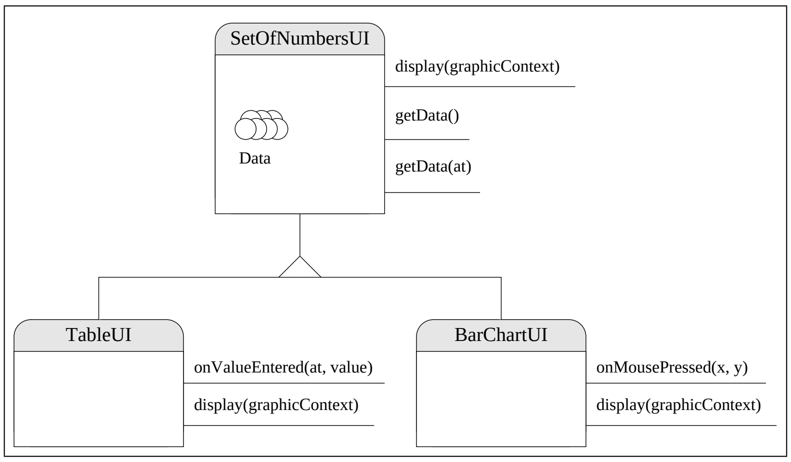


Figure 3.2: Better – but still naïve – solution for a set of numbers GUI

- View: the 'how' to display the data to the user is an idea of its own. We can imagine numerous ways to graphically represent the same set of data. It is logically referred to as the View in the MVC.
- User inputs: how to react to user inputs should also be considered independently from a view. The view is responsible for showing the data, not interpreting the user interactions. In the MVC, this behavior is kept aside in the idea of Controller.

Figure 3.3 shows the conceptual model of MVC. Decoupling views and models enables to attach multiple views to a given model. Views can be added without affecting the implementation of the data handling. A View is responsible for reflecting the current state of the data in the Model. Thus, a mechanism of registration/notification should link a View and the Model. The Views registers themselves as "interested" in the changes of the Model. Each time the Model changes, it notifies the change to the Views that registered. The Views then get the freshly updated data from the Model to update their display.

Decoupling views from controllers lets us change the way to respond to user interactions, without affecting the current visual presentation. Obviously a View will collect the user inputs. But it then delegates the response mechanism to the Controller object, which can update the data in the Model for instance. The strategy of response can be easily swap by changing the Controller associated with a View.

The MVC approach for graphical user interfaces brought design choices to

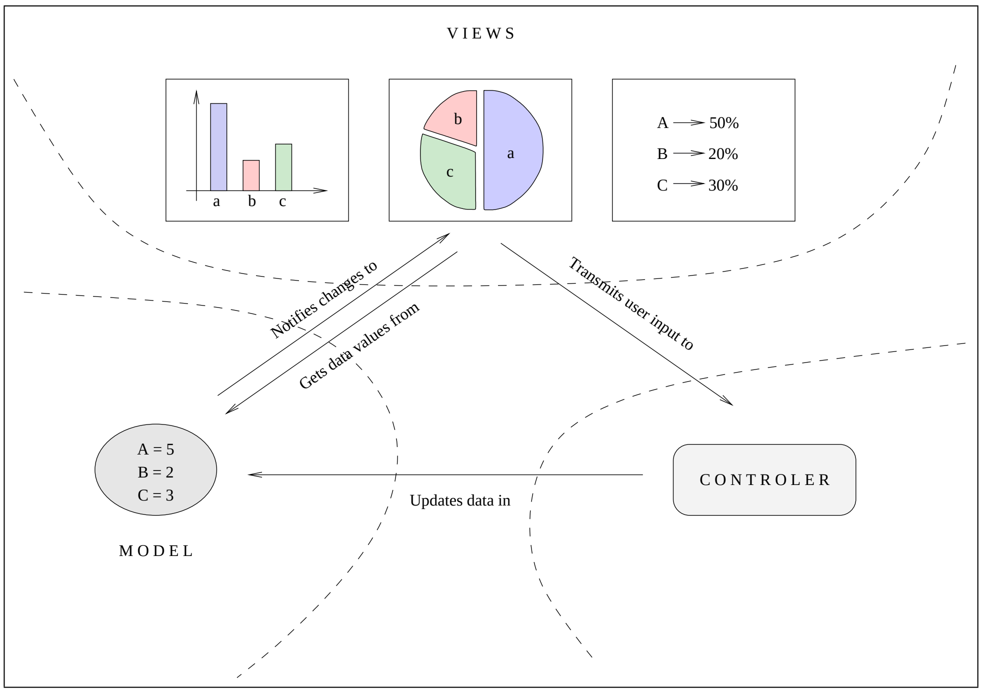


Figure 3.3: the Model/View/Controller model

enable an increase in the flexibility and reuse of user interface components. Moreover, it leads to a more uniform approach: no matter the interface you are designing, it is always a Model, a View, and a Controller which respectively encapsulate the notions of the data, the visible feedback to the user, and the strategy of interaction with the user.

3.3 An example of current object-oriented modeling

We will study in this part an example of what would be a current object-oriented modeling of complex entities or phenomena in GIS – we will call them GIS objects from now on –.

3.3.1 GIS objects examples

A traditional example in literature of a complex GIS object is a land parcel, in the context of cadastral applications. These applications are used to manage the theme of land ownership. Land is divided into parcels which each belongs to someone. The complexity of modeling a land parcel relies on its dynamic properties. A parcel can be sold, implying to keep track of the previous and current owners of the parcel. It can also change in shape, by aggregating other parcels, or by dividing itself into new parcels.

Another famous theme in GIS literature is the moving point. We will integrate it in the example by considering temperature captors. Some of the thermometers will be static in position, while others will be moving – so, acting like moving point –. These captors will measure from time to time the temperature at their current position. Once again, the complexity of these entities comes from their dynamic properties. They must keep track in some way of their position/temperature measures history.

Finally, we will use these captors to model another usual GIS object: a continuous field. A continuous field is composed of a region where the field is applied, some sample points where the value of the field is known, usually by means of measures. As resources are limited, nothing is continuous in computer science, and unless you find a mathematical model you will have to guess unknown values of the field using the sample points. We will refer to this issue as 'the discrete view of reality'. Thus, the continuity of the field is simulated with interpolation strategies.

3.3.2 Current object-oriented approaches

Objects have sometimes been used in GIS modeling but, often, not enough emphasis has been given to the separation of concepts, resulting for instance in mixing conceptual behaviors of entities with their physical representation. Objects have also been used for representing the frameworks that will handle the geographical data, but the concept of GIS object as an individual object is not considered in most of those cases.

The framework proposed in [Gor01] tries, however, to take into account this separation of concepts, and exploits objects so as to model each geographical entity as an individual object.

Quick overview of [Gor01]'s Framework

The research carried out in [Gor01] and [GB98] studies the use of objects and introduces design patterns as a conceptual tool for designing GIS applications. In particular, it studies the example of continuous fields. The model is summarized in figure 3.4.

Entities to be incorporated in a GIS will have as concerns both an appearance and a spatial localization. To avoid tangled concepts and repetition of behavior, appearance and localization have been taken apart from geographical objects and placed into an AbstractGeoObject.

An AbstractGeoObject collaborates with Appearance objects, which are responsible for giving a view of the geographical object adapted to the application needs – media, scale parameters, etc –. The relation AbstractGeoObject↔Appearance is quite similar to the relation Model↔View in the MVC approach.

Each geographical entity has a location, usually described in a particular reference system. To avoid geographical objects to each deal with concerns about translation of coordinates from a reference system to another, or to implement their own geometry objects (to represent points, lines, arcs, polygons, etc), spatial concepts have been isolated from the AbstractGeoObject. The Localization object is composed of a Geometry in a ReferenceSystem.

All GeoObjects are defined as subclasses of AbstractGeoObject. Note the use of the design pattern *Decorator* that enables to add responsibilities to individual objects. The GeoDecorator encapsulates the object to decorate, referring to it for the original behavior, and adds the functionalities needed. The level of enforcement is not class-based but instance-based. It allows to add functionalities in an non-invasive way: instead of creating a new specialization of a given concept, which would mean juggling with instance creation of the original and the more specific type, it enables to reuse all the objects instantiated as general, and to add to them the properties that make them more specific.

As seen before, continuous fields incorporate the concepts of a domain where to apply the field, an interpolation strategy for simulating the continuity, and sample points where the value of the field is known.

The sample points are arranged in what is called in the model a Representation. This abstraction enables to organize the sample in the form of regular grids, irregular grids, Triangulated Irregular Networks (TIN), etc. The ContinuousField object encapsulates a Representation and an InterpolationMethod, which represents the strategy used to calculate the value of the field at unknown points.

3.3.3 Example model using [Gor01, GB98]'s framework

We will see in this part how we could model the GIS objects given as examples in section 3.3.1, using the framework described in [Gor01, GB98]. We will start with the captors as they will be the bases on which we will redesign the continuous field model of the framework. Finally, we will add the land parcels to the model.

