Object-Oriented Representation of Mixed Models Knowledge in the Design of Electronic Devices in CAD Electra

G. A. Dolin
Moscow Technical University of Communications and Informatics
Moscow, Russia
Dolin1974@gmail.com

***Abstract*— At present days a promising direction is the development of information management systems design, allowing to describe the subject area in the form of a set objects. The purpose of this paper is to consider representation models knowledge in expert systems and highlighting the benefits of using object-based -approximate representation of knowledge about the design area radio device. This paper outlines the principles of construction object-oriented knowledge base management systems that would be compatible with existing programming languages with respect to structures representations of managed knowledge and would provide opportunities to integrate them into expert systems operating on the basis of the concept of open systems. It is proved that the most optimal is the separation of knowledge and meta-knowledge complex mixed types resulting from inheritance. At the same time, internal representation of complex objects formed as a result of inheritance, looks like inheritance chains. When working with complex this system automatically searches special links, allowing you to perform operations according to the SQL standard.**

Keywords*— CAD; electronic; synthesis; object oriented; device design; expert system

I. INTRODUCTION

An essential feature of modern software systems for designing radio technical devices (RTD) is their increasing complexity [4, 6]. This complexity is generated by the properties of real devices, which are reflected in the implemented models.

As a rule, RTD have, to one degree or another, a hierarchical structure. In RTD, you can find many different types of hierarchies. In a radio receiver, for example, it is possible to distinguish systems of separation (filtering), demodulation, signal amplification, etc. Such a partition gives a structural hierarchy of the type "this is part of that." On the other hand, a device such as an audio frequency amplifier is just a type of amplifier endowed with properties that distinguish it from other amplifiers. Based on this, it is possible to distinguish a typical and structural hierarchy, called, respectively, the hierarchy of classes and the hierarchy of objects. The structures of classes and objects are not independent: each object represents a specific class.

Thus, in designing a complex RTD can be composed of subsystems, each of which can be synthesized, analyzed and optimized independently of the others. As a criterion for decomposition, you can choose whether the elements of the device belong to different abstractions of the subject area. In this interpretation, the RTD is represented by many autonomous operating components (functional units and cascades) that interact with each other. Each component of the RTD has its own behavior that simulates the behavior of a real device, and is an instance of one of the classes, which, as indicated, form a hierarchy. This style is called object-oriented decomposition.

Based on the foregoing, the authors believe that at present a very promising direction is the development of expert information systems (ES), which allow to describe the subject area (in particular, the field of designing a RTD) in the form of a set of objects [5]. Therefore, the aim of this work is to consider models of knowledge representations in ES and highlight the benefits of using an object-oriented representation of knowledge about the design field of a RTD.

The following sections contain definitions and examples of various aspects of knowledge representation. Section 2 discusses in detail the issues related to the hierarchy of objects in object-oriented knowledge bases (OOBZ). Section 3 describes how to implement an object-oriented representation of knowledge. Section 4 discusses the features of using mixed knowledge representation models. In conclusion, an overview is given of concepts that seem essential for further analysis.

II. OBJECT ORIENTED APPROACH

The most developed way of representing knowledge in ES is an object-oriented paradigm. This approach is a development of the frame representation. It is based on the concepts of object and class [1].

In the subject area of interest to the developer, specific objects, as well as abstract or real entities, i.e. a combination of properties and characteristic actions that are essential for modeling interesting aspects of the study area (hereinafter, the term “object” is used as a shortening of the combination of “type of object.” Where it can cause confusion, it is specified, we are talking about instances or types objects). For example, the objects of radio engineering can be: a radio system; radio channel; radio device; functional or structural blocks and cascades of radio devices; electronic components, etc. An object has individuality and behavior, has attributes whose

978-1-7281-4772-7/20/$31.00 ©2020 IEEE

values determine its state. So, a specific radio receiver, when the reception conditions are worsened, may be in a state where the used decoder cannot correctly decode the signal, and its behavior in this case consists in changing the decoding principle. In addition, each object is a representative of a certain class of objects of the same type.

A class defines common properties for all its objects. Such properties include the composition and structure of data describing the attributes of a class and corresponding objects, and the totality of methods - procedures that determine the interaction of objects of this class with the external environment. For example, the description of the “power amplifier” [13 - 17] or “transmitter” [18 - 20] classes may include such attributes that determine the state of objects that are individual for each object of this class — a specific amplifier (for example, an ultrasonic amplifier), as well as general methods to determine them, etc.

