# Bytecode unification of geospatial computable models

*by Jan Kolář*

*Grifinor Project*. jan.kolar@grifinor.net

## Abstract

Geospatial modelling revolves around the structures of data and the semantics of these structures. This is enough in simple cases, but becomes insufficient when the best structure and semantics is hard to find or the solution is too heterogeneous to fix and reuse. Field-based and objects-based geospatial models often share common GIS data structures interchangeably, but their all possible meanings are too many to define in an immutable manner. Less studied approach to geospatial modelling is using mutable structural properties and their semantic interpretation. This work shows that the functional aspect of geospatial models is just as important as the structural and semantic aspects. It also shows that semantic and even structural properties may change when functionality is integral part of the data model, and not exclusively separated at software implementation level. The paper uses this modelling paradigm to address the divide caused by field-based and object-based data models, and other challenges regarding synergy of geospatial systems that need to use both types of data models.

**Keywords:** field-based model, object-based model, computability, managed objects, geographic space, 3d, time, scale.

## 1 Introduction

The geospatial branch of information science (GISc) has a dual approach to representing the real world. Its duality is manifested by 1) discrete objects, which are identifiable, countable entities existing in otherwise empty geographic space, and by 2) continuous fields, which for every location in a given domain of geographic space have a value forming a field of certain quantity. The twofold nature of the object / field modelling is a concrete, classical problem that has been eluding specialists in GISc since 1980 (Goodchild, 2012). This work studies the comparative suitability of object-based and field-based representations for computable geospatial models. The goal is to provide a theoretical and engineering solution supporting both field- and object-based geospatial models in a uniform way. This is the scope depicted in Figure 1. The problem in principle is that the solution must provide a better modelling flexibility without hindering simplicity and applicability in practice. Due to these unmet requirements the solution to duality of fields and objects might also address number of other practical issues, such as dealing with heterogeneity of models across many application domains, design and management of complex geospatial models, or difficulties with handling and exchange of the models by various information systems.

Computability is essential for this work since computable geospatial modelling is the main subject, but it is also important in a much broader sense because computable models can do work that humans would have to do otherwise. The rest of this section provides a necessary introduction to the computability theory. This gives a basis to Section 2 where the method for geospatial modelling is formalized and the engineering design presented. Section 3 reports on implementation related topics and on the resulting technology.

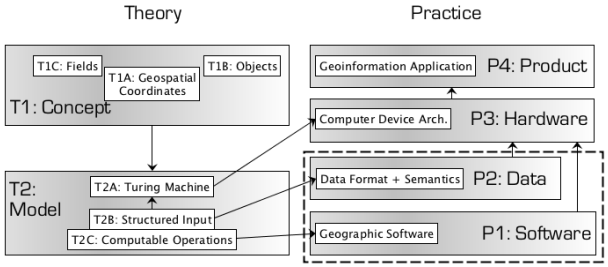


Figure 1: Computable geospatial modelling in context.

### 1.1 Computable models

This work deals with geospatial models that are *computable*. Computable in the sense that humans could also calculate the model manually. This means that the set of instructions followed to carry out a computation must be finite, that the computation carried is real - not imaginary (must be effective), and it must be possible to show how exactly the computation is performed (must be constructive). Exactly like in a program or a recipe.

The theory of computation is based on the concept of Turing machine. It implies, for example, that any finite set of Turing machines can be represented by a single one. Turing machines define a unique and natural class of "computable", which is fundamental for this work, but also for all computer science and mathematical logic.

Therefore, according to Sipser (1996) any computable model $M$ can be formalized as Turing machine by a six-tuple:

$$M(\mathbf{Q}, \mathbf{\Sigma}, \delta, q_0, q_A, q_R) \tag{1}$$

where: $\mathbf{Q}$ is a finite set of all possible computational states of the model; $\mathbf{\Sigma}$ is the alphabet used to describe any possible instance of the model; $\delta$ is the *transition function* that tells how the model is evaluated between two consecutive steps; $q_0 \in \mathbf{Q}$ is the start state; $q_A \in \mathbf{Q}$ is the accept state; and $q_R \in \mathbf{Q}$ is the reject state.

The central element that makes the model "work" is the computable function $\delta$ also called as algorithm.

$$\delta : \mathbf{Q} \times \mathbf{\Sigma} \quad \rightarrow \quad \mathbf{Q} \times \mathbf{\Sigma} \times \{L, R\} \tag{2}$$

Given a *model instance* $\mathbf{I}$, which is an input string over alphabet $\mathbf{\Sigma}$, the computable function $\delta$ of $M$ works as follows: for every valid state $q \in \mathbf{Q}$ and input character $c \in \mathbf{\Sigma} \wedge c \in \mathbf{I}$, the function returns the next state, writes (outputs) a character, and moves on reading the next character to the left $L$, or to the right $R$, from the current position in the input string $\mathbf{I}$. This continues forever unless $\delta$ yields the accept state $q_A$, or the reject state $q_R$.

Therefore the input is a model instance $\mathbf{I}$, which should not be confused with the model $M$ itself. The $M$ refers to a more general solution of the modeled phenomena. In contrast, the model instance $\mathbf{I}$ is a rather concrete utterance for which $M$ computes an output.

For example, consider a model $M_A$ deciding a "distance between two points on Cartesian plane". The model instance (input) is a pair of Cartesian coordinates, e.g. $[0, 0]$ and $[2, 0]$, and the decision (output) for this instance is $2$, which is the distance between the points. A different model $M_B$ would be necessary for "rendering the line between two points on a display". While the instances could be reused from the previous example, the transition function $\delta$ and output would be different in this case. Also, the model $M_B$ may depend on yet another model, for instance representing the display. Such model hierarchy is common in practice and often gets complex.