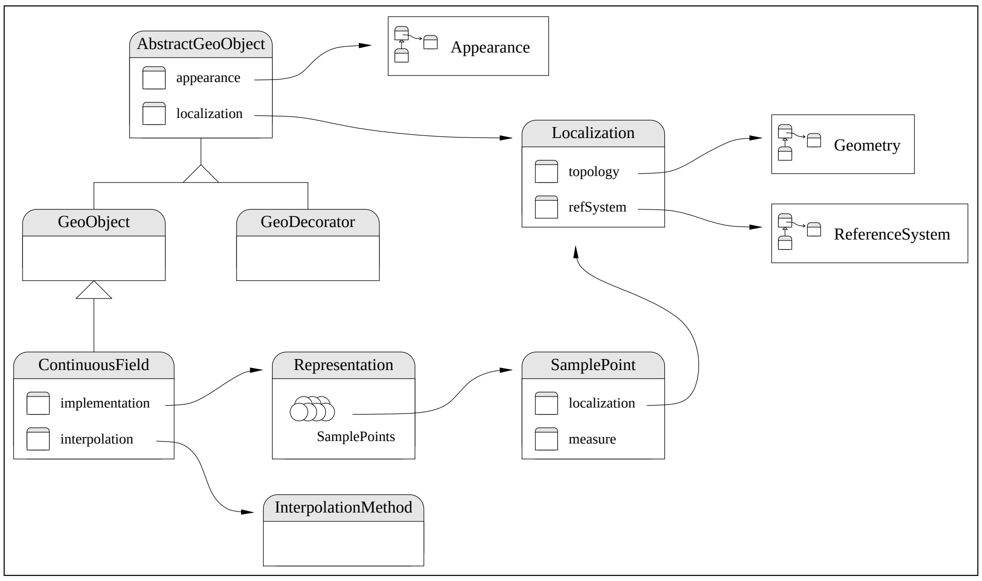


Figure 3.4: Overview of [Gor01, GB98]'s framework.

Captors

As seen previously, a captor is a device taking measures from time to time. It is a concept of the reality of its own; we will model it as a class. A FixedCaptor (see figure 3.5) has a position, and is responsible for giving access to its measures history. Thus, it will keep all its Measures, each of them consisting in the value measured and the conditions of the measurement, in a collection. When receiving the message getSamplePoint(time), the FixedCaptor will return a sample containing its position and the measure done at the given time. A problem arises if there is no measure for the specified time. The captor should guess what it could have been. In order to provide flexibility in the choice of the interpolation to apply, the design pattern *Strategy* [GHJV94] is used. The FixedCaptor collaborates with an InterpolationStrategy, which is in fact the root of an interpolation methods hierarchy. The interpolation strategy used by a captor will depend on the interpolation method actually encapsulated in the InterpolationStrategy it collaborates with.

A MobileCaptor (see figure 3.6) is not so different from a FixedCaptor. They have the same behavior regarding the measures. Nevertheless, a mobile captor does obviously not have a fixed position and, in the same way we are recording the measures history, we keep track of its movement history, in the form of position samples (because of the discrete view of reality). We use the same design, as for the measures, for the interpolation strategy to apply when querying for a position at a time for which no sample is available. The method associated to the message getSamplePoint(time) returns a sample containing the position and the measure done at the time given as a parameter.

We can refactor the FixedCaptor and the MobileCaptor by defining them as subclasses of Captor, which will be responsible for the behavior concerning the measures. The FixedCaptor and the MobileCaptor define the position policy of the captor.

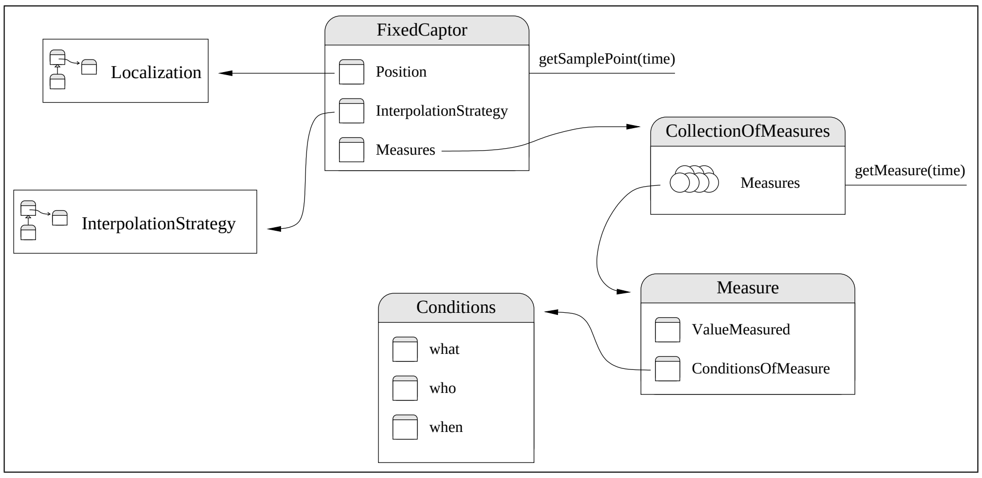


Figure 3.5: Fixed captor

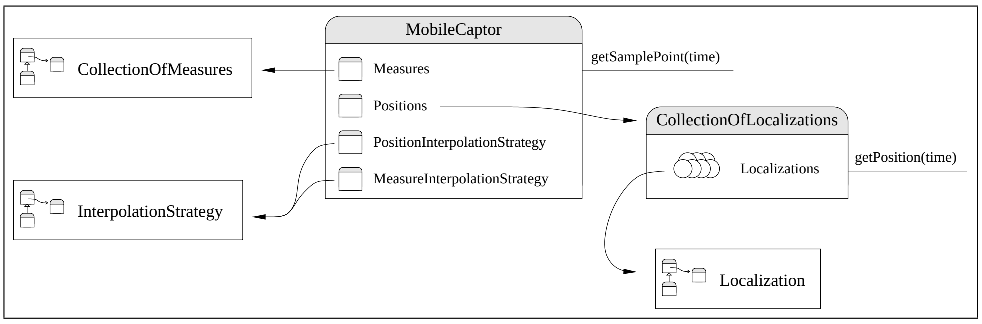


Figure 3.6: Mobile captor

Continuous Field

In [Gor01]'s framework, the continuous fields are based on sample points. In reality however, the samples come most of the time from measures done by captors. Thus, we will integrate our concept of Captor seen previously into the already existing architecture. A ContinuousField will not collaborate anymore with the sample points directly. It will get them from the Captors it will be composed of.

The interpolation strategy to determine the value of the field for unknown positions relies on the Representation object in which the samples are organized, as the interpolation might depend on the organization. Time was not explicitly integrated in the model of [Gor01] and a Representation represented in fact a snapshot of the continuous field sample points for a particular time. As the continuous field can now evolve in time, we need to generate several Representation instances, corresponding to the samples snapshots at the instants we will consider the continuous field. We will encapsulate this generation of Representations in a RepresentationFactory. The RepresentationFactory collaborates with the Captors the field is composed of, to get the SamplePoints from which a Representation can be instantiated. So, when a ContinuousField receives the message getValue(time, position), it gets from its RepresentationFactory a Representation corresponding to the sample points at the time given as a parameter in the query. It then uses its InterpolationStrategy on it to retrieve the value of the field at the given position.

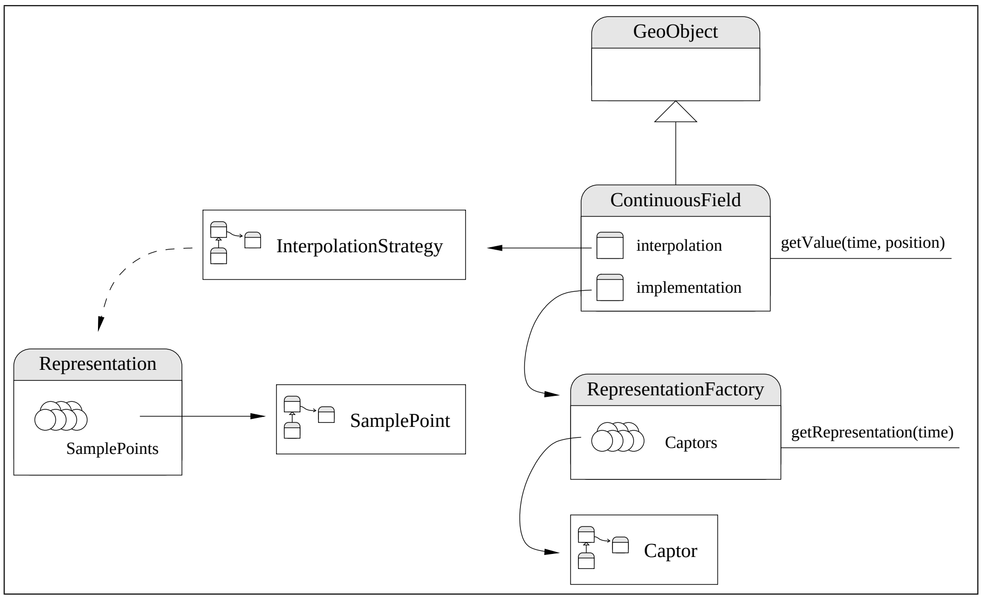


Figure 3.7: Continuous field

Land parcel

We saw in section 2.4 that a model for GIS must take into account events and evidences, to ensure data quality and allowed good causal relationships detection analysis. As they mostly evolve by clearly defined administrative events, land parcels are suitable for this integration.

The concept of land parcel is encapsulated in the LandParcel class. A LandParcel is responsible for giving access to its owner and its border at a given date. In order to record its history, it collaborates with a ParcelHistory, which keeps all the events related to the parcel (see figure 3.8). An Event happened at a given date, and has a description. We can specialize the concept of event to express particular types of events, with additional information. For instance, we can define the four following types:

1. EventNewParcel: generated when a parcel is created. It contains the information about the border of the new parcel.

2. EventNewOwner: generated when the owner of a parcel changes. It contains the information about the new owner of the parcel.

3. EventParcelSplits: generated when a parcel is split in several new parcels. It contains information about all the parcels created from the parcel that owns the event.

4. EventBorderModif: generated when the border of a parcel changes. It contains information about the border modifications, as well as the parcels involved in the change.

