УДК 519.626, 519.6, 621.276.12
DOI: 10.32626/2308-5916.2020-21.43-51

Andriy Verlan, PhD,
Jo Sterten, PhD
Norwegian University of Science and Technology, Gjovik, Norway

INTELLIGENT OBJECT-ORIENTED APPROACH TO DYNAMIC ENERGY SYSTEMS' MODELLING

Proposed object-oriented approach and general architecture of intellectual software for mathematical modelling of dynamic energy systems. Introduced and considered in detail an architecture of knowledge bases for modelling of systems described by linear integral equations. Considered a knowledge-based system as a composition of a specific functional network and an expert system.

Key words: *modelling, simulation, dynamic systems, intelligent software, integral equations, knowledge base, functional network, expert system.*

Introduction and problem set up. New techniques for modelling of complicated energy systems are being rapidly developed. There is a broad spectrum of available simulation packages, however the main scientific problem which remains actual is a need in advancing of intelligent support for end-users. Here innovative mathematical modelling methods and corresponding intelligent software development concept should be applied. Generally, any scientific problem, including the indicated above, can be described by means of algebraic, differential, integral or other mathematical equations, graphs, logical descriptions etc.

There are many methods for dealing with these model descriptions, many of them have got their software implementation. If a user could specify exactly which method should be applied for solving current task, it would not encounter serious problems. The user could call appropriate procedure or even create a new code for the algorithm. But the typical problem is that it's unclear which proper method to use for solving the task, also the parameters of the method could be unknown. Frequently it requires deep knowledge in mathematics and modelling, and advanced capabilities to formalize the problem and to build a mathematical model. Then user deals with fuzzy descriptions instead of exact models.

Another problem is the following. There are many techniques of choosing and optimizing parameters of specific algorithms. These techniques are commonly implemented in the programs performing the algorithms. But it appears useful to move the parameters selection, optimization etc. from the level of specific programs to the kernel of the modelling environment. This approach makes program engineers' work significantly effective.

Therefore, the problem of creating advanced intellectual environments for solving practical problems of computer modelling of dynamic energy objects appears to be very urgent.

Object-oriented approach and intelligent software development concept used. According to the common approach, when user makes a query to the dialog system, the system should accomplish three stages:

- translation of the query to the inner language of the system;
- processing the query formulated in the inner language for getting answer;
- reverse translation of the answer formulated in the inner system language to the user's language.

For dynamic energy objects considered in this article, the language of mathematical formulae and mathematical equations can be regarded as the inner language of the modelling system. Therefore, we can consider the following proposed more detailed stages:

- translation of fuzzy user's query to a mathematical model;
- solving of the model for getting inner answer;
- reverse translation of the answer.

This paper is focused mainly on the second stage. Conversely to the first and the third stages, the second stage does not depend upon the subject. We consider a mathematical model being known. The problem then is to automate selection of algorithms and their parameters as well as choosing relevant forms of data representation.

Some general principles of creating knowledge bases on the algorithms for solving model equations given in [1]. According to the approach proposed in current paper, a functional network, which is the particular case of more common semantic networks, should be constructed. Here we consider application of this approach to the specific, however very powerful type of equations used for modelling, — to the linear integral equations of the following kind [2, 3]:

$$\int_a^b K(t,s)\,y(s)\,ds + y(t) = u(t). \qquad (1)$$

As specified above, generally the main tasks in the problem of creating modern intellectual environments for dynamic energy objects' modelling are: translation of user's queries to the inner languages of mathematical models; and automation of selecting the proper methods and getting optimal parameters for these methods.

More formally, the solving of the user's query Q can be described as following composition:

$$R = L_S^{-}(U_M(L_S^{+}(Q))), \qquad (2)$$

where R is the system reply, L_S^+ is the translation of original query which depends upon the semantic context S. This semantic context depends on the knowledge about the subject, the user's features, the goals of the dialog, the language, the own opportunities of the system, etc. The result of this translation is the specific mathematical model. L_S^- is the inverse translation. U_M is the solving operator for the obtained model; it takes into account the knowledge base M about mathematical methods. This knowledge base doesn't depend on the subject. It contains knowledge about the representing of mathematical entities (functions, vectors, matrixes, operators etc.), about the methods for solving specific equations, etc.

Here we consider the operator U_M only. To specify this operator firstly we should describe the knowledge base. So, the objective is developing the object-oriented approach which considers the processing as an interaction of some interconnected classes. We can introduce different levels of such description. Here we consider four levels: the *black box* level; the *generating process* level; the *functional network* level and the *data structures* level.