The computable model can be understood as a compact collection of the problem-solving model $M$ together with one or more distinct instances $\mathbf{I}$ solvable by $M$. Considering and treating computable model this way as a single, compact unit provides a key postulate in the modelling method suggested in Section 2. The content unit based on this view of $M$ then has the ability to store not only the data $\mathbf{I}$ and the intermediate results of the computation, but also to store the transition function $\delta$ that brought about the computation. In Figure 1 is such compact unit depicted as *T2*, in which *T2B* would correspond to the model instance $\mathbf{I}$ and *T2C* to the transition function $\delta$. The introduction of established computability theory formulated by Equation 1 is essential for addressing the concepts of space, fields and objects in the geospatial modelling method.

# 2 Method

A new method for geospatial modelling is based on the equivalence of finite composition of computable models $M$ mentioned in Section 1.1. This composition equivalence can be expressed as:

$$M = (M_1, ..., M_n) \mid n \geq 1 \tag{3}$$
$$\delta = (\delta_1, ..., \delta_n) \tag{4}$$

Note that $\delta_1$ is the transition function of $M_1$, $\delta_n$ is the transition function of $M_n$, and therefore $\delta$ is the function(s) of the computable model $M$. The principle of Equation 3 is applied to a universal geospatial model supported by the definition of geographic space.

## 2.1 Geographic Space

Every information system is constituted by a unique space and by contents associated with that space. This is true of any space. Not only abstract spaces used in database systems for querying and indexing of data, but any space with certain order, can be used as a basis for an information system. This includes the representation of the real-world space where the Earth exists. The conception of contents here are the computable models and their instances, but the space must be defined first. Space is critical for an information system because points of the space are the means of accessing the content. For example the World-Wide-Web (the Web) employs a non-topological, URL address space, which makes it possible to access a concrete Web page. Without the URL space the Web cannot exist.

The idea of devising a unique definition of space for all geospatial modelling is easy to express, but

a good, general solution is missing. The definition of geographic space must sufficiently correspond to physics, but at the same time must facilitate the nature of the computable content. Modern geoinformation applications utilize many different spaces with two, three, and in rare cases with four and more dimensions resulting in more than thousand different coordinate systems used in practice (EPSG, 2013). The arguments for unique geocentric space are well known (Burkholder, 2000, Kjems & Kolar, 2005), but a definition that supports various geospatial models and that can be adopted by a broad variety of applications at different dimensions and scales still remains to be found.

The majority of geospatial applications can use the Newtonian description of space to obtain values that are correct to a sufficiently high order of accuracy. The realization of the geocentric reference frame, however, uses astrometry operating at the angular resolution exceeding one milliarcsecond, or atomic clock ticking at nanosecond level and moving on high precision orbits. Modelling events at such precision level, high speeds and gravitation differences requires consideration of general relativity (Kopeikin, 2007), and relativistic corrections must be used. Neglecting the relativistic curvature of space in these cases would degrade the observed measure (Pogge, 2013).

The *geographic space* is represented through a reference system, which relates coordinates of points existing in reality to a unique and common basis for computational geospatial models. The coordinates of the geographic space $\mathbf{S}$ have six-dimensions:

$$\mathbf{S} \quad = \quad \mathbf{X} \times \mathbf{Y} \times \mathbf{Z} \times \mathbf{T} \times \mathbf{\Gamma} \times \mathbf{\Omega} \tag{5}$$

The space $\mathbf{S}$ is defined close to the Earth and is co-rotating with it. Following the Newtonian physics the space is considered as Euclidean using Cartesian right-handed coordinates $\mathbf{C}$ with the same unit used for each dimension.

$$\mathbf{C} \quad = \quad \mathbf{X} \times \mathbf{Y} \times \mathbf{Z} \subset \mathbf{S} \tag{6}$$

The origin is close to the Earth's center of mass (geocenter), the orientation uses $\mathbf{X}$ and $\mathbf{Y}$ coordinates to represent the equatorial plane and the $z+$ axis is the direction of the north pole and of the Earth's rotation axis. In addition to Cartesian coordinates, other coordinates, e.g. geographical coordinates, could be used (Boucher, 2001). The temporal coordinates $\mathbf{T}$ are associated with the time running at the reference geopotential level (geoid) of the Earth.

It was mentioned that the coordinates are the means of accessing contents. However, due to the continuity of the Euclidean space, where all points are topologically relevant to each other, obtaining geospatial models for a given coordinates leads to a progressive loading of contents from the entire space (all the contents). Following the principle that closer things are more relevant than distant things the geographic space $\mathbf{S}$ has two *proximity dimensions*. The proximity dimensions in Equation 5 are *time proximity* $\mathbf{\Omega}$, and *spatial proximity* $\mathbf{\Gamma}$.

The spatial proximity $\mathbf{\Gamma}$ represents the extent of geometric neighborhood, and indicates a relative geometric magnitude within the space $\mathbf{S}$. Given a position $p \in C$ the spatial proximity $\mathbf{\Gamma}$ allows to decide the range of relevant neighborhood for models of various geometric magnitude (size) on the scale from the largest to the smallest geometric magnitude. The *spatial proximity level* $\gamma = 0$ is for models of the largest geometric magnitude, $\gamma = 1$ is for smaller and so on. The levels of spatial proximity are obtained by a recursive octant subdivision (Weisstein, 2103) of the Euclidean space $\mathbf{C}$. In order to perform computationally such octant subdivision bounds for the ranges of coordinates $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ must be specified. Choosing the same symmetric range for all three coordinates implies that the shape of the geometric space $\mathbf{C}$ is a cube, and its center coincides with the geocenter. This "root" cube also corresponds to the spatial proximity level $\gamma = 0$.