Objects and classes have characteristic properties that are actively used in the object-oriented approach and largely determine its advantages. These properties include: abstraction, encapsulation, inheritance, messaging, and polymorphism.

An object-oriented approach considers program resources: data, modules and systems - as objects. Each object contains some structure (or type of knowledge), framed by a set of procedures that can manipulate this data. Using this methodology, the developer can create his own abstract type and map the problem area to these created abstractions, instead of the traditional mapping of the problem area into predefined control structures and knowledge structures of the implementation language. Thanks to the object-oriented methodology, large programs become more understandable, since all interconnected executable fragments and a description of knowledge are in one place. In addition, the modularity of programs makes them easier to navigate and develop. Thus, the approach implemented on the basis of an object-oriented programming methodology is more natural than processing-oriented (process-oriented) methodologies because of the ability to create various types of abstractions of knowledge types that most accurately describe the field during the design process designing informal radio circuits and devices. On this path, the developer can concentrate on the system design, without worrying about the details of the information objects used in the system. The structure of each model, its mathematical description becomes simpler, and in order to complicate the models, the inheritance of properties from earlier created ones is used (for example, when moving from a simplified model of the Bi-Polar Transistor to the full Humel-Pun model, it’s enough to introduce additional functional relations, not developing again the components of the program operating with components with a simplified model). This allows you to significantly reduce the machine time required for modeling the RTD, and requires less computer memory, because the complexity of the models and their implementations affects the volume of the basic or structural schemes of the RTD, which can actually be designed by the program, through the use of models of varying complexity, requiring different analysis times.

So, the object-oriented approach consists in representing the system as a combination of classes and objects of the subject environment. Moreover, the hierarchical nature of a complex system is reflected in the form of a class hierarchy, and its functioning is considered as the interaction of objects with which, for example, ES production rules are associated. Moreover, the association of ES production rules with the class hierarchy is achieved through the use of general rules, where the reference to the class to which this rule applies is usually used as a prefix.

To illustrate, we give the following example. The "resistance" of the resistor can be fixed at room temperature. The current state of the characteristics of the device is determined by the ambient temperature, which can increase or decrease when it changes. Such an object can be described, for example, as follows:

Example 1

```
OBJECT RESISTOR:
   Resistance whole
OPERATION Zoom (How many integer):
{
    Resistance = Resistance * (1 + exp ((Ut - How much) / Ut));
}
```

The program may exist several different instances of objects of this abstract type in accordance with the resistors available in the simulated device. However, an important feature of the object-oriented approach is the uniqueness of each of the created objects of this type, which reflects the binding of the program object to a real unique object and the possibility of delimiting it from other objects of this program. Often the restriction of visibility of the internal structure of an object to other objects due to encapsulation is applied.

The hierarchy organizes the system of abstractions, allowing you to create complex types of objects, in the form of inheritance, when one class uses the structural and / or functional part of one or more other classes (respectively, simple or multiple inheritance). In other words, inheritance is a hierarchy of abstractions in which subclasses inherit a structure from one or more superclasses. As a rule, in the hereditary hierarchy, the common part of the structure and behavior is concentrated in the general superclass. At the same time, subclasses reflect more specialized abstractions. Here is an example of creating new classes using inheritance:

Example 2

```
OBJECT electronic_component:
    Type characters 20;
OBJECT resistor Inherits electronic_component:
    Resistance whole;
OBJECT capacity INHERITED electronic_component:
    The capacity is whole.
```

Each of the objects “resistor” and “capacitance” will have a common attribute “type”, as well as individual attributes that they should have according to the specifics of the simulated subject area.

Subclasses can be supplemented and modified with respect to superclasses. Usually, it is possible to override the implementation of methods inherited from superclasses. Keeping the interface of the method described in the superclass, the body of the subclass method contains a description of its own actions. This important property allows you to have only the name and parameters of the method call in the generalizing class, leaving implementation questions to each successor class that has a common property, but has individual characteristics. A typical example is the method for determining the required electronic component, depending on whether it is a "resistor" or "capacitance" (see example 2). The implementation of multiple inheritance seems to be more complicated due to occurring coincidences of element names (the so-called collisions) inherited from different superclasses, or, on the contrary, designations by the opposite attributes of superclasses of the same entities of the simulated domain.