To answer a query, the LandParcel will retrieve the information asked within its ParcelHistory's Events. Note that all the changes occurring to a land parcel are discrete. Therefore, we do not need any interpolation mechanisms to simulate continuity. Questions of performance are not taken into consideration. Whether to record for each border change all the new border – for fast access – or only the changes – for low storage – is a matter of application needs for instance. Nevertheless, the LandParcel concept can be decoupled from this issue using the *Strategy* pattern. The LandParcel would delegate the responsibility to interpret the Events, with particular efficiency improvement algorithms, to its ParcelHistory.

Example final model

Figure 3.9 gives an overview of the global model for the three previous examples. Note that this model includes the refactoring of the FixedCaptor and MobileCaptor under a common superclass Captor. Moreover, the original framework of [Gor01] abstracted the localization of each GeoObject in the

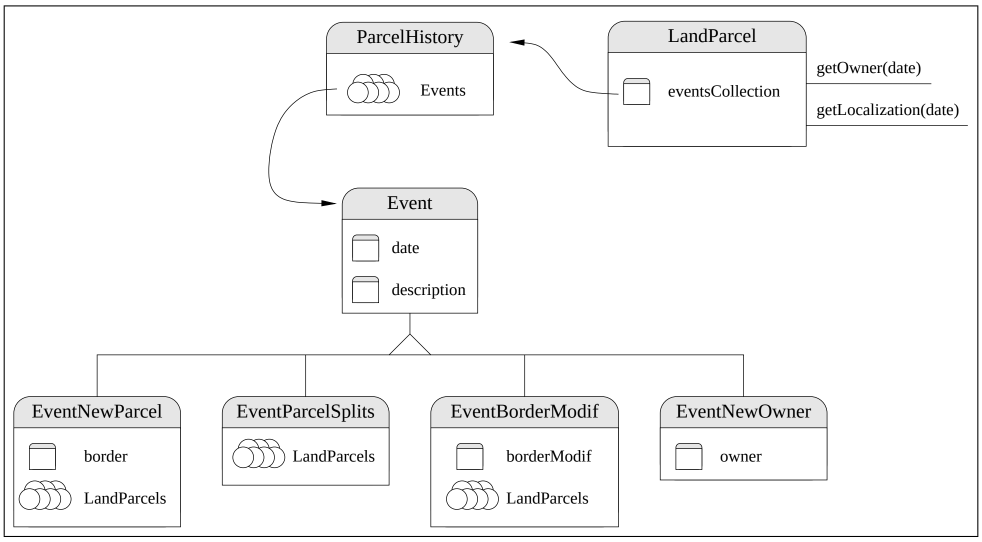


Figure 3.8: Land parcel

AbstractGeoObject. This abstraction has been removed as now, not every geographical object behaves in the same way in regard to shape and position. Some have a fixed localization while others have a changing one.

3.4 Toward a new modeling approach

3.4.1 Example analysis

The example of section 3.3.3 might seem appropriate at first glance. Nevertheless, it suffers from some major conceptual problems.

What is a GIS Object?

Before trying to model complex entities or phenomena, the notion of what was a GIS object in [Gor01] was clear: an entity with a localization and which can be represented, thanks to an appearance, in a GIS. When introducing moving objects – with the mobile captors and the land parcels – the notion of localization is no longer universal for every GIS object. Indeed, every object has a position and shape, but some will need an extra parameter (time or date) to be able to determine it.

A naïve solution to the problem would be to refactor the concept of AbstractGeoObject as two concepts: a MobileAbstractGeoObject and a FixedAbstractGeoObject. This solution, however, is not acceptable. It results for instance in breaking the concept of FixedCaptor and MobileCaptor into two independent concepts for fitting the new hierarchy, while being both Captors. Moreover, if we imagine that the Appearance of a GIS object now depends in some case on some parameters, or if the position of a new type of GIS object does not depend on a time but another parameter (a scenario for instance), we will need to refactor once again our abstract concepts into more sub-concepts, leading to an incoherent overloaded hierarchy of classes.

Introducing the modeling of complex entities or phenomena questions the notion of GIS object. We will need to find what is common to most of them so as to build a new abstraction.

Independent view of concepts

We had three complex GIS objects to model: captors, continuous fields, and land parcels. Each was complex because some of its properties were depending on some parameters – time and position in our case –. We modeled each of them independently, and for each we tried to solve simultaneously the how to define the concept, and to manage its variable properties. It resulted in a high coupling between the concept of variable property and each GIS object. For each, we have found out an individual solution to deal with the changes.

The example tends also to repeat similar concepts in different part of the model. This is the case for instance of events. They are introduced twice: one time explicitly in the LandParcel part, the other implicitly in the Captor part. The LandParcel part manages directly the notion of event in order to store and retrieve the information concerning the parcel's localization and membership. The Captor part uses Measures to store the information it needs. But this notion of measure can in fact be viewed as an event: it occurred at a given time, and has a description (ConditionOfMeasure).

The notion of interpolation methods is also spread over the model: for the value of a ContinuousField, the position of a MobileCaptor and the measure of a Captor. The actual model of the interpolation methods has not been detailed, but it would certainly have ended up with remodeling the concept each time it is used because of the slight differences that might exist for each case.

These two examples point out a problem with the current approach: we are tempted to identify and separate too many concepts, loosing thus abstractions. We want to focus on each, trying to keep it apart from the others, and it ends up crosscutting some of them, or modeling them several times

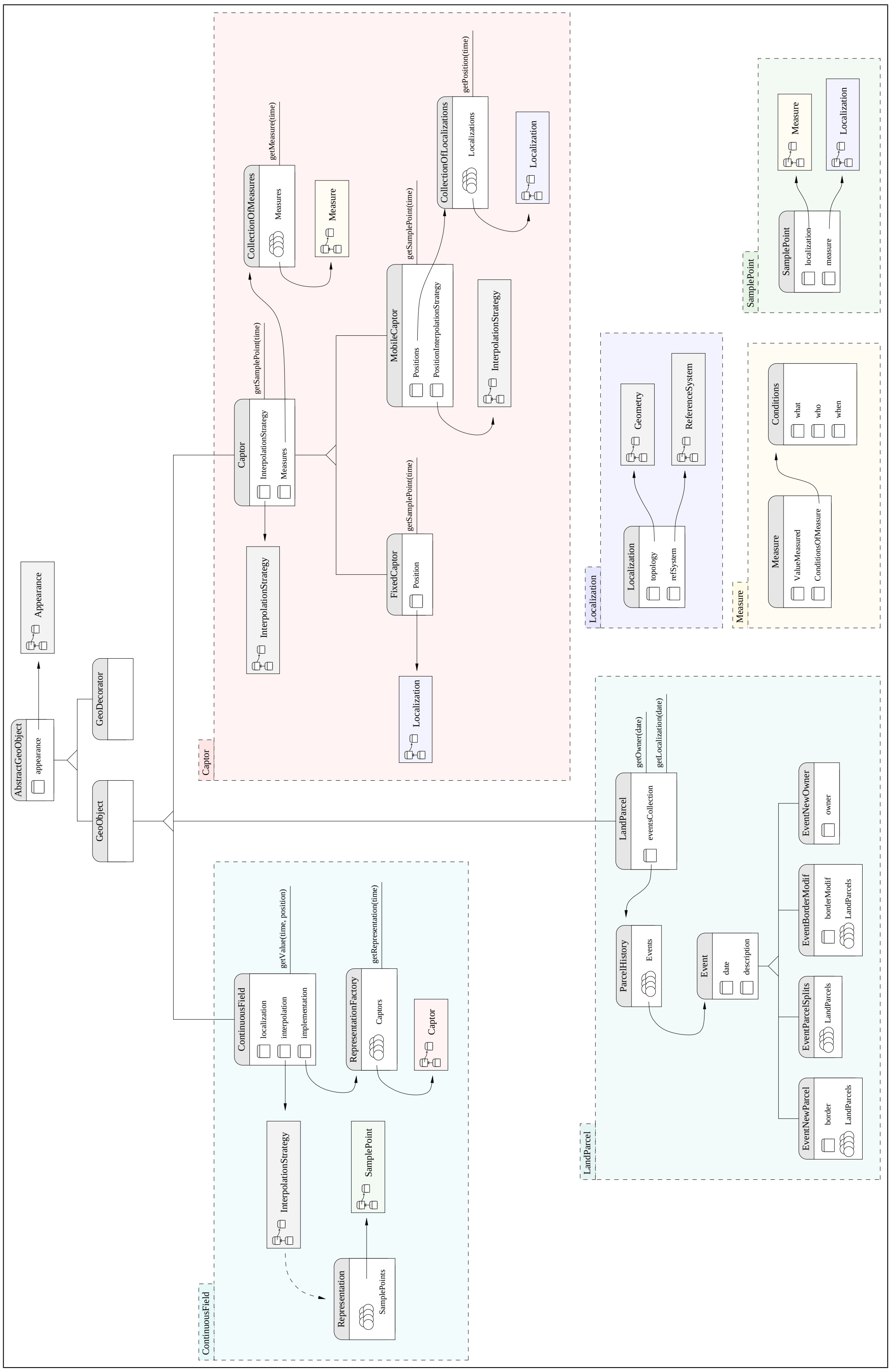


Figure 3.9: Final example model

in different contexts (e.g. for different concepts).

3.4.2 GIS Object: identifying an abstraction

The problems encountered are similar in a way to those encountered while modeling graphical user interfaces before the MVC approach: each interface was thought independently from the others, mixing the concepts of data, view and strategies to deal user interactions. The MVC brought the idea of separating the three concepts for all the graphical user interfaces. Each user interface is then conceptually modeled by the union of a particular view, a particular model and a particular controller.