Analogously the time proximity $\mathbf{\Omega}$ is a dimension that facilitates decision about which models are relevant for a given instant on scale ranging from the shortest-term duration to the longest-term duration. The levels of time proximity are obtained from the interval of all time coordinates $\mathbf{T}$ by its binary recursive subdivision.

## 2.2 Geospatial Models

A computable geospatial model has a concrete formulation, but it is an abstract entity that can be applied to an arbitrary more concrete geospatial model. These models can be object-based, field-based, or complex including models combining both approaches. All computable geospatial models are defined in the geographic space $\mathbf{S}$ introduced in the previous section and carry several common properties. With consideration of Equation 3 the computable geospatial model $G$ is expressed as a three-tuple:

$$G \quad = \quad (M_{sdx}, M_{ref}, M_{fun}) \tag{7}$$

The key part is the *functional model* $M_{fun}$, which can represent any computable model, again by fol-

lowing the concept behind Equation 3. While traditional solutions strictly separate the functionality $\delta$ from data $\mathbf{I}$ this method, in contrast, allows structures of data together with functionality. Together they form the content that is exchanged between applications. Although this perspective is common to programmers it is unfamiliar to content providers who store and exchange data. The traditional separation is depicted in Figure 1 by $P1$ and $P2$. Using this method means fusion of $P1$ and $P2$ into a single entity as denoted by the dashed line in Figure1. This is a significant concept and a key to the uniform approach to object-based and field-based geospatial models.

The second term in Equation 7 is the *reference model* $M_{ref}$, which provides values that reference the model $G$ in the geographic space $\mathbf{S}$. There are three types of reference values:

$$P_{ref}(x,y,z) \in \mathbf{C} \mid P_{ref} \in M_{ref} \quad (8)$$

$$\gamma_{ref} \in \mathbf{\Gamma} \mid \gamma_{ref} \in M_{ref} \quad (9)$$

$$T_{ref}(t_{start}, t_{end}) \mid t_{start} < t_{end},\ T_{ref} \in M_{ref} \quad (10)$$

The reference point $P_{ref}$ specifies Cartesian coordinates with which the model $G$ is associated. The spatial proximity $\gamma_{ref}$ indicates the metric size of both the model and its spatial neighborhood. Every model in the geographic space $\mathbf{S}$ exists for a limited period of time bounded by temporal coordinates $t_{start} \in \mathbf{T}$ and $t_{end} \in \mathbf{T}$. The value $T_{ref}$ represents these *temporal bounds*. The only mission of $M_{ref}$ is to return these reference values whenever needed.

### 2.3 Geospatial Index

According to Equation 7 each geospatial model $G$ includes *the model of geospatial index* $M_{sdx}$. The geospatial index has two components with close relationship to the spatial proximity $\mathbf{\Gamma}$, and to the time proximity $\mathbf{\Omega}$ introduced in Section 2.1. They facilitate several properties common for all geospatial models. The spatial index $c_{sdx}$ maps from $P_{ref}$ and $\gamma_{ref}$ to a set of *index coordinates* $\mathbf{C}_{sdx}$ by:

$$c_{sdx} : \mathbf{C} \times \mathbf{\Gamma} \rightarrow \mathbf{X}_{sdx} \times \mathbf{Y}_{sdx} \times \mathbf{Z}_{sdx} = \mathbf{C}_{sdx} \subseteq \mathbf{C} \mid c_{sdx} \in M_{sdx}. \quad (11)$$

Every $\mathbf{C}_{sdx}$ is a subspace of the Euclidean space $\mathbf{C}$ in such a way that it coincides with some octant generated by the subdivision of the spatial proximity $\mathbf{\Gamma}$. Given a spatial proximity level $\gamma = k \in \mathbf{\Gamma}$ there are $8^k$ distinct octants denoted as $\mathbf{C}^k_{sdx}$. All $\mathbf{C}^k_{sdx}$ have equal size, never intersect each other, and their sum fills the entire space $\mathbf{C}$. It is assumed that the reference point is one of the index coordinates $P_{ref} \in \mathbf{C}^k_{sdx}$. The signature of each octant $id(\mathbf{C}_{sdx})$ is used for a redundant indexing structure that facilitates rapid access to spatially relevant models and can also provide a paging mechanism for the access. Because $\mathbf{\Gamma}$ subdivision that generates $\mathbf{C}_{sdx}$ is space-driven rather then content-driven the indexing is applicable in distributed, decentralized information systems. When considered as local coordinate system $\mathbf{C}_{sdx}$ can be also utilized for more data-efficient geometry representation because fewer significant digits can be used. In this sense $\mathbf{C}_{sdx}$ also facilitates visual rendering and multiple levels of resolution because utilizing a fixed number of significant digits over large range and over smaller subsection leads to different resolution over these spatial domains.