In addition to inheritance, another type of hierarchy is often used - aggregation, while an element of one class can also have an abstract type, i.e. be in turn an object of a class

III. STRUCTURAL HIERARCHY

The types of structural hierarchy in object-oriented de-composition — inheritance and aggregation — were examined above. The principles of simple inheritance in the objects of the knowledge base will be described below. The principles for implementing multiple inheritance and aggregation will be mentioned without justification. So far, the functional properties of objects represented by their methods are not affected in order to dwell on the structural hierarchy. Also, for simplicity, in all examples, only variables are given as properties of objects, although, as follows from the above knowledge model, all results are generalized to properties that reflect interobjective relationships.

Consider the simplest inheritance structure:

Example 3

fields

OBJECT electronic_component:

X type characters 20;

OBJECT resistor Inherits electronic_component:

Y resistance whole;

A similar description of the type structure in programming languages (PL) will lead to the fact that in the RAM will be allocated an area for storing field X of the object electronic_component and independently an area for storing fields X and Y of the object is a resistor. In other words, PLs form independent segments of knowledge for each complex type resulting from inheritance.

One of the important principles of KB technology is the redundancy (as far as possible) of the stored information. In addition to saving resources, redundancy contributes to greater reliability in data manipulation. Indeed, in the case of duplication of knowledge, when making changes to them, it is necessary to have a mechanism for monitoring the maintenance of integrity (avoiding discrepancies) and ultimately produce at least twice as much work (modifying knowledge in two places). Therefore, the optimal is the separation of knowledge and metacognition of complex types resulting from inheritance. We use the same simple example to illustrate. The part of objects of the resistor type that is inherited from objects of the superclass electronic_component (fields X) is preserved in the same way as if they were instances of objects of the type electronic_component. The own part of objects of the resistor type (fraction Y) is stored separately as objects containing only the Y field would be stored. It can even be said that the resistor objects inherit the storage mechanism in that part, which consists of the X field - the electronic_component borrowed from the object.

The described situation reflects the following view of the structure of objects electronic_component and resistor. The electronic_component objects have the X property, and the resistor objects have the X and Y properties. All resistor objects can be "projected" onto the electronic_component objects (orthogonal to the space of the object variables the electronic_component - in this case, the variable X is projected onto the hyperplane), which is illustrated in Example 1.

As for knowledge, for modeling which a similar structure is used, then with the described approach, objects of type B continue to exist as objects of type electronic_component. This looks more clearly in Example 2: the electronic_component, being a resistor, does not cease to be an electronic component, keeping the part of knowledge that characterizes it as an electronic_component, and not as a resistor, unchanged. One cannot exclude the possibility of transitions of an object from class to class. Of course, inheritance can also be used to construct object types with the help of more complex abstract types, but for knowledge bases, it is characteristic of the case of refinement of objects in subclasses.

The main sets of the algebraic system described in Section 2 can be depicted as in Example 2. The common perpendicular to the electronic_component and resistor symbolizes the relationship between the intrinsic and inherited parts of the resistor object (the intersection points are the values of the X and Y fields of the resistor object). At the same time, the possibility of the independent existence of electronic_component objects and the possibility of their "building up" a resistor to objects is emphasized.

To ensure normal operation with complex objects formed during inheritance, the system uses special relationships between the "inherited" and "own" parts of the objects. These links are not visible to the user and are used for internal control over operations with complex objects. It is understood that the following integrity constraints are present:

- in the KB, separate parts of subclasses cannot exist separately;
- each inherited part of a complex object must correspond to only one own part.

Example 4

OBJECT electronic_component:
X integer;
OBJECT resistor Inherits electronic_component:
Y is an integer;
OBJECT variable_RESISTOR FOLLOW RESISTOR:
Z is an integer;

As a result, the internal representation of complex objects (example 4) formed as a result of inheritance looks like chains, as shown in Example 3.

The resistor_variable complex object contains three fields: X, Y, Z. Its own part consists of the element Z. The sequence of relations that implement the inheritance hierarchy will be called the inheritance chain or simply a chain if it is clear what is being discussed. So, we can say that the storage of the values of all fields of the object C is provided separately by the storage of the own elements of all the objects involved in the chain.

With this principle of storing the values of objects, the work of applied programs interacting with the knowledge base can be complicated due to the mismatch of the impedance discussed in Section 2. To weaken its influence, the following approach is proposed. And the variable_resistor generates a description of the structure of knowledge corresponding to the used PL. During program operation, these structures are used as a buffer for transferring knowledge from external memory to other program variables. So, for a knowledge base containing data whose structure is described in Example 4, descriptions of three objects with the same variables will be created.

