Conceptual programming in GIS development

V Ya Tsvetkov and A V Buravtsev

Russian Technological University (RTU MIREA), 78, Vernadsky Prospekt, Moscow, 119454, Russia

E-mail: cvj2@mail.ru

Abstract. The article explores the application of conceptual programming methods in the development of software for geographic information systems. The connection between conceptual and functional programming through the apparatus of functional grammars is shown. It is proved that a natural language in conceptual programming can be expressed through a domain-specific language, which is an integral part of language-oriented programming. The article proposes the implementation of conceptual programming ideas by combining functional and language-oriented programming styles. It is proved that this approach is more acceptable in comparison with the imperative approach for conceptualizing the subject area. Approach implementation example for a geographic information system for maintaining an address register is given. The example uses the multi-paradigm functional programming language F#. The article proposes a new approach to constructing algorithms called constructive algorithms. It is based on the principles of conceptual design and conceptual programming. This approach improves the quality and speed of geographic information systems development through the use of ontological algorithms.

1. Introduction

Geographic information systems (GIS) are one of the types of integrated information systems [1]. Building an integrated system starts with design, which includes software design. The quality of design determines the quality of technology [2] of GIS work. This justifies the substantiation of design and modeling methods for GIS software. Modern GIS is considered as complex systems [3]. This motivates the use of system analysis [4] and conceptual [5, 6] design when creating a GIS as a complex information system [7]. The design of a geographic information system begins with the development of its concepts [8]. The concept denotes a generalized way of understanding or interpreting any phenomena. A conceptual model is a general model represented by a variety of concepts and relationships between them, which determine the semantic structure of the object under consideration. The most important conceptual models of the domain are called concepts [9]. The immediate picture of the world [10] includes knowledge and also conceptual knowledge [9] as a special kind of knowledge.

From the point of view of the information system being developed, a concept is a generalized description that simultaneously serves as the basis for constructing: the structure of the system, the technology of its operation, data types, software for the system. Therefore, one of the popular design methods is the method of constructing concept maps in terms of the subject area. An important stage in the implementation of a geographic information system is the process of converting general concepts into software concepts. This task is solved through the use of conceptual programming. Consequently, conceptual programming, on the one hand, is a logical consequence of conceptual design. On the other
hand, conceptual programming is a method of increasing the reliability and quality of software for any complex information system. An additional tool for the transition from a conceptual approach to the creation of real programs is the method of constructive programming and constructive algorithmization.

2. Research methodology
The work used methods of conceptual analysis and design, as well as methods of functional, conceptual and language-oriented programming. The examples use the functional programming language F # as the programming language. To run the code, the user must have the latest Microsoft Visual Studio or Visual Code development environment on the user's computer.

3. Research results

3.1. Features of conceptual programming
Conceptual programming is a direction in computer science "based on the description of concepts to express the meaning and form of a specific problem, according to the description of which results can be automatically obtained or programs are synthesized for execution" [11].

In other words, conceptual programming is one way of solving problems on a computer, which consists in applying concepts sufficient to express the meaning of a problem and write it in terms of these concepts. Based on this description of the problem, the compiler builds an algorithm for its solution and performs the necessary calculations.

Conceptual programming is characterized by: programming in terms of a subject area; use of a computer at the stage of delivering tasks; synthesis of the algorithm and program for solving the problem; the use of semantic memory to store knowledge about the tasks being solved.

The main difficulty of conceptual programming is the difficulty of formulating a problem for setting it in front of a computer, in a form that is understandable for both a person (the task manager) and a computer, since the terms of the subject area need to be described in sufficient detail and completely, taking into account the fact that during the initial formulation of the problem you can see the exact algorithm for its solution.

Conceptual programming technology is designed to synthesize program code for solving a wide range of problems from various subject areas by describing them in natural language using concepts and generalized concepts. The natural language used for the transmission and perception of information is fundamentally different from programming languages in its capacity, ambiguity of meanings, associativity, rich vocabulary, ambiguity of interpretation, cognition, inconsistency, etc.

Information systems use artificial languages that are limited subsets of natural language. Artificial languages are subject-oriented. They have rigidly fixed sets of forms for constructing sentences or information structures from natural language forms. They are less controversial and tend to be logically consistent. When formulating these constructions, the tools of linguistics and the theory of formal grammars are used.

Methods of language constructions creation in geoinformatics and programming require:

1) an exact indication of the subject area for the problem being solved;
2) determination of the set and class of problems to be solved.