We saw in section 3.4.1 that we were missing an abstraction of what are GIS objects: we could not find any common attribute among them. This last sentence shows our mistake. We were searching for common attributes among GIS objects while the search was based on an implicit assumption: all GIS objects have attributes. Here lies the first abstraction we can make. It is more important to note that a GIS object has attributes than to note it has the same attributes as this other one.

With the same approach, we can also identify that GIS objects attributes depend on other attributes. For instance, the MobileCaptor has its position and measure attributes depending on a time attribute, the ContinuousField has its value depending on a time and position, etc.

These dependencies lead to the third point we can abstract. A GIS object has attributes and dependencies among them. Which strategy should we use to model the dependencies? Will we use an interpolation approach to represent a continuity in a dependency? Or an event-based approach? Each attribute dependency in a GIS object has to adopt a strategy to link the data actually known and the queries to be done. This will be our third abstraction. We mean by 'query to a GIS object' the fact to retrieve the value of one of its attributes given the values of the argument attributes.

Finally, each GIS object comes with a set of operations to be applied on itself. It was not shown in the example model, but for instance a ContinousField has a set of operations for combining it with other continuous fields (union, intersection, etc).

To conclude with, we have just pointed out four concepts common to GIS objects and which will help us building the GISObject abstraction: attributes, attributes dependencies, dependencies strategies, operations. We will detail them in the next section.

3.5 Domain/Data Controllers/Operations model

In order to model complex GIS objects in an uniform way, we have looked in the previous sections for the key concepts they are all sharing. A GIS object can be separated in three blocks: its domain, the data controllers, and the operations. This section will first present the conceptual model of Domain/Data Controllers/Operations (DDCO), before showing a possible implementation.

3.5.1 Conceptual model

Domain

Every geographical phenomenon or entity to model can be described by its attributes and their respective dependencies. For instance a MobileCaptor can be described as a Position and a Measure that depend on a Time.

$$\begin{cases} \textbf{Attributes}\text{: Position, Measure} \\ \textbf{Dependencies}\text{: Position} \rightarrow \text{Time, Measure} \rightarrow \text{Time} \end{cases}$$

This is the *Domain* of a GIS object, its attributes interface. It describes what the GIS object is sharing, and under which conditions. Every attribute query is done through the *Domain*, which checks for its validity with regard to the arguments needed to compute the result.

Data Controllers

To each attribute of a GIS object is associated a *Data Controller*. Its role is to link the queries done to the GIS object and the data actually known. When a query has been validated by the *Domain*, it is passed on to the *Data Controller* associated with the attribute requested. The *Data Controller* is responsible for computing the result of the query.

The notion of *Data Controller* encapsulates the notions of data storage and data retrieving strategy, which are respectively responsible for storing and giving access to the samples, and retrieving a particular value of an attribute in function of parameters, applying eventually some interpolation mechanisms on the known samples. In the example of the MobileCaptor, the *Data Controller* associated with the attribute Measure might store the measures done with an event-based approach, and retrieve measures using a linear interpolation strategy if there is no sample corresponding to the time attribute given as parameter of the query. Note that a *Data Controller* can use – if needed – other attributes values or operations results.

Operations

The *Domain* and the *Data Controllers* deal with the attributes of a GIS object. The *Operations* part groups all the behaviors of the object not directly related to attributes. For instance, a LandParcel may have an operation to calculate its area. Obviously, the operation might need to get some attributes values of the object, but as any other, it can get them through the *Domain* of the object. The getArea(time) operation of a LandParcel will for example get the Boundary attribute of the land parcel corresponding to the time given as a parameter. It will then delegate the computation of the area to the boundary itself.

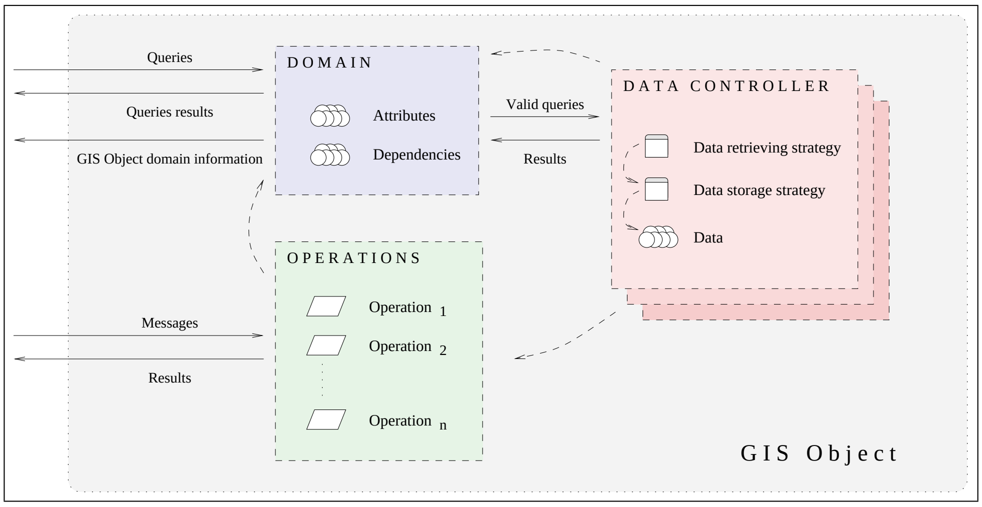


Figure 3.10: DDCO Conceptual model

3.5.2 Implementation approach example

Having the conceptual model of the DDCO approach, we could think of many possible implementations. This part will give an example of implementation approach – Smalltalk based – of the model.

Attributes

We define a class Attribute whose instances will represent particular types of attributes, like a time or a position. An Attribute has a name, and we will check for attributes equality thanks to object identity. They serve as keys to define the attribute a particular object should be used as.

Domain

The concept of domain is encapsulated in two classes: Domain and DomainSpecifications. A DomainSpecifications instance collaborates with a Collection of Attributes, and a Dictionary of dependencies. The Dictionary keys are the Attributes, and the values are Collections of Attributes they are depending on. A Domain instance collaborates with a DomainSpecifications and a Dictionary associating each Attribute with a DataController.

A query to a GISObject goes through its Domain object thanks to the message attribute:getValueAt:, which takes as parameters the attribute to return, and a collection of arguments – in the form of associations between Attributes and values – needed to compute to result. The Domain checks the validity of the collection of arguments against the DomainSpecifications object. If the necessary arguments – defined by the dependency – are present, the query is forwarded to the DataController associated with the queried attribute.

In the same way we can make inherit a class from another, a domain can have a superdomain. It inherits from it its attributes and dependencies. A DomainSpecifications just needs to specify the differences it has with its superdomain, by adding new attributes and new dependencies – possibly by removing dependencies also –.

Data Controllers

Data controllers can take any form. They are triggered by Domain objects with the message getValueAt:for:, which takes as arguments the valid collection of arguments of the query, and a reference to the GISObject instance if they need values of other attributes, or results of the GIS object operations. Note that nothing prevents DataControllers from being composed.

Operations

Operations can simply be encoded as methods of the GISObject.

GISObject

A GISObject is an union of a Domain – which comes with the references to DataControllers –, and operations. The default GISObject class defines a default DomainSpecifications for all GIS objects, and the default operations – for instance, access to attributes through the Domain –. More specific GIS objects will be modeled as subclasses of GISObject. They add or overwrite operations, and define a subdomain of the DomainSpecifications of their superclass.

To instantiate a GISObject means to instantiate:

- all the DataControllers that deal the attributes of the object,
- a Domain based on the DomainSpecifications of the GIS object class, and the associations between Attributes and the DataControllers instantiated previously.

Note that a GIS object class acts as a factory, by keeping the DomainSpecifications of a particular type of GIS objects and the way to initialize them, especially the DataControllers. Nevertheless, each instance of a GIS object can be altered in its Domain and DataControllers. For example, two instances of the same GIS object class can have different types of DataControllers for managing their attributes.

3.6 Conclusion

This chapter has shown how geographical information systems could benefit from object-oriented paradigm. It has also shown that despite their wide variety, geographical entities or phenomena could be looked at with a uniform approach, by separating the notions of *Domain*, *Data Controllers*, and *Operations*. The *Domain* of a GIS object describes the attributes it is composed of, and their respective dependencies. *Data Controllers* are used to link the data actually known by a GIS object and their retrieving strategies for attributes queries. Finally, *Operations* encapsulate the behavior of the geographical objects.