But if you describe a similar hierarchical structure in an object-oriented language, the random access memory allocated to the values of the elements of the objects will have a different look. In Object Pascal (Delphi), for example, the type description will be like this.

Example 5

```
component = OBJECT
    X: INTEGER;
END

resistor = OBJECT (component)
    Y: INTEGER;
END

Var_resistor = OBJECT (resistor)
    Z: INTEGER;
END
```

which is similar to the above. But for the value of the component object, the amount of memory equal to the size of the X element will be allocated, for the value of the resistor object - equal to the sum of the sizes of the elements X and Y, and under var_resistor - equal to the sum of the sizes of X, Y and Z.

When trying to combine knowledge structures of knowledge base and knowledge base, uncertainty arises in establishing correspondence between elements of knowledge base objects and knowledge objects. That is, when transferring the value, for example, of the component element from the KB to the program, it is not clear whether it should correspond to the value of component.X, resistor.X or var_resistor.X - a typical manifestation of the impedance mismatch.

The essence of solving the problem is as follows. In the representation of knowledge generated by the program for YP, instead of types of elements, dynamic pointers to the values of these types are used. When a connection is established between the knowledge base and the program at the time of its launch, the component.X, resistor.X, or var_resistor.X pointers are installed on the memory location into which the values of the elements will be transferred. Thus, the value of the pointers component.X, resistor.X or var_resistor.X will be the same address. As a result, a rather organic combination of a system of types of object-oriented PLs and knowledge redundancy in the knowledge base is formed.

At the stage of working with an existing knowledge base, it is necessary to take into account the integrity constraints for the inheritance chains indicated above. SQL (Structured Query Language) INSERT, DELETE, UPDATE statements can be applied to object instances to modify the knowledge base, and SELECT statements can be used to transfer object values to the memory working area.

As noted above, the implementation of multiple inheritance is not much complicated relatively simple. Without touching here the collision analysis algorithm, we should mention only one more integrity constraint, traditionally introduced by all known systems: no more than one instance of each type of object of superclasses is involved in the relations of the object's own part with inherited parts.

As for the aggregation hierarchy, it can also be implemented similarly to the inheritance hierarchy using relationships. As the name implies, this hierarchy represents the "Consists of" structure. But in contrast to the inheritance hierarchy, where connections are established "from top to bottom", aggregation generates connections "sideways" - instead of "Consisting of", the structure "In" can be considered. Unlike inheritance, relationships reflecting aggregation can break - a compound object can be "disassembled". At the conceptual design level, the question may be considered whether the constituent parts of objects are complex attributes or relationships implicitly described using the same syntax as simple attributes.

Access to an object through any operator can be associated with a call to the method (s) immanent to this operator in accordance with the specifics of the subject area. The method of the superclass cannot use information about subclasses, while for methods of subclasses it can be about circumstances, about the actions that must be performed with their inherited parts have already been completed to the beginning of the call to its own parts.

IV. Conclusion

The main advantage of using object-oriented programming in developing RTD design systems is the support of methods that facilitate code reuse. However, as many researchers note, the effect of introducing an object-oriented programming technology begins to appear only after 5-10 years [1]. This is

due to the need to accumulate development experience and form a stable and fairly flexible class hierarchy. Obviously, such costs are unacceptable for knowledge engineering tools, where one of the defining requirements is the need to create a "quick prototype". Therefore, an object-oriented toolkit for creating knowledge-based systems should include a library of standard, but fairly easily modifiable objects.

The use of the object-oriented approach in RTD design systems brings to the fore another feature of it, namely the possibility of natural decomposition of a task into a set of subtasks represented by fairly autonomous agents working with knowledge. Today it is the only practical opportunity to work in conditions of exponential growth of complexity (the number of interconnections), typical for designing a RTD.

The development of object-oriented databases, developing into knowledge bases, is now proceeding rapidly throughout the world [2, 12]. Most object oriented data bases are developed based on a relational knowledge model. In this paper, an approach to the implementation of object-oriented properties was described based on a network model of knowledge.

An important feature of the development is to ensure the compatibility of knowledge structures of object oriented data bases with the main object-oriented languages. The combination of an effective strategy for storing complex objects in the form of their own parts along the inheritance chain with generally accepted type systems makes it possible to widely use object oriented data bases together with object-oriented software systems, which is used by the authors when developing an object-oriented hybrid expert system for RTD designing in MTUCI.