Last-mentioned are called computational or computational and logical problems. Their description in a restricted natural language in general is as follows:

$$\text{Knowing } M, \text{ calculate } (y_1, \ldots, y_n) \text{ from } (x_1, \ldots, x_m)$$  \hspace{1cm} (1)

In expression (1), $M$ defines the subject area (for example, land registry, geometry, geoinformatics, etc.). A tuple of the form $(x_1, \ldots, x_m)$ contains identifiers of variables with known values, and a tuple $(y_1, \ldots, y_n)$ contains identifiers of variables whose values are to be determined.
One of the significant limitations of conceptual programming technology is the assumption that a computer has domain priori model to manipulate with it. In the context of conceptual programming technology, semantic networks called computational models are used to represent a domain model. When implementing this method, we get two important results:

1) the program exactly matches the description of the task;
2) instead of debugging the program, the task description is "debugged".

Conceptualization is the process of moving from a description of a domain in a natural language to an exact specification of this description in a formal language oriented towards a computer representation.

Computational models are developed as a mathematical apparatus for conceptualization within the framework of conceptual programming. They are varieties of semantic networks. The semantic network $S$ is generally defined as follows:

$$S = (0, R) = (\{o_i | i = 1,2,\ldots,k\}, \{r_j | j = 1,2,\ldots,l\})$$

where $O$ is a set of objects of the subject area $|O| = k$; $R$ is a set of relations between objects of the subject area $|R| = l$. A computational model for a given subject area is defined as a tuple:

$$\left(\{p_i\}, \{f_j\}, \{u_k\} \right)$$

where $p_i$ is the name of the concept; $f_j$ - functional relationship between concepts; $u_k$ - control structure. The functional relation $f_j$ is given by three elements:

$$f_j = (X_j, F_j, Y_j)$$

where $X_j = (x_{j1}, \ldots, x_{jm})$ is a set of input variables for $f_j$; $F_j$ is a reference to a procedure that computes $Y_j = F_j(X_j)$; $Y_j = (y_{j1}, \ldots, y_{jm})$ is a set of output variables for $f_j$. The $u_k$ control structures implement maps of $X_j$ and $Y_j$ to the set of permitted data types, and they also allow you to assign known and computed values to variables.

Work [12] describes the use of functional grammars in conceptual programming. It shows the advantage and versatility of conceptual programming. In [12], it was substantiated that functional grammars more adequately describe the knowledge base and the model of inference of the solution. It shows that the result of conceptual programming in the form of a superposition of functions makes it possible to find a solution to the problem in two versions: directly and in the form of a synthesized computer program in any chosen programming language.

Let us give a formal definition of a superposition of functions. Superposition is the technique of obtaining new functions by substituting the values of some functions instead of the values of the arguments of other functions. Let a partial function be given:

$$g(x_1, x_2, \ldots, x_n)$$

and partial functions:

$$f_i(x_1, x_2, \ldots, x_n) = f_i(x_1, x_2, \ldots, x_n)$$

the superposition of the functions $g$ and $f_1, \ldots, f_m$ is a partial function:

$$h(x_1, x_2, \ldots, x_n) = g(f_1(x_1, x_2, \ldots, x_n), \ldots, f_m(x_1, x_2, \ldots, x_n)),$$

which is specified on the set $(x_1, x_2, \ldots, x_n)$ by the specified formula, if all values $y_i = f_i(x_1, x_2, \ldots, x_n), \ldots, y_m = f_m(x_1, x_2, \ldots, x_n)$ and value $g(y_1, y_2, \ldots, y_m)$.

In a functional sense, any program can be viewed as a function or a superposition of functions, which receives output data from input data. The association of conceptual programming with functional grammars provides a basis for expressing conceptual programming constructs using functional programming.
Due to the fact that at present conceptual programming practically does not develop in the applied aspect (at the level of programming languages, interpreters and compile systems), the problem arises of implementing the problems to be solved in the style of conceptual programming with the help of other styles that are as close to it as possible.

Conceptual programming technology is designed to synthesize programs and solve problems by describing them in a language that is, in fact, a domain-specific language.

One such DSL is “Domain Specific Language”, specialized for a specific application. "The construction of such a language and / or its data structure reflect the specifics of the tasks being solved" [13]. Domain-specific programming language is a key concept in language-oriented programming. Language Oriented Programming is a style of programming that creates specialized code. This code looks like a piece of a domain-specific language, but operates in a general-purpose programming language.

The connection of conceptual programming with the apparatus of functional grammars makes it possible to implement the style of conceptual programming using language-oriented programming using functional programming languages (figure 1):

![Figure 1. The relationship between conceptual and functional programming.](image)

A positive feature of functional programming is that it allows you to describe a program in a "declarative" form. In this form, a rigid sequence of performing many operations is not explicitly specified, but is generated automatically in the process of calculating functions. This property and the absence of states makes it possible to apply complex methods of automatic optimization to functional programs.