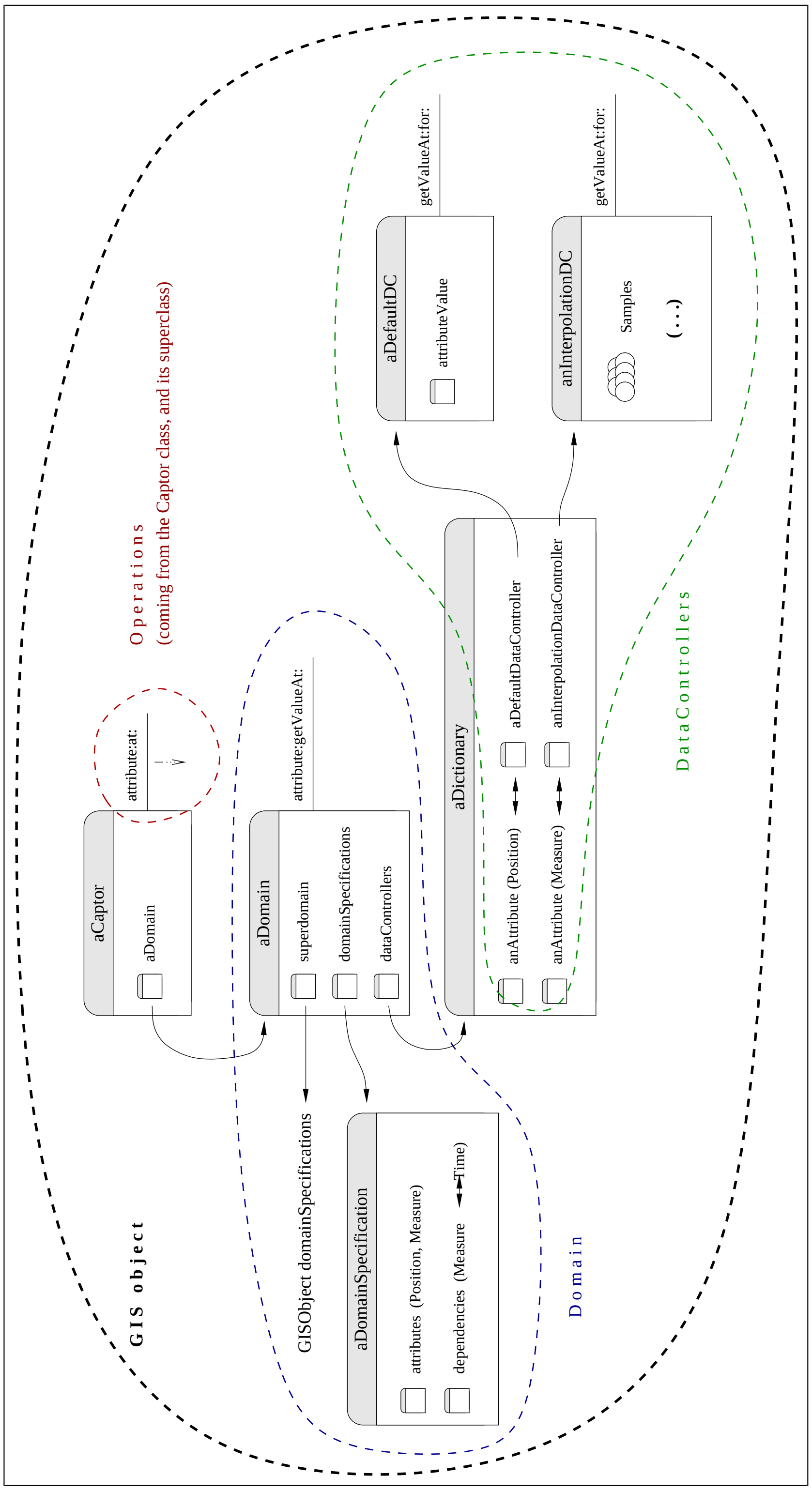


Figure 3.11: DDCO implementation example

Chapter 4

Case study

This chapter will show that, besides its simplicity, the DDCO model is powerful and enables the modeling of complex GIS objects in an intuitive and easy way.

We will base our case study on a simple regional planning management system. The system aims at recording the history of a region in the following fields:

- Population distribution
- Resources distribution
- Transportation facilities
- Farming industry
- Meteorology

We will present in this chapter how we would model, with the DDCO approach, GIS objects to be manipulated by such a system. Note that we called the system 'simple' because we do not have any complex functionalities, and its realism is weak. Nevertheless, it handles some complex objects and phenomena that are hardly used nowadays because of the lack of appropriate models. Note also that the *Domain* and *Data Controllers* are detailed so as to show their use and potential. No much attention is given, however, to the *Operations* part. In fact, *Operations* would mostly depend on the needs and functionalities of the system, but they were not taken into account for the scope of this example.

4.1 Some aspects of the system

4.1.1 Population distribution

We consider that people are living only in the cities. The system keeps track of the cities belonging to the region. A city might appear or disappear. Moreover, during its living period it evolves: the geographic shape is changing with the construction of new blocks or destruction of old ones, and the number of inhabitants is barely constant.

4.1.2 Resources distribution

Wood and water are two traditional natural resources. The system models resources in water thanks to lakes and rivers, and resources in wood with forests. In function of the rainfalls, lakes and river may contain more or less water. Forests size change because of humans interactions: use of the wood, or replanting of the forest with new trees.

4.1.3 Transportation facilities

Rivers and road constitute the transportation facilities of the region. The system records their global statistics of use thanks to periodic measurement of the traffic at key points.

4.1.4 Farming industry

The farming industry is based on land parcels. Each year, a product to farm is assigned to a parcel, and the production results are recorded in the system, as well as the parcels evolution: parcels may change, by aggregation of parcels or when splitting into smaller ones.

4.1.5 Meteorology

The system keeps track of two meteorological phenomena: temperature and storms. Temperatures are recorded with thermometers, which can be fixed or mobile. Note also that thermometers might break down. Information about storms are collected by satellite images. The system makes forecasts of the evolution of storms, in function of different scenarios corresponding to different evolution prediction strategy.

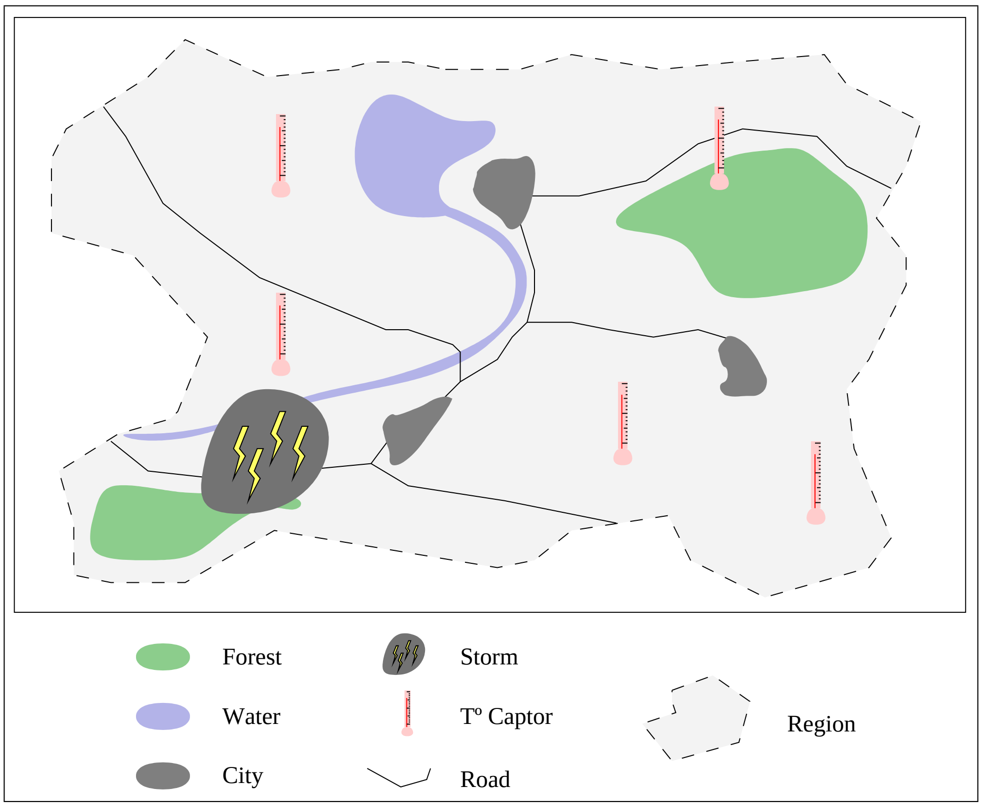


Figure 4.1: A regional planning management system view

4.2 Modeling of GIS objects involved

4.2.1 Domains

From the above specifications, we can define which GIS objects will be involved in the system and their *Domain*. In the following, when speaking about time, date and time is meant.

Position

Every GIS object will have to interact with positions. As a position can be expressed in many reference systems, and in order to define an abstract concept of a Position that can be handled independently from the reference system it was defined in, we create a Position object which domain is:

$$\mathsf{Position:}\quad \begin{cases} \textbf{Attributes}\text{: Values} \\ \textbf{Dependencies}\text{: Values} \rightarrow \text{ReferenceSystem} \end{cases}$$

The values correspond to the coordinates of the position in a given reference system.

Area

Many of the following GIS objects are kind of areas – cities, forests, lakes, region, etc –. An Area is an object with a border delimitating its superficies, and a name. Both are time dependent.

$$\mathsf{Area:}\quad \begin{cases} \textbf{Attributes}\text{: Name, Border} \\ \textbf{Dependencies}\text{: (Name, Border)} \rightarrow \text{Time} \end{cases}$$

City

A City is an Area. It also has a number of inhabitants attribute. A City has a LivingState attribute, which describes its existence period.

$$\mathsf{City:}\quad \begin{cases} \textbf{SuperDomain}\text{: Area} \\ \textbf{Attributes}\text{: NbrInhabitants, LivingState} \\ \textbf{Dependencies}\text{: (NbrInhabitants, LivingState)} \rightarrow \text{Time} \end{cases}$$

Given the definition of domains inheritance in section 3.5.2, the domain of a City has a name, border and number of inhabitants attributes depending on time.