The temporal index component is represented by function $t_{sdx}$ that maps the temporal bounds $T_{ref}$ to a unique interval of the linear time coordinates $\mathbf{T}_{sdx}$ as follows:

$$t_{sdx} : \mathbf{T} \times \mathbf{T} \quad \rightarrow \quad \mathbf{T}_{sdx} \subseteq \mathbf{T} \mid t_{sdx} \in M_{sdx}. \quad (12)$$

The signature $id(\mathbf{T}_{sdx})$ is utilized for fast access to temporally relevant models, which is an analogy to the use of $id(\mathbf{C}_{sdx})$ for indexing purposes. Because the Euclidean space $\mathbf{C}$ and the time $\mathbf{T}$ are independent dimensions of the geographic space $\mathbf{S}$ it is convenient to combine $c_{sdx}$ and $t_{sdx}$ in the single indexing model $M_{sdx}$.

#### 2.3.1 Geospatial Object

An object-based representation of any geographic feature that is computable can be described using the geospatial model $G$ expressed in Equation 7, and is called computable *geospatial object*. As used in the introduction, "object-based" is meant as a contrast to the field-based representation of geographic features, which will be addressed in the next section. It must be stressed that an abstract word "object", especially when computer science and engineering is involved, can easily lead to a confusion. The context to which the term "object" is related must be clarified. Here the geospatial object refers to an object related to the geographic space $\mathbf{S}$. We will see in Section 2.4 that the geospatial model itself can be called as object, but in context of an engineering solution to computable models - hence completely different type of object compared to the geospatial object.

The simplest geospatial object can be described by Equation 7 in which the functional model $M_{fun}$

does nothing. Such model of geospatial object provides the reference values $P_{ref}$, $\gamma_{ref}$, $T_{ref}$, and through the spatial index $c_{sdx}$ and temporal index $t_{sdx}$ they also have access to their proximity coordinates $\mathbf{C}_{sdx}$ and $\mathbf{T}_{sdx}$. If any of the reference values are missing then either geospatial model $G$ cannot be made, or appropriate default values must be specified or generated by the application.

Any geospatial object can, however, through the functional model $M_{fun}$ use other models arbitrarily, for instance dealing with various geometric types, topological operations, exchange data formats and other subjects as depicted in Figure 2. Note that the geospatial model is not only an entity in the system diagram, but also a content unit that can be stored and exchanged. The relatively simple representation described by Equation 7 is used, regardless of the specialization or complexity behind a particular geospatial object, . If an instance requires any specialized model that is part of $M_{fun}$ it will be provided by the geospatial model $G$. That provides almost arbitrary flexibility to geospatial modelling while keeping the actual computable geospatial model $G$ relatively simple for adoption by broad variety of applications. All the properties of geospatial objects described in this section also apply to the field-based models described in the following section.

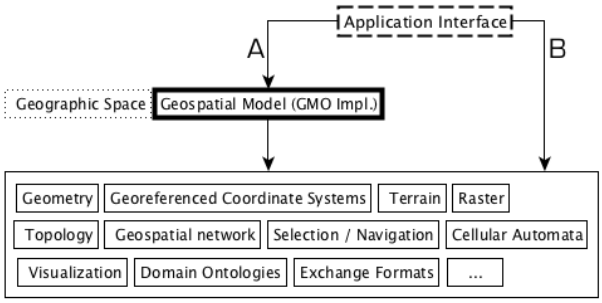


Figure 2: Geospatial model in relationship to the application interface.

### 2.3.2 Geospatial Field

A field-based representation of any computable geographic feature can be described using the geospatial model $G$ from Equation 7, and is called computable *geospatial field*. The functional model $M_{fun}$ of every geospatial field is characterized by a function $f$ that maps from a spatial domain $\mathbf{D}$ to a field $\mathbf{V}$ as follows:

$$f : \mathbf{D} \rightarrow \mathbf{V} \times \mathbf{S} \mid f \in \delta_{fun} \wedge \delta_{fun} \in M_{fun} \quad (13)$$

The domain $\mathbf{D}$ is a geometric space, which may be a subset of the geographic space $\mathbf{S}$. The function $f$ associates every point from $\mathbf{D}$ with value $v \in \mathbf{V} \times \mathbf{S}$, which means that the value $v$ is also associated with the geographic space $\mathbf{S}$. The values $\mathbf{V}$ can be of any non-variable kind as long as it is computable by the function $f$. The function $f$ must be expressed as part of the transition function $\delta_{fun}$ of the functional model $M_{fun}$. The extra (relatively to geospatial objects) function $f$ for geospatial fields is required for the reason of computability and aggregates discrete values $v$ over computable representation of space using discrete coordinates.

Every field-based model requires a certain domain $\mathbf{D}$ and a the type of field values. A good example of such domain is a three-dimensional Euclidean space with computable value $v$ being the Cartesian coordinates $(x, y, z) \in \mathbf{C}$. Note that in this example both $\mathbf{D}$ and $\mathbf{C}$ are representations of three-dimensional Euclidean space, but their definition might be different as long as $\mathbf{D}$ allows function $f$ to be computable. Hence the domain $\mathbf{D}$ might cover the geographic space $\mathbf{S}$ entirely or partially, but will always be a redundant representation of $\mathbf{S}$ specific to only some model $G$. The resulting field of coordinates would contain a set smaller or equal to all coordinates of the geographic space $\mathbf{S}$.