References

[1] Avdeev V.A. *Interactive workshop on digital circuitry execution on Delphi*. Saratov: Vocational Education, 2017

[2] Alekhin V.A. *Electronics and circuitry. Abstract of lectures using computer simulation in the environment of "Tina-Ti": a multimedia electronic tutorial*. Saratov: University education, 2017.

[3] Arkhipov S.N. *Circuit design of telecommunication devices: a teaching aid.* Novosibirsk: Siberian State University of Telecommunications and Informatics, 2015.

[4] Afonin V.L. *Intellectual robotic systems*. Moscow: Internet-University of Information Technologies (INTUIT), 2016.

[5] Dolin G.A. Development of a distibuted database of electronic components for synthesis and simulation of electronic devices. *In journal: INFORMATION SOCIETY TECHNOLOGIES proceedings of XIII International scientific-technical conference*. MTUCI, 2019. Pp. 215-219..

[6] Dolin G.A. The algorithm of automatic circuit design of the RTD. *In the collection: The educational environment today and tomorrow Materials of the X International Scientific Practical Conference*. Moscow. : MIT Publishing House, 2015, pp. 276-280.

[7] Dolin G.A. Selection and development of methods for circuit design of electronic equipment and communication systems. *In the collection: The educational environment today and tomorrow Materials of the X International Scientific and Practical Conference*. Moscow: MIT Publishing House, 2015, pp. 280-283.

[8] Dolin G.A. Through automated circuit design of radio engineering devices. *Applied Research and Technology: Proceedings of the International Conference ART2015* (May 27-29, 2015, Moscow). Moscow: MTI, 2015, pp. 61-64.

[9] Dolin G.A. Development of intelligent CAD for equipment and systems of info-communications. *Abstracts of the report on participation in the X International Scientific Conference "Information Society Technologies".* Moscow: MTUCI, 2016.

[10] Korobova I.L. *Decision Making in Knowledge-Based Systems*. Tambov: Tambov State Technical University, 2012.

[11] Orlova M.N. *Circuit design: a course of lectures*. Moscow: Publishing House MISiS, 2016.

[12] S. S. Dymkova, "Prototype of the Information System for Promoting Publications of Scientific and Educational Organizations in the Field of Wave Electronics and its Applications," *2018 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF)*, St. Petersburg, 2018, pp. 1-4. DOI: 10.1109/WECONF.2018.8604368.

[13] V.B. Kozyrev, "Class-E amplifier with a parallel filtering circuit," *Synchroinfo Journal*, vol. 2, no. 1, pp. 2-5, 2016.

[14] A.V. Smirnov, "Modeling OFDM signal distortion in a non-linear amplifier with memory effects," *Synchroinfo Journal*, vol. 4, no. 1, pp. 12-17, 2018

[15] O. V. Varlamov and I. V. Chugunov, "Modeling of efficiency UHF class-D power amplifier with bandpass sigma-delta modulation," *2017 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SINKHROINFO)*, Kazan, 2017, pp. 1-3. DOI: 10.1109/SINKHROINFO.2017.7997508.

[16] A. O. Bolotov, R. G. Kholyukov and O. V. Varlamov, "EER power amplifier modulator efficiency improvement using PWM with additional sigma-delta modulation," *2018 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO),* Minsk, 2018, pp. 1-4. DOI: 10.1109/SYNCHROINFO.2018.8456955.

[17] O.V. Varlamov, A.V. Pestryakov, I.V. Chugunov, "Research of RF power amplifiers based on the method of separate amplification of modulated signal components via the use of software and hardware simulator," *Synchroinfo Journal*, vol. 1, no. 6, pp. 3-8, 2015.

[18] O. V. Varlamov and E. P. Stroganova, "Frequency extension circuit for EER transmitters operating with electrically short antennas," *2018 Systems of Signals Generating and Processing in the Field of on Board Communications*, Moscow, 2018, pp. 1-5. DOI: 10.1109/SOSG.2018.8350577.

[19] V. N. Gromorushkin, O. V. Varlamov, A. V. Dolgopyatova and A. A. Voronkov, "Operation Problems of the EER Transmitter with Narrowband Antenna," *2019 Systems of Signals Generating and Processing in the Field of on Board Communications*, Moscow, Russia, 2019, pp. 1-5. DOI: 10.1109/SOSG.2019.8706736.

[20] R.Yu. Ivanushkin, O.A. Yuryev, "Ways of construction of VHF digital radio broadcasting transmitters," *Synchroinfo Journal*, vol. 4, no. 2, pp. 16-20, 2018.