Forest

A Forest is an Area. The Forest domain inherits thus the name and border attribute of the Area domain. It adds to it a number of tree attribute which gives an approximation of the wood capacity of the forest. A Forest has a LivingState attribute, which describes its existence period as for the City.

$$\text{Forest:}\quad \begin{cases} \textbf{SuperDomain}\text{: Area} \\ \textbf{Attributes}\text{: NbrTrees, LivingState} \\ \textbf{Dependencies}\text{: (NbrTrees, LivingState)} \rightarrow \text{Time} \end{cases}$$

Lake

A Lake is a quite simple GIS object – at least for this example –. It is an Area, and it does not add anything to the domain except a LivingState attribute. The quantity of water contained by the lake is estimated with the area of its border attribute.

$$\text{Lake:}\quad \begin{cases} \textbf{SuperDomain}\text{: Area} \\ \textbf{Attributes}\text{: LivingState} \\ \textbf{Dependencies}\text{: LivingState} \rightarrow \text{Time} \end{cases}$$

Farms

A Farm is an area that belongs to someone (the owner attribute). It is also used to produce farming products. Therefore, the domain of a Farm has a production attribute, which indicates the product and quantity farmed a given year. A Farm has also a LivingState attribute.

$$\text{Farm:}\quad \begin{cases} \textbf{SuperDomain}\text{: Area} \\ \textbf{Attributes}\text{: Owner, Production, LivingState} \\ \textbf{Dependencies}\text{: (Owner, Production, LivingState)} \rightarrow \text{Time} \end{cases}$$

Transportation facilities

All the region transportation facilities taken into account by the system are defined by a name, a route and traffic statistics. The route attribute defines the path the transportation facility is going through. The traffic statistics indicate the use of the transportation facility at a given position and time. Name and route attributes are time dependent.

$$\mathsf{TransportationFacility:}\quad \begin{cases} \textbf{Attributes}\text{: Name, Route, Traffic} \\ \textbf{Dependencies}\text{: (Name, Route)} \rightarrow \text{Time} \\ \qquad\qquad\qquad\quad \text{Traffic} \rightarrow \text{(Position, Time)} \end{cases}$$

A Road is a simple TransportationFacility. It does not add anything to the TransportFacility domain, expect a LivingState attribute.

$$\mathsf{Road:}\quad \begin{cases} \textbf{SuperDomain}\text{: TransportationFacility} \\ \textbf{Attributes}\text{: LivingState} \\ \textbf{Dependencies}\text{: LivingState} \rightarrow \text{Time} \end{cases}$$

A River, in addition to the TransportationFacility features, have a flow attribute which indicates the flow of the river at a given time and position. It also has a LivingState attribute.

$$\mathsf{River:}\quad \begin{cases} \textbf{SuperDomain}\text{: TransportationFacility} \\ \textbf{Attributes}\text{: Flow, LivingState} \\ \textbf{Dependencies}\text{: (Flow, LivingState)} \rightarrow \text{(Position, Time)} \end{cases}$$

Storms and Storm forecasts

A Storm is an entity with a clearly defined border, which varies in time. The precipitation degree depends on the position inside the storm and the time. A Storm exists during a clearly defined period. Thus, it has a LivingState attribute.

$$\mathsf{Storm:}\quad \begin{cases} \textbf{Attributes}\text{: Border, Precipitation, LivingState} \\ \textbf{Dependencies}\text{: (Border, LivingState)} \rightarrow \text{Time} \\ \qquad\qquad\qquad\quad \text{Precipitation} \rightarrow \text{(Position, Time)} \end{cases}$$

A StormForecast behaves like a storm, except that all its attributes depend also on the forecast scenario we are considering. Thus the domain of a StormForecast is:

$$\mathsf{StormForecast:}\quad \begin{cases} \textbf{SuperDomain}\text{: Storm} \\ \textbf{Dependencies}\text{: (Border, Precipitation)} \rightarrow \text{Scenario} \end{cases}$$

The precipitation information of the storms are stored in RainfallSatelliteImages. This images record the degree of precipitation in function of the position.

$$\mathsf{RainfallSatelliteImage:}\quad \begin{cases} \textbf{Attributes}\text{: Precipitation} \\ \textbf{Dependencies}\text{: Precipitation} \rightarrow \text{Position} \end{cases}$$

Temperatures field

The field of temperatures of the system is based on the captors taking the measures. A Captor has a position. It also has a measured value that depends on the time. As they can break down, Captors have an OperationalState attribute that depend on the time, in addition to their LivingState.

$$\mathsf{Captor:}\quad \begin{cases} \textbf{Attributes}\text{: Position, Value, OperationalState, LivingState} \\ \textbf{Dependencies}\text{: (Value, OperationalState, LivingState)} \\ \qquad\qquad\qquad\quad \rightarrow \text{Time} \end{cases}$$

Some captors are mobile, but they are still Captors. The difference is that their position is time dependent.

$$\mathsf{MobileCaptor:}\quad \begin{cases} \textbf{SuperDomain}\text{: Captor} \\ \textbf{Dependencies}\text{: Position} \rightarrow \text{Time} \end{cases}$$

The TemperaturesField is applied on a particular border. It is composed of captors from which the field is calculated. Finally, it has a position/time dependent value.

$$\mathsf{TemperatureField:}\quad \begin{cases} \textbf{Attributes}\text{: Border, Captors, Value} \\ \textbf{Dependencies}\text{: (Border, Captors)} \rightarrow \text{Time} \\ \qquad\qquad\qquad\quad \text{Value} \rightarrow \text{(Position, Time)} \end{cases}$$

Region

The Region object is our main object, representing the region to manage. It is an Area, so it has a name and a border. It has also a link to all the GIS objects it is composed of at a given time. They are accessible through the attributes WoodRessources, WaterRessources, Cities, TransportationFacilities, Storms, StormForecasts, TemperaturesField.

$$\text{Region:} \left\{ \begin{array}{l} \textbf{SuperDomain}\text{: Area} \\ \textbf{Attributes}\text{: WoodRessources, WaterRessources,} \\ \qquad\qquad\quad \text{Cities, TransportationFacilities,} \\ \qquad\qquad\quad \text{Storms, StormForecasts,} \\ \qquad\qquad\quad \text{TemperaturesField} \\ \textbf{Dependencies}\text{: (WoodRessources, WaterRessources,} \\ \qquad\qquad\qquad\quad \text{Cities, TransportationFacilities,} \\ \qquad\qquad\qquad\quad \text{Storms, StormForecasts)} \rightarrow \text{Time} \end{array} \right.$$

4.2.2 Data Controllers

We have our GIS object defined with their domain. We now need to associate with each dependency a *Data Controller*.

Non dependent attributes

The attribute position of a Captor and the attribute temperatures field in a Region do not depend on any parameters. We define a simple DefaultDataController which encapsulates the value of the attribute and give access to it.

Discrete changes: an event-based controller

The following dependencies have all in common that they model discrete changes of the reality. If at a time t an Area changed its name, it has the old name for an instant before t, and the new name for an instant after t. The change is instantaneous.

$$\left\{ \begin{array}{l} \text{Area: (Name, Border)} \rightarrow \text{Time} \\ \text{City: (NbrInhabitants, LivingState)} \rightarrow \text{Time} \\ \text{Forest: (NbrTrees, LivingState)} \rightarrow \text{Time} \\ \text{Lake: LivingState} \rightarrow \text{Time} \\ \text{Farm: (Owner, Production, LivingState)} \rightarrow \text{Time} \\ \text{TransportationFacility: (Name, Route)} \rightarrow \text{Time} \\ \text{Road: LivingState} \rightarrow \text{Time} \\ \text{River: LivingState} \rightarrow \text{Time} \\ \text{Captor: (OperationalState, LivingState)} \rightarrow \text{Time} \\ \text{Storm: LivingState} \rightarrow \text{Time} \\ \text{TemperaturesField: Border} \rightarrow \text{Time} \end{array} \right.$$

Such changes usually happened because of a particular event. In order to keep track of the causes of a change, in order to facilitate causal relationships discovery analysis for instance, an event-based approach is chosen for the data controller. An EventBasedDataController will associate Event objects with values of the attribute the controller is associated with. An Event object gathers objects related to the event with the role they play in it. One mandatory object to include is the one used to classify the events, so that the EventBasedDataController can find the appropriate event in order to return the value of the attribute associated with, when a query is done.

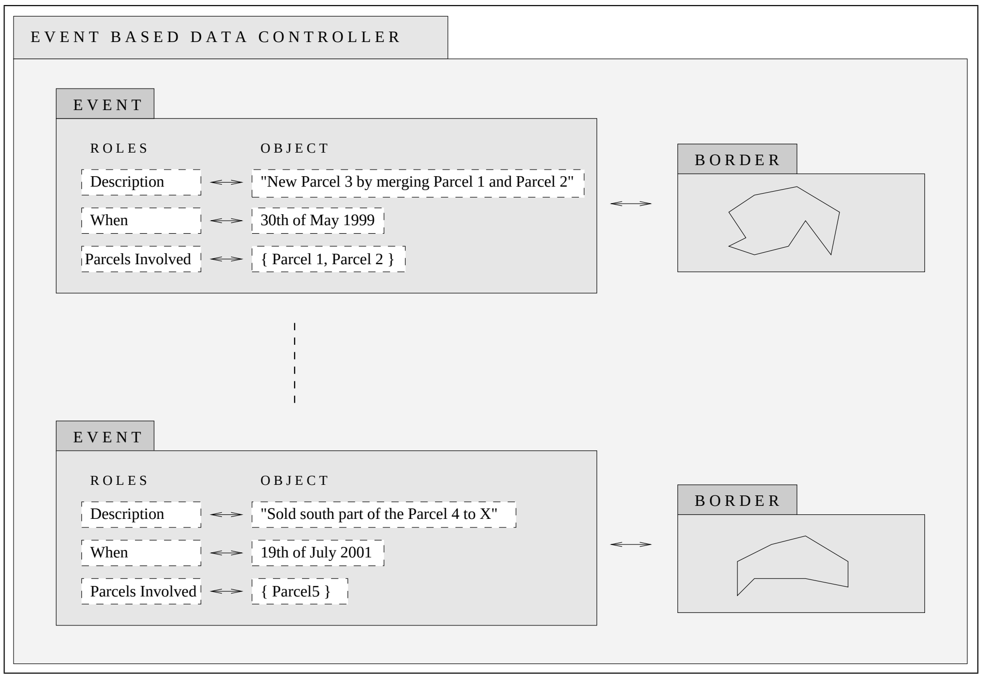


Figure 4.2: Event-based data controller

Continuous changes: an interpolation-based data controller

Dependencies like:

$$\begin{cases}\textsf{TransportationFacility: } \text{Traffic} \rightarrow (\text{Position, Time}) \\ \textsf{River: } \text{Flow} \rightarrow (\text{Position, Time}) \\ \textsf{Storm: } \text{Border} \rightarrow \text{Time} \\ \textsf{Storm: } \text{Precipitation} \rightarrow (\text{Position, Time}) \\ \textsf{RainfallSatelliteImage: } \text{Precipitation} \rightarrow \text{Position} \\ \textsf{Captor: } \text{Value} \rightarrow \text{Time} \\ \textsf{MobileCaptor: } \text{Position} \rightarrow \text{Time} \\ \textsf{TemperaturesField: } \text{Value} \rightarrow (\text{Position, Time})\end{cases}$$

represent continuous changes in reality. Nevertheless, because of the discrete view of reality problem (see section 3.3.1), only some values of the attributes are known and we need to guess the unknown ones using the available samples.