If we consider a geospatial model $G$ with an empty $M_{fun}$ as the simplest case, one can argue that the object-based approach is superior to the field-based approach. This is true in the context of this method. All "computable" is based on a finite set of discrete steps calculated by a discrete model (see Section 1.1), which effectively prevents making truly continuous representations. We always end up 1) with many discrete values within a geospatial model, or 2) with many discrete geospatial models. Either case manifesting the principles of object-based. At best fields can be represented using a set of computable coordinates that sample the field sufficiently. Quite similarly to the coordinates of the geographic space $\mathbf{S}$, but with an important difference that all the coordinates of the field must be evaluated, which in computable modelling always depends on some transition function $\delta$.

Since model $G$ allows for an arbitrary function $\delta$ under $M_{fun}$ an arbitrary computable field-based model can be achieved. A true unification method for object-based and field-based modelling cannot a priori discount either approach. This requirement is fulfilled, but on the basis that geospatial model $G$ with empty $M_{fun}$ has object-based properties, and therefore each computable geospatial field inherits these object-based properties resulting in two layers

of structure: first an object, then a field structure. Both approaches share the resolution limit of the geographic space coordinates **S**.

## 2.4 Engineering Hierarchy of Computable Models

The design and engineering aspects must be addressed In order to facilitate practical creation and exchange of the computable geospatial model $G$, which has been presented in theory. This section introduces four engineering layers considered for the realization of Equation 7 together with its conceptual design.

Each computable model is expressed as a separate Turing machine $M$ - a theoretical computer. Manufacturing a special device for every computable model would make this method impossible to apply, but because of Equation 3 this requirement can be avoided. The design of general-purpose computers, first addressed in Burks et al. (1946), utilizes the equivalence of finite composition of computable models expressed in Equation 3. A general-purpose computer is a single electronic device that allows to load and run different computable models.

The assembly of electronic circuitry that can run computable models is the first engineering layer to consider. It is denoted as $L1$ in Figure 3. The central processing unit (CPU) chip $L1A$ runs *machine code*. The general-purpose computer uses machine code that keeps both the transitional functions $\delta$ (algorithms) and the model inputs **I** (data) of computable models together. A special kind of an in-chip model that can start loading computable models from an attached secondary memory is denoted as $L1B$. This loader $L1B$ is the first computable model that is loaded and run automatically on every computer $L1A$ start-up.

When successful, the loader $L1B$ passes the control over loading and starting of computable models to the next engineering layer depicted in Figure 3 as $L2$. Layer $L2$, which is called an operating system kernel, makes it easier to make and run programs by abstracting attached physical devices by computable models called drivers. Drivers together with the loader model denoted as $L2B$ are the main interfaces used by programs, and introduce differences between operating systems. The loader $L2B$ can manage the machine code included in programs and start them.

The third engineering layer $L3$ consists of programs that require a particular kernel for their execution. Note that "program" is yet another term for a computable model following Equation 3. Figure 3 shows three relevant types of programs; 1) geoinformation program $L3A$ that can utilize the computable geospatial model $G$ (see Equation 7); 2) compiler $L3B$ that can translate source code into an executable code for certain operating system or virtual machine (VM); and 3) VM $L3C$ that implements similar features as entire layer $L2$ for the sake of making programs independent of a particular kernel. Hence VM $L3C$ also has a component $L3D$ that can load and start programs.

The fourth layer $L4$ depicted in Figure 3 contains programs that require a particular VM $L3C$ for their execution, but can (in principle) be independent of layer $L2$. Such independence is achieved by an intermediate code that is executed by the VM $L3C$. The independent executable code is called *bytecode* (dictionary.com, 2013) and is the key engineering concept used for the unification of geospatial computable models. The independence of bytecode from operating systems is attractive because it allows computable models to be portable over a broad variety of devices and systems in a similar manner as data in traditional exchange formats. Another advantage of VM based on bytecode is that the functionality $\delta$ can be coded using many input languages. The layer $L4$ conceptually mimics the layer $L3$, but for this work is necessary only the geoinformation program $L4A$ that can utilize, on the basis of bytecode, the computable geospatial model $G$ from Equation 7.

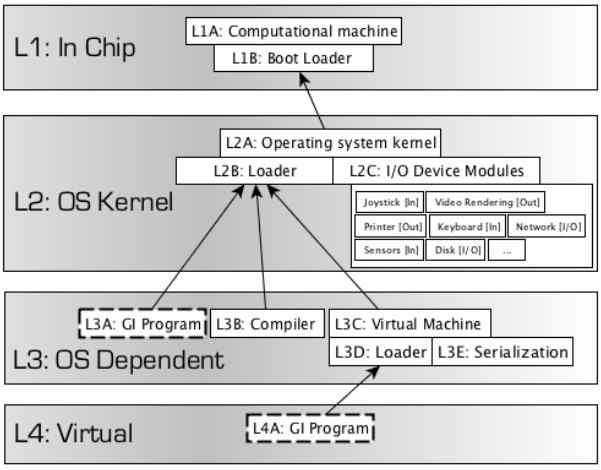


Figure 3: Engineering Hierarchy of Computational Models.

## 2.5 GMO: Geospatial Managed Object

The engineering hierarchy of general computable models suggests two nodes where the geospatial

model $G$ can be implemented. The two nodes are highlighted in Figure 3 as $L3A$ and $L4A$, and the dashed line denotes the interface to the geospatial model $G$ depicted in Figure 2. Due to the comparative properties of layers $L3$ and $L4$ the most flexible design is to implement the geospatial model $G$ under the node $L4A$ in Figure 3. The on-demand provision of components from the functional model $M_{fun}$, which is the key concept described in Section 2.2.2, then rely on the loader $L3D$. The ability of storing the geospatial model $G$ depends on the serialization[32] model depicted in Figure 3 as $L3E$.