Practical examples in the article are given using the multi-paradigm programming language F#. It focuses on the functional style, allowing for imperative and object-oriented styles as needed. The multi-paradigm nature of the language allows the programmer to avoid the limitations imposed by each of the programming styles and to be more flexible in solving problems. The F# language uses the following functional tools:

- currying (defining a function of several arguments as a higher-order function),
- tuples, function types, name binding, computation expressions,
- meta-representation of the abstract syntax tree of the program, etc. [14].

The language has many areas of application, of which one should highlight those that approach the field of geographic information systems in the context of conceptual programming: modeling, simulation, processing large-scale data (for example, large geodata), language-oriented programming, extensible code, parallel and multi-threaded programming.

In general, we will show the relationship between the syntax of programs in conceptual programming and its correspondence to a program in a functional style. An example of a complete metasyntactic program formula in conceptual programming looks like this:

```xml
<program> ::= 

  program <ID>;
```

```xml
```
The metasyntactic formula of a program in modern functional programming languages looks a lot like formula (8), except for the explicit designation of the `program` and `end` blocks. In F#, blocks `let` and `actions`, in their simplest form, can be expressed through the `let` binding statement. The binding operator binds an identifier to a value or function:

```fsharp
let (description) (relationship) (act)
```

### 3.2. An example of conceptual programming methods implementation for geographic information systems

Let us consider conceptual programming in more detail in the context of the development of geographic information systems. As a geographic information system, let us take a specialized geo-information system - the Federal Information Address System (FIAS), whose main task is to maintain the state address register (SAR) [15].

An address is location conditional codification of an object in space, used for various purposes: postal communication, statistics, forecasting, tax charges for real estate, etc. The mathematical model of the address can be represented in the form of an oriented (directed) tree - an acyclic graph [16]. “An acyclic digraph (directed graph that does not contain cycles) is a graph in which only one vertex has a zero-entry degree (no arcs lead to it), and all other vertices have an entry degree 1 (exactly one arc leads to them). The vertex with a zero degree of entry is called the root of the tree, the vertices with a zero degree of exit (from which no arc emanates) are called end vertices or leaves” [17]. Let's take an example of representing the structure of an abstract addressable object from the book "Conceptual Programming" [11] (figure 2):

![Figure 2. The structure of an abstract addressable object.](image)

Let's implement the prototype of the hierarchical structure of the address object using the F# language. First, let's describe the "Coordinates" data type that is part of the address object type:

```fsharp
// unit of measure "degree" (latitude or longitude)
[<Measure>] type degree

// coordinates - width and longitude
type Coordinate = {Latitude: float <degree>; Longitude: float <degree>}
```

The example uses one of the unique features of the F# type system — the ability to provide context for numeric literals through units. A special language construct allows you to associate a numeric type
with a unit of measure. This allows the compile system to check the types of numeric literals that make sense in a particular context, eliminating run-time errors. This unique feature of the language can be used in the implementation of geographic information systems. Here is an example of using units of measurement:

```fsharp
// connecting the F # core library, which defines the SI (International System of Units) types of units:
open Microsoft.FSharp.Data.UnitSystems.SI.UnitNames

// defining the unit of measure "mile":
[<Measure>]
type mile =
// conversion factor for miles to meters:
static member asMeter = 1609.34 <meter / mile>
// distance expressed using imperial system:
let d = 50. <mile>
// same distance expressed in metric system:
let d2 = d * mile.asMeter

// output the result to the console:
printfn "% A =% A" d d2 // 50.0 = 80467.0

// compilation error: mismatch of measurement systems:
let error = d + d2

Let us return to the description of the concept of an address object in terms of a subject area in the F # language:

```fsharp
// generic address type
type AddressType =
| Node of NodeAddressType
| Leaf of LeafAddressType

and NodeAddressType = {Name: string; Coordinates: Coordinate; SubItems: AddressType list} // address-forming objects

and LeafAddressType =
| House of HouseType
| ApartmentType of ApartmentType

and HouseType = {Name: string; Coordinates: Coordinate; SubItems: ApartmentType list} // buildings and structures

and ApartmentType = {Name: string} // premises

// description of the structure of the "tree" type
type Tree <'LeafData,' INodeData> =
| LeafNode of 'LeafData
| InternalNode of 'INodeData * Tree <'LeafData,'INodeData> seq