Not every dependencies above will apply the same interpolation mechanism. But most of them get their samples through actions that can be associated to events. For instance, the Captor can store its measures history thanks to an EventBasedDataController, each measure being associated with an event describing the conditions – who, what, when, etc – of the measure.

All these dependencies, except the RainfallSatelliteImage and the TemperaturesField ones, will be managed by a DefaultInterpolationDataController. This object encapsulates an EventBasedDataController, which will be the source of the samples, an InterpolationMethod, which given a collection of samples returns the interpolated value, and a SamplesSelectionStrategy, which determines the relevant samples needed by the InterpolationMethod. Here is one example:

- Captor: Value → Time
    * SamplesSelectionStrategy:
        - Getting the two closest samples in time before and after the time given in parameter.
    * InterpolationMethod: (linear for instance)
        - T: time parameter
        - (T_1, M_1): first sample (time, measure)
        - (T_2, M_2): second sample

      ⇒ Result $= \frac{(T-T_1)(M_2-M_1)}{T_2-T_1} + M_1$

The RainfallSatelliteImage will be based on a raster data model (see section 1.1.2). No event can be attached to the samples it contains – which does not prevent from attaching an event to a RainfallSatelliteImage itself, as used in the Storm –. The data controller for this dependency will work as the previous one, except for getting the samples: it will directly collaborate with internal collaborators representing the raster, rather than with an EventBasedDataController.

Finally, the TemperaturesField: Value → (Position, Time) dependency will work using another attribute. Its data controller will first get the operational captors of the field at the time given as parameter. It will then build from the collection retrieved a Representation. Note that each captor eventually knows its interpolated measure and position for the given time. At last,

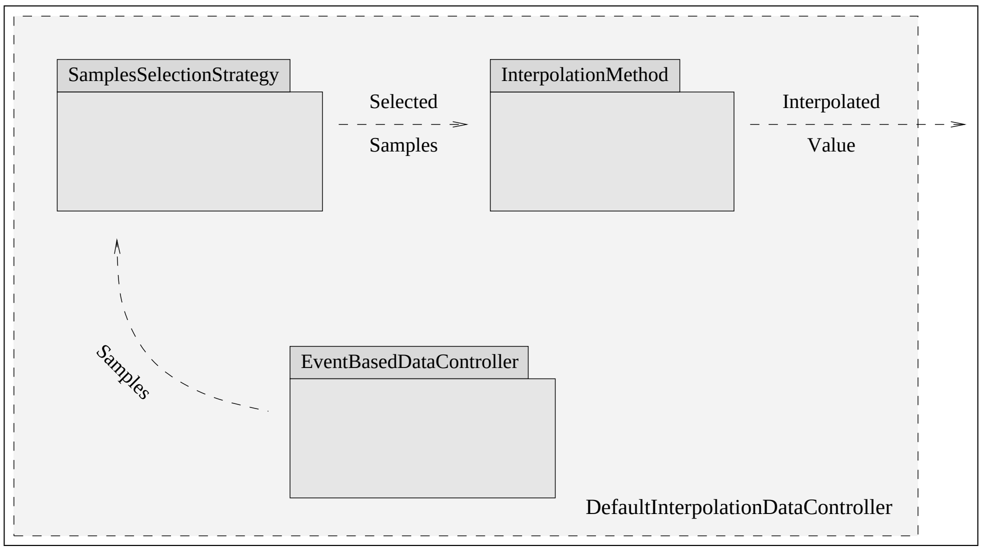


Figure 4.3: Default interpolation data controller

It will use on it an InterpolationMethod to get the interpolated value for the position given as parameter (see section 3.3.3).

Temperatures field captors, Region attributes

The time dependent attributes of a Region, and the captors of a Temperatures-Field are 'Living' objects, which means that they have an existence period. The dependency is there to filter alive objects. When querying for the captors of a TemperaturesField at a given time, all and only the captors that exist at this instant should be returned.

The LivingObjectsDataController collaborates with a collection containing all the values of the attribute. When a query is done, it filters the collection keeping only the objects having a LivingState attribute, for the time given as parameter, of value 'Alive'.

Scenario

The dependencies StormForecast: (Border, Precipitation) $\rightarrow$ Scenario enable to get storm forecasts localization and precipitation in function of different prediction strategies. A ScenarioDataController encapsulates as many Data-Controllers as possible scenarios. When receiving a query, it delegates it to the DataController associated with the scenario given as parameter.

A StormForecast is based on a particular Storm from which it gets the data to extrapolate. We could give access to the Storm to all the data controllers of the ScenarioDataController; or the ScenarioDataController could associate a default scenario to a data controller forwarding queries to the Storm itself. In the latter case, data controllers would access data of the Storm by querying the StormForecast with the default scenario as parameter.

Reference System

The Position object can give access to the coordinates of the position it represents, but the coordinates returned depends obviously on the reference system we are considering.

A ReferenceSystem object is an object that knows how to convert a set of coordinates expressed in another ReferenceSystem object to the equivalent set of coordinates expressed in itself.

A Position is defined by coordinates in a particular ReferenceSystem object. When receiving a query, the RefSystemDataController of the Position will return the result of the conversion – done by the ReferenceSystem given as parameter – of the coordinates/Reference System it stores.

For example, the query TransportationFacility: Traffic → (Position, Time) depends on a Position. With this approach, we can define such position as:

- Coordinate: position on the TransportationFacility, for instance km.73.
- Reference System: a TFReferenceSystem initialized with a time and a TransportationFacility – as km.73 will have a meaning that depends on the Route attribute of the TransportationFacility, which is time dependent –.

4.2.3 Operations

We will give in the section a few examples of *Operations* for our GIS objects. The list is not exhaustive.

One traditional operation, attached to Positions, is the computation of distances. Therefore, we can defined an operation Position distanceWith:aPosition. The algorithm of the operation could be as follows:

1. Getting the coordinates of the position receiving the message in a reference system in which the formula to compute the distance is known.
2. Getting the coordinates of the position given as parameter in the same reference system.
3. Computing the formula of the distance and return the result.

We can think of others spatial operations like:

$$\left\{\begin{array}{l}\textsf{Area getArea}\\ \textsf{Area inside:aPosition}\\ \textsf{Area intersection: anArea}\\ \textsf{Area union: anArea}\\ \textsf{Area difference: anArea}\\ \ldots\end{array}\right.$$

In the same way a Lake would have an operation to estimate the quantity of water it is composed of using its border attribute, a City object might have operations like computing its population density (City populationDensityAt: aTime):

1. Getting the border of the city at the time given as parameter.
2. Getting the area of the border.
3. Getting the number of inhabitants of the city at the time given as parameter.
4. Computing the population density and return the result.

In [Gor01], some continuous field operations are defined. Among them all the operations for combining fields:

$$\left\{\begin{array}{l}\textsf{TemperaturesField intersection: aTemperaturesField}\\ \textsf{TemperaturesField union: aTemperaturesField}\\ \textsf{TemperaturesField difference: aTemperaturesField}\\ \ldots\end{array}\right.$$

Other possible types of operations are the one giving access to attributes of an object in an 'easier' way. For instance, in the case of a captor, an operation to add a new measure sample could take care of instantiating the Event needed by the EventBasedDataController, thanks to some values given as parameters (Captor addNewMeasureSample: aMeasure takenAt: aTime by: aName).

4.3 Conclusion

This chapter introduced a kind of methodological design approach while using the DDCO model. It shows the role of the *Domain*, *Data Controllers*, and *Operations* inside the GIS objects. When modeling a geographical object, we first need to find out what its attributes are, and what they are depending on, defining thus its *Domain*. Then, we 'plug' to each attribute a *Data Controller* – possibly composed of others – adapted to its kind of dependencies, and adapted to the features required by the system. Finally, we define all the *Operations* the GIS objects will handle to fulfill the system requirements.

Note also that the DDCO model could be used to deal with traditional problems of geographical objects representation. For instance, to solve the scale problem when displaying GIS objects on a map, the *Domain* of an object can include an attribute Appareance depending on an attribute ScaleDisplay. The *Data Controller* attached to this attribute would give access to an appropriate object – in function of the ScaleDisplay parameter – responsible for drawing the GIS object.

Chapter 5

Model analysis

This chapter gives a quick evaluation of the DDCO model, with regard to criteria often mentioned in GIS literature.

Thanasis Hadzilacos and Nectaria Tryfona defined in [HT98] some criteria as guidelines to test geographical data models suitability. They picked out five criteria: expressiveness, power of abstraction, complexity, friendliness extendibility. Let us study the compliance of the DDCO model with them.