Design employing the loader model $L3D$ and the serialization model $L3E$ is considerably more advanced then traditional designs for exchange of geographic data. The traditional exchange data formats enforce fixed predefined structure and exchange of functionality is usually impossible or very limited, in contrast the geospatial model $G$ in form of bytecode allows for variable structure and exchange of functionality. The nodes $L3D$ and $L3E$ from Figure 3 provide the engineering solution for Equation 3. The computable geospatial model $G$ implemented under the node $L4A$ using the loader $L3D$ and the serialization model $L3E$ is called *geospatial managed object* (GMO). GMO is the engineering design associated with this method.

Design similar to GMO can be implemented under the node $L3A$ from Figure 3, but with limited applicability in practice. It is also possible to implement the geospatial model $G$ under the node $L4A$ in a way that is dependent on layer $L2$, or limited in serialization of the geospatial model $G$. Such designs cannot be called geospatial managed objects. The GMO option is the most powerful engineering design in terms of sustainability, portability, applicability, and ability to reuse the geospatial model $G$.

The term "object" in GMO refers to an engineering paradigm called "object-oriented paradigm" used for programing of computable models and systems in general. In contrast the "geospatial object" is from the context of the geographic space $\mathbf{S}$. Keeping this in mind, the GMO is an engineering entity that corresponds to the computable geospatial model $G$. Hence GMO can be used for modelling of "geospatial fields" as well as of "geospatial objects" (see Section 2.2.3 and Section 2.2.2.) The term "managed" in GMO refers to the management of both data and functionality at the level of content, which is the key property of the presented method. The management of functionality is more complicated than management of only data. The bytecode and VM model $L3C$ solves this "management" of data and functionality on a common basis.

## 2.6 Model Incompleteness

The method of geospatial model $G$ fully utilizes the computability theory for modelling geospatial phenomena. GMOs provide the optimal engineering design, but certain conceptual limits cannot be avoided. They come in three levels: computability, complexity and incompleteness. For example real or irrational numbers cannot be used in computational models - only their representations. An attempt to enumerate numbers such as $\pi$ or $1/3$ would take eternity. The computable transition function $\delta$ must be always reducible to a sequence of basic arithmetic and logical operations that are physically implemented in the CPU $L1A$.

Complexity limits are given by the time and memory resources needed for computation. The precision of computable numbers used by GMOs is one aspect, because too precise representations of numbers (too much data) may not fit into the memory or take too long to evaluate. Also, many problems are inefficient when described by the transition function $\delta$ resulting in an algorithm with complex set of problem-solving operations. For example, a geospatial model deciding the shortest path through given number of cities $n$ starting and ending in the same point requires handling of $(n-1)!$ distances (Applegate et al., 2007). A computer capable of billion operations a second (THz) would compute the solution for $n = 20$ cities in about 3.8 years, and for $n = 25$ cities would already need almost twenty million years. In the logistics and telecommunication industry, this example is an essential algorithm. But because it is so hard to compute, the practical solutions must use approximations and heuristic assumptions, leaving the exact solution unknown. This example is not exceptional, there are hundreds of such complex problems (wikipedia.org, 2013). Due to their inherent complexity, many solutions to well defined models are computationally intractable.

The limits of computability and complexity might be surprisingly constraining, but the reality is even worse. In 1931 Kurt Gödel derived a mathematical proof showing that a consistent set of axioms cannot be complete. There he also showed that proof of consistency cannot be derived from the given set of axioms alone (Hirzel, 2000). This has a profound im-

[32] In terms of computer science the process of transforming computable models from the form suitable for execution to a form suitable for storage and exchange is called "serialization".

plication on computational geospatial models, which are in essence axiomatic systems using arithmetic. The Gödel's theory informally says the best we can do about any geospatial model is to assume its consistency and at the same time hope that it sufficiently addresses the problem we want to model. We have to accept that all important aspects may NOT be addressed by the model, because there is always some true statement that is relevant but that cannot be covered by the model (Devlin, 1998).

Let's consider an example of a geospatial model from Goodchild (2012) that shifts a representation of a polygon in 2d by a given vector. This is a typical object-based model, which might be incomplete for field-based features. The shift by a given vector can lead to a result when polygon intersects other polygon, which for cadaster boundaries or contourlines is an invalid result. It highlights again that field-based and object-based considerations are ubiquitous in geospatial modelling, but the main point here is the nature of incompleteness. Regardless the limits of computability and complexity there is always more specialized case for which a geospatial model is incomplete. Hence, each computable geospatial model $G$ and all its applications are primarily (and naturally) incomplete, and only then functional, field- or object-based, complex, secure, elegant and so on.

These limitations are valid for any computational model. They are included due to their importance in order to provide more consistent description of the method. The nature of incompleteness is often underestimated in geoinformation science and neglected by geospatial industry despite its implications on sustainability and efficiency of geospatial information systems. The geospatial model $G$ provides an effective and flexible way how to deal with the nature of incompleteness.

# 3 Implementation

The design of geospatial managed objects introduced in Section 13 has been implemented as a software library named *geospatial reference interface for Internet networks* (GRIFIN). This experimental library additionally implements practical features not addressed in this article including mechanisms for storage and retrieval of GMOs, exchange of GMOs over network, automatic 2d interpretation of 3d geometries, support for visualization, and API for client applications to use GMOs. The experiment is maintained at `http://grifinor.net`, and the author's intention is to provide the library to everyone under the GNU General Public License.