// definition of the "address" type through the structure of the "tree" type
type AddressTree = Tree <LeafAddressType, NodeAddressType>
```

The presented code describes the concepts of the domain. According to the principles of conceptual programming, in terms of these concepts, the task and the way to solve it will be described. In the context of the address registry, tasks can be as follows: inserting (expanding) a new type into an existing one, searching for an object in a tree according to a given criterion (recursive iteration over a hierarchical structure), calculating the distance in space between geographic coordinates, modeling processes in a GIS, etc.

3.3. Constructive algorithmization

Using the methods of conceptual design and conceptual programming, it is possible to propose a new approach to the construction of algorithms, called constructive algorithms. In conceptual programming, there are two principles associated with the transfer and preservation of knowledge in a program.
The first principle: “The system must remember and, if necessary, use what has been told to it at least once” [11]. The second principle for subroutines: "Subroutines are indivisible units of knowledge and can be programmed in any programming language supported by the operating environment" [11]. Thus, we can talk about the ontology of algorithms using the principles of conceptual design and programming.

Constructive algorithmization is a method of constructing an algorithm by initially constructing a conceptual model called an information structure [18]. The informational construction of the algorithm is primarily focused on tasks and subject area. It includes a number of tasks for which the algorithm can be applied.

Usually, in practice, an algorithm is developed to solve a specific problem according to the rule “one task - one algorithm”. Information construction includes theoretical and algorithmic foundations for solving a certain group of problems: "one construction - many related problems and many similar algorithms." Thus, the design problem is replaced by the selection problem. Schematically, constructive algorithmization consists of the following blocks (figure 3):

![Figure 3](image.png)

**Figure 3.** Scheme of constructive algorithms used to increase the flexibility and adaptability of algorithms.

Thus, constructive algorithmization is a new approach to the construction of algorithms, replacing technical algorithms with ontological ones. Constructive algorithms increase the flexibility and adaptability of algorithms when new requirements or new conditions appear. This approach is analogous to the use of patterns in design, but in the transfer of this approach to the area of knowledge and conceptual models. Thus, constructive algorithmization improves the quality of software products.

4. **Conclusion**

A significant drawback of conceptual programming methods, which hinders its development and use, is the use of a limited natural language in it, focused on narrow problem areas and having rigidly fixed sets of forms for constructing sentences. To remove these restrictions, further development of artificial intelligence methods, linguistics and the theory of formal grammars is required. Conceptual programming and constructive algorithmization makes it possible to save and reuse knowledge, which increases the efficiency of using programs, increases the reliability and speed of developing programs and information systems. Conceptual programming has a number of challenges to be solved in the future using artificial intelligence techniques.

**References**

[1] Bugaevsky L M and Tsvetkov V Ya 2000 *Geographic Information Systems* (Moscow, Russia: Zlatoust) 224

[2] Ilyin V 2019 *Management of the Quality of Projects. Practical experience* (Litres)

[3] Kuj S A 2015 GIS as a complex system *Slavic Forum* 1(7) 108-16
[4] Ipatova E R 2008 *Methodologies and Technologies of System Design of Information Systems* (Flint)

[5] Teslinov A 2017 *Conceptual Design of Complex Solutions* (Litres)

[6] Leonov A V and Semerikov N V 2011 General methodology of conceptual design of high-tech dual-use products *Double technologies* 4 64-71

[7] Monakhov S V, Savinykh V P and Tsvetkov V Ya 2005 *Methodology for the Analysis and Design of Complex Information Systems* (Moscow, Russia: Russia: Education) 264

[8] Bliskavitsky A A 2013 Conceptual modeling and design of GIS *Informatization and communication* 2 43-5

[9] Mayorov A A 2014 Concept and infosphere *Russian technological journal* 4(5) 1-17

[10] Tsvetkov V Ya 2014 Worldview Model as the Result of Education *World Applied Sciences Journal* 31(2) 211-5

[11] Tyugu E Kh 1984 *Conceptual Programming* (Moscow, Russia: Nauka) 255

[12] Kravchenko V A, Shirapov D Sh and Olzoeva S I 2016 The use of functional grammars in conceptual programming *Bulletin of the Buryat State University* (Ulan-Ude: Publishing house of the Buryat State University) 4 3-12

[13] Fridland A Ya and Chanamirova L S 2003 *Computer Science and Computer Technology: Basic Terms* (explanatory dictionary) (Astrel) 01.01. 2003 p 270 ISBN 9785170145461

[14] Eason K 2018 *Stylish F #: Crafting Elegant Functional Code for .NET and .NET Core* (New York: Apress) p 408

[15] 2019 *Federal Information Address System* (date of access: 28 May 2019) Retrieved from: http://fias.nalog.ru/

[16] Tsvetkov V Ya and Buravtsev A V 2017 Directed graphs in complex organizational and technical fiscal systems *Educational resources and technologies* 3(20) 33-40

[17] Ore O 1980 *Graph Theory*, 2nd ed (Moscow, Russia: Nauka) 336

[18] Buravtsev A V 2018 Constructive Algorithmization *ITSEM: Information Technologies in Science, Education and Management* 1 85-92