Expressiveness : "*The more expressive a model is, the more close to the real world application will be, the more semantics will capture.*"

Intended to be used as part of an object-oriented approach, the DDCO model grabs from it a high level of expressiveness. Furthermore, the separation between *Domain*, *Data Controllers* and *Operations* prevents from having the need of an extremely high expressivity to cope with a mix of key concepts of GIS objects.

Power of abstraction : "*One of the criteria to evaluate conceptual models is their ability to represent real world in a highly abstract way. Being able to understand and attribute objects structure without including details allow us to come closer to objects semantics and their role in the application.*"

The DDCO model aimed at abstracting the notion of geographical entities and phenomena. It resulted in having a single conceptual model – independent from the GIS objects to model – which clearly separate the concepts – and their role – they are composed of. All geographical objects are looked at as a union of a *Domain*, *Data Controllers* and *Operations*. Moreover, object-oriented approaches tend toward the discovery of concepts and abstractions when modeling a system.

Complexity : "*The usual trade-off between expressiveness and complexity exists. The more expressive a model is, the more complex appears to be.*"

The *Domain* and *Operations* parts of the DDCO model do not bring any complexity. They are the transcription of our thoughts about the GIS objects. For example a continuous field can be seen as a value depending on a position and a time. It is expressed in the same way by the *Domain*:

$$\begin{cases}\textbf{Attributes}\text{: Value} \\ \textbf{Dependencies}\text{: Value} \rightarrow \text{(Time, Position)}\end{cases}$$

Complexity in the DDCO model appears, however, in the *Data Controllers* part when they need to be modeled for the first time. But as they are easily reusable, this complexity disappears when having a few of them ready to be composed.

Friendliness : "*Friendliness and ease of adaptation/use of a model relates to its complexity. It is considered important, as a powerful – in terms of expressiveness – model may result in a useless model in terms of usage.*"

Someone used to the object-oriented paradigm should not have troubles to use a model such as DDCO. It does not bring any new or complicated notions. Moreover, having separated the concept of a GIS object in three well defined parts guides the user of the DDCO model for modeling geographical entities or phenomena in an easy and logical way, as is doing the MVC model for graphical user interfaces.

Extendibility : "*Another important issue is how easily a model can be extended.*"

Through the concept of *Data Controllers*, the model offers a privileged place for extensions. As an example let us considered the TemperaturesField of chapter 4. We now need to rely on a database for storing the samples of the field. These samples were kept in an EventBasedDataController inside the concept of Captor. By making this *Data Controller* accessing a database in order to retrieve the events – instead of retrieving them directly –, we are adding the functionality needed without changing the global model.

The DDCO model complies well with the criteria defined in [HT98]. Another usual concern about geographical data is to maintain objects identities throughout the evolution in properties and relationships [Yua96]. Such an object-oriented approach naturally solve this issue as the notion of identity of objects is part of the paradigm itself. Finally, GIS need to overcome the difficulty in handling geographic complexity, scale differences, generalization, and accuracy [BF95]. The DDCO model offers the means of dealing with them, especially scale differences which can be seen as dependencies – an Appearance attribute depending on a Scale for example –, and generalization, which was the aim of the model and is an intrinsic concept of object-oriented programming.

Chapter 6

Future work and Conclusion

To cope with the complexity of entities and phenomena that we can encounter in the world and need to model in GIS applications, this work aimed at defining a new object-oriented abstraction of what are GIS objects: a simple union of a *Domain*, *Data Controllers*, and *Operations*.

- The *Domain* describes the characteristics of a GIS object by encapsulating the notion of attributes and attributes dependencies of the object. For instance, a continuous field of temperature can be seen as an attribute Temperature depending on an attribute Time and an attribute Position.

- The *Data Controllers* are the links between queries done to a GIS object and the data it actually knows. In fact, even if a dependency models a continuous change, only samples can be stored effectively. We might apply different strategies to store these samples, to retrieve them, and to guess the unknown ones. Using a kind of *Strategy* design pattern, we associate to each attribute of a GIS object *Domain* a *Data Controller* which will know how to manage the data.

- We used objects so as to represent geographical phenomena or entities as individual objects. We saw that objects were encapsulating together data and operations on these data. Thus, we introduced the notion of *Operations* for a GIS object. It represents all the operations one can apply on it. For instance, if a Position is seen as Coordinates depending on a ReferenceSystem, the Position object might provide an operation to calculate its distance with another Position.

This abstraction brings uniformity in the development of a GIS application. The same mechanisms are used to model every GIS objects. Moreover, it is easier to reuse concepts: as it is explicitly defined, and since its behavior do not crosscut the notion of GIS object, a *Data Controller* can be used by several ones. This works presented a kind of methodology design approach that logically comes from the model abstractions.

Work still has to be done. First of all, the current model lacks a formal definition and representation. Having such a formalism would then enable to build a framework to automatically generate object-oriented GIS application from a formal definition of the GIS objects involved.

One of the crucial point of the model is the *Data Controller* part. All the problems of discrete representation of the reality for instance have been grouped in it. Some research has to be carried out to determine the best ways to cope with highly variable data, in order to model adapted *Data Controllers*, which cope with geographical data accuracy among others.

The *Data Controller* part is also a privileged place to easily integrate new functionalities to a GIS object. For instance, persistence features can be added by swapping a data controller accessing data in a GIS object itself by a DatabaseDataController accessing the same data in a database.

Finally, a significant area to investigate is how to query such a GIS system. In fact, in order to play its purpose of decision making support, one must be able to give a GIS application complex queries so as to retrieve for example causal relationships between phenomena. Once again the *Data Controller*s will have an important role to play in it, as they can for instance associate events information with the data stored to help a causal relationships discovery analysis.

Bibliography

[Arm88] M. Armstrong. Temporality in spatial databases. In *Proceedings of GIS/LIS '88*, pages 880–889, 1988.

[Aro89] Stan Aronoff. *Geographic Information Systems: A Management Perspective.* WDL Publications, Ottawa, Canada, 1989.

[BF95] P.A. Burrough and A.U. Frank. Concepts and paradigms in spatial information: are current geographical information systems truly generic? *International Journal of Geographical Information Systems*, 9(2):101–116, 1995.

[BGLS91] A. Beller, T. Giblin, L. K. Litz, and D. Schimel. A temporal GIS prototype for global change research. In *Proceedings of GIS/LIS '91*, pages 752–765, 1991.

[CT95] C. Claramunt and M. Theriault. Managing time in GIS: An event-oriented approach. In S. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases*, pages 23–42, Zurich, Switzerland, September 1995. Proceedings of the International Workshop on Temporal Databases, Springer Verlag.

[Did90] Michel Didier. *Utilité et Valeur de l'Information Géographique.* Economica, Paris, 1990.

[GB98] Silvia Gordillo and Federico Balaguer. Refining an object-oriented gis design model: topologies and field data. In *Proceedings of the sixth ACM international symposium on Advances in geographic information systems*, pages 76–81. ACM Press, 1998.

[GBE$^+$00] Ralf Hartmut Güting, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and querying moving objects. *ACM Transactions on Database Systems*, 25(1):1–42, March 2000.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Massachusetts, 1994.

[Gor01] Silvia Gordillo. *Modélisation et Manipulation de Phénomènes Continus Spatio-temporels.* PhD thesis, Université Claude-Bernard Lyon I, 2001.

[HT98] Thanasis Hadzilacos and Nectaria Tryfona. Evaluation of database modeling methods for geographic information systems. *Australian Journal of Information Systems*, 1998.

[KP88] G. E. Krasner and S. T. Pope. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, 1(3):26–49, August/September 1988.

[LC88] G. Langran and N. Chrisman. A framework for temporal geographic information. *Cartographica*, 25(3):1–14, 1988.

[PD95] Donna J. Peuquet and Niu Duan. An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data. In *International Journal of Geographical Information Systems*, volume 9, pages 7–24. 1995.

[PSZ99] Christine Parent, Stefano Spaccapietra, and Esteban Zimányi. Spatic-temporal conceptual models: Data structures + space + time. In *ACM International Workshop on Advances in Geographic Information Systems, ACM-GIS'99*, pages 26–33, 1999.

[PT98] Dieter Pfoser and Nectaria Tryfona. Requirements, definitions, and notations for spatiotemporal application environments. In *Proceedings of the sixth ACM international symposium on Advances in geographic information systems*, pages 124–130. ACM Press, 1998.

[RL95] Jonathan Raper and David Livingstone. Development of a geomorphological spatial model using object-oriented design. In *International Journal of Geographical Information Systems*, volume 9, pages 359–383. 1995.

[RSKH01] Sudha Ram, Richard T. Snodgrass, Vijay Khatri, and Yousub Hwang. DISTIL: A design support environment for conceptual modeling of spatio-temporal requirements. *Lecture Notes in Computer Science*, 2224:70–83, 2001.

[Wac99] Monica Wachowicz. *Object-Oriented Design for Temporal GIS.* Taylor & Francis, 1999.

[Wor94] Michael F. Worboys. A unified model for spatial and temporal information. *The Computer Journal*, 37(1):26–34, 1994.

[YdC95] Tsin-Shu Yeh and Béatrix de Cambray. Managing highly variable spatio-temporal data. In *Sixth Australiasian Database Conference*, pages 221–230, January 1995.

[Yua96] May Yuan. Modeling semantical, temporal, and spatial information in geographic information systems. In M. Craglia and H. Couclelis (London: Taylor & Francis), editors, *Geographic Information Research: Bridging the Atlantic*, pages 334–347, 1996.

[Yua01] May Yuan. Representing complex geographic phenomena in gis. *Cartography and Geographic Information Science*, 28:83–96, 2001.