In order for the GMO implementation to be applicable and robust few already existing technologies have been applied. The most crucial in this regard is the selection of the VM technology. Note that the utilization of VM here is not a mere software engineering convenience for implementation or porting to different operating systems. The nature of VM, as described in Section 13, is an inherent part of the GMO method. The role of the VM's bytecode relatively to GMOs is similar to the importance of HTML relatively to Web pages. The consequence of a future change to a different VM would have an analogy in changing HTML to, let's say, PDF format - making all the previous content obsolete and nonfunctional. Hence requirements on the VM technology are relatively strict and include: non-proprietary solution, strong focus on backward compatibility, production quality with commercial leadership, and widely established availability. Given these priorities the HotSpot VM, which is the original VM used within the Java ecosystem, stands out as nearly unchallenged choice, despite its currently marginal availability on mobile platforms.

The GMO concept has been coded in form of an abstract class, which implements the referencing mechanism $M_{ref}$ and an abstract method $manage$ representing $M_{fun}$. This guarantees access to the referencing mechanism and custom functionality for all implementing subclasses and their GMO instances. The geospatial index $M_{sdx}$ introduced in Section 13 has been implemented as a hash function mapping from the geospatial coordinates $\mathbf{S}$ to an array of 32 bytes. Examples at `http://grifinor.net/examples` also provide references to the source code. In order to facilitate prototyping and re-use of GMO models most of the examples utilize Scala language and GRIFIN Shell (GShell), which allows for an interactive use. GShell can be used to create, manage and consume the geospatial content on GRIFIN platform. It has all management, server, and remote access features available from a uniform environment and provides a way to exchange and execute code on GRIFIN's distributed network. This follows the original vision of a space for collaboration on model development, and not just a one-way publishing medium for static, predefined, and hard-to-change types of geospatial information.

Since 2006 several projects used GRIFIN, and many different GMO models were implemented as both object- and field-based representations of the real world environment. Figure 4 depicts results

from InfraWorld project (Kolar, 2010) in three different software clients.

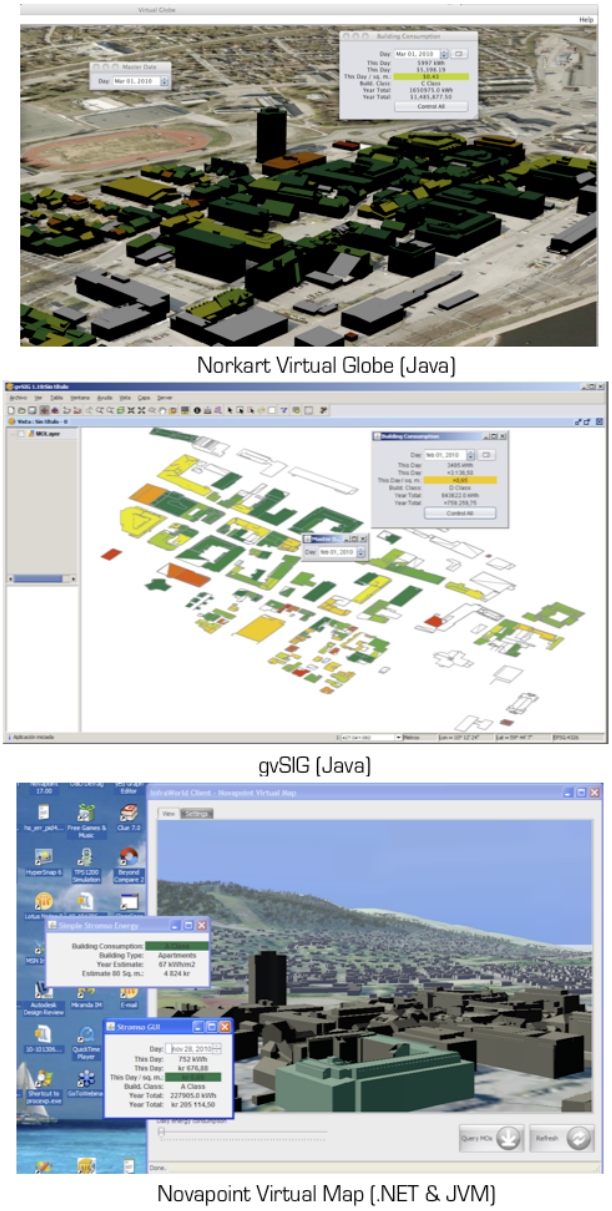


Figure 4: Three different software clients consuming identical GMOs from a server.

The city model and a daily energy consumption per building spanning a one-year-period were modelled as GMOs. While the model is relatively complex the software clients only implement the API for handling GMOs, which accounts circa twenty methods. The city model itself has several times more methods. The model brings all its functionality to the different clients, its specification and definition undertook big changes while the API for GMO could be implemented independently in parallel, and once the GMO API is done the software client is ready for all future models using the GMO method. This summarized three practical properties, which might be hard to address using non GMO solutions. Figure 5 depicts two examples of field-based representations. The right-hand side depicts a GMO model for "Nomenclature of Units for Territorial Statistics" including its hierarchical subdivision of administrative units. The spherical grids in Figure 5 show a GMO implementation of the Global Indexing Grid described in Kolar (2009), which is suitable for a spherical subdivision at arbitrary resolution and is a convenient basis for more complex models using the nearest neighbor queries. Energy City Frederikshavn (Wen et al., 2010) is other project based entirely on GMO method, featuring an object-based city model and a field-based terrain representation.

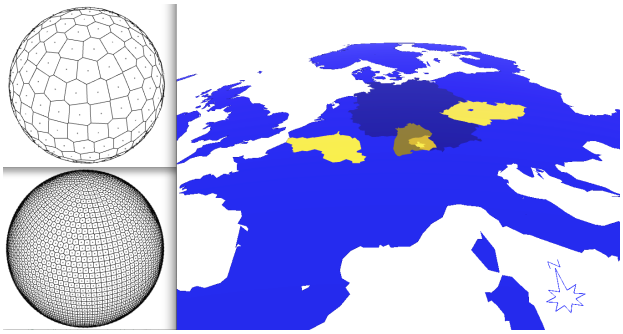

Figure 5: Field-based geospatial representations modelled using the GMO method.

## 4 Conclusion

A uniform theoretical and engineering solution supporting both field- and object-based computable geospatial models was presented and implemented. The geospatial model $G$ and its GMO engineering design supports any computable representation of object-based or field-based geographic features, as well as complex geospatial models combining both approaches. The unification of the method leans on the definition of geographic space $\mathbf{S}$, which includes time, and which can be adopted at different scales and subdimensions. Scales are represented as unit-less dimensions of the space $\mathbf{S}$. Computable geospatial field inherits certain object-based properties enforced by the computability theory resulting in two layers of structure: first an object, then a field associated with the function $\delta$. The key concept of the method is keeping the structured data together with the functionality $\delta$ as one content unit.

This is not new in general, but the method fully unfolds this potential of computability theory in GISc and brings it to geospatial applications. The byte-code, which is the engineering solution for keeping data and functionality together, is inherently associated with a particular VM. That is a unique requirement compared to solutions based on more traditional data exchange. GMO, as the optimal engineering design for the geospatial model $G$, encapsulates models on the level $L4$. This design is more flexible in addition and changes of models, more portable, and easier to deploy than solutions designed on the level $L3$ as well as those on $L4$ without the GMO encapsulation. GMOs make it possible to handle data and functionality at all ends of a distributed network, also after the object is instantiated, exchanged and consumed on the client side. This could facilitate the exchange of research developments of geospatial models. It was stressed that computational models are naturally incomplete, which has profound implications for the geospatial industry in terms of efficient sustainability of geospatial applications and for synergy of various information systems into larger infrastructures. Although the incompleteness is an unsolvable issue, GMOs provide an effective and flexible way to mitigate it. It also avoids many issues met by standardization efforts ongoing in the geospatial industry. The GMO design was implemented in the GRIFIN framework and is associated with the HotSpot VM. The VM requirement is the key engineering disadvantage of the method. Practice shows, however, that for the useful models the VM overhead can be minimized to a negligible level by applying adequate representations.

# References


Applegate, D. L., Bixby, R. E., Chvatal, V. & Cook, W. J. (2007), *The traveling salesman problem: a computational study*, Princeton University Press.

Boucher, C. (2001), Terrestrial coordinate systems and frames, *in* 'Encyclopedia of Astronomy and Astrophysics', Nature Publishing Group, pp. 3289–3292.

Burkholder, E. F. (2000), The global spatial data model, *in* 'International Conference on Discrete Global Grids', Santa Barbara, California, USA.

Burks, A. W., Goldstine, H. H. & von Neumann, J. (1946), 'Preliminary discussion of the logical design of an electronic computing instrument', http://grifinor.net/cite/2013/FOSS4G2013/4.

Devlin, K. J. (1998), *The Language of Mathematics: Making the Invisible Visible*, W.H. Freeman & Company.

Dictionary.com (2013), 'Define byte-code'. URL: http://grifinor.net/cite/2013/FOSS4G2013/2

EPSG (2013), 'European Petroleum Survey Group: Geodesy Parameters Archived Versions'. URL: http://www.epsg.org/archive.html

Goodchild, M. F. (2012), 'Field-based spatial modeling', http://grifinor.net/cite/2013/FOSS4G2013/3.

Hirzel, M. (2000), 'On formally undecidable propositions of Principia Mathematica and related systems I', http://grifinor.net/cite/2013/FOSS4G2013/7.

Kjems, E. & Kolar, J. (2005), From mapping to virtual geography, *in* 'CUPUM', London, U.K.

Kolar, J. (2009), GIG: a projection-free global grid system suitable for indexing, *in* 'Proceedings of the 6th Symposium of the International Society for Digital Earth', Beijing, China.

Kolar, J. (2010), 'InfraWorld Project, The VERDIKT-Conference 2010', http://blog.grifinor.net/post/3360221925.

Kopeikin, S. M. (2007), Relativistic reference frames for astrometry and navigation in the solar system, *in* 'AIP Conference Proceedings', p. 268.

Pogge (2013), 'GPS and Relativity by Richard W. Pogge'. URL: http://grifinor.net/cite/2013/FOSS4G2013/1

Sipser, M. (1996), *Introduction to the Theory of Computation*, 1 edn, PWS Pub. Co., pp. 125–130.

Weisstein, E. W. (2013), '"Octant." From MathWorld–A Wolfram Web Resource'. URL: http://gri[FB01?]nor.net/cite/2013/FOSS4G2013/6

Wen, W., Kjems, E., Bodum, L. & Kolar, J. (2010), Dynamic features in a 3d city model as an energy system, *in* 'ISPRS Conference: International Conference on 3D Geoinformation', Berlin, Germany.

Wikipedia (2013), 'List of NP-complete problems'. URL: http://grifinor.net/cite/2013/FOSS4G2013/5