The black box level is the highest one. It deals with the connections determined by the operator equation

$$A\xi = \eta. \qquad (3)$$

Both ξ and η are objects of the class FUNCTION but their roles in the equation are different. We introduce two roles: KNOWN and UNKNOWN. Then any function g can be represented as

$$g = IS\_A(FUNCTION, Role, ID), \qquad (4)$$

where *ID* is a set of parameters which distinguish this function from the others.

Then the connection is described in the following way:

$$Operator\ (IS\_A(FUNCTION, Role1, ID1),$$
$$IS\_A(FUNCTION, Role2, ID2), IS\_A(FUNCTION, Role3, ID3). \qquad (5)$$

The *Operator* can be regarded as a method of some other class. But it can be described as a specific class which depends upon the *Role*. This class is formed by three subclasses:

- *Q* is an operator for solving the direct problem which appears when *Role*1 = *KNOWN*, *Role*2 = *KNOWN*, *Role*3 = *UNKNOWN*;
- *R* is an operator for solving the inverse problem which appears when *Role*1 = *UNKNOWN*, *Role*2 = *KNOWN*, *Role*3 = *KNOWN*;
- *S* is an operator for getting *A*; in this case *Role*1 = *KNOWN*, *Role*2 = *UNKNOWN*, *Role*3 = *KNOWN*.

The methods for solving specific equations and the software which implements these methods can be described as objects of the classes *Q*, *R*, *S*.

The next level is the level of generating processes. It depends upon the kind of equations. Here we consider the indicated above linear integral equations of the (1) kind.

There is a wide choice of methods and specific software modules for solving these equations (for instance given in [4-10]). According to the proposed approach given specific programs should be integrated into the knowledge base, and this base has to provide the user possibility/tools to solve the three types of tasks:

- direct tasks ($y(t)$ and $K(t, s)$ are known; $u(t)$ is to be found);
- reverse tasks ($u(t)$ and $K(t, s)$ are known; $y(t)$ is to be found);
- identification tasks ($u(t)$ and $y(t)$ are known; $K(t, s)$ is to be found);

The particular case of such equations is the Fredholm integral equations of the 1-st kind [11-14], which have many important applications, for example, the signal restoration problem. This problem is commonly formulated in the following way: to obtain a required signal from experimental data by means of solving an operator equation

$$Ay = u, \qquad (6)$$

where u is an available experimental signal, y is a required/sought signal, A is a certain distorting operator, here we regard it as an integral operator.

This problem is incorrect since the inverse operator does not exist or isn't limited. We can apply methods of solving such problems based on specific regularization procedures. These procedures should significantly rely upon the available preliminary information about the solution. There are many procedures of this type but there is a lack of general recommendations how to apply them properly. The most common algorithms are Tikhonov method [15, 16], Lavrentiev method [17], etc.

The information model which describes this problem is formalized as a triplet $V_H = <Q, M, F>$, where Q is a set of generating random processes; functions $u(t)$ and $y(t)$ may be regarded as realizations of these processes; M is a set of model spaces where the realizations may be projected to; F is a set of various dependencies between the elements of $\sigma(\Omega^* \cup M)$, where Ω^* is a set of all random processes' realizations from Ω, $\sigma(\Omega)$ is a set of all subsets of the set Ω.

Model spaces determined by some sets of coordinate functions are crucially important. These are, for example, spaces determined by Fourier, Karhunen-Loeve and other expansions.

The Karhunen-Loeve expansion is known to be an optimal linear expansion in respect of data compression. Moreover, Fourier spectral functions of non-stationery random processes depend upon two variables. Therefore, use of Fourier expansion for analysis of non-stationery processes encounters difficulties. Then the Karhunen-Loeve expansion might be applied instead of the Fourier one [18].

Another positive feature of the Karhunen-Loeve expansion appears when we are solving inverse tasks described by integral equations. This is just

the particular case considered in this article. If we expand input functions by the eigenfunctions of the kernel, then we can use eigenfunctions method which enables to solve inverse problems in a more convenient way.

The next level is the level of the functional network. Following the principles of conceptional programming [19], we develop a specific functional network for integral equations (1). Here we apply the object-oriented approach and consider our functional network as a composition of some basic objects.

The general architecture of the proposed functional network is represented in the figure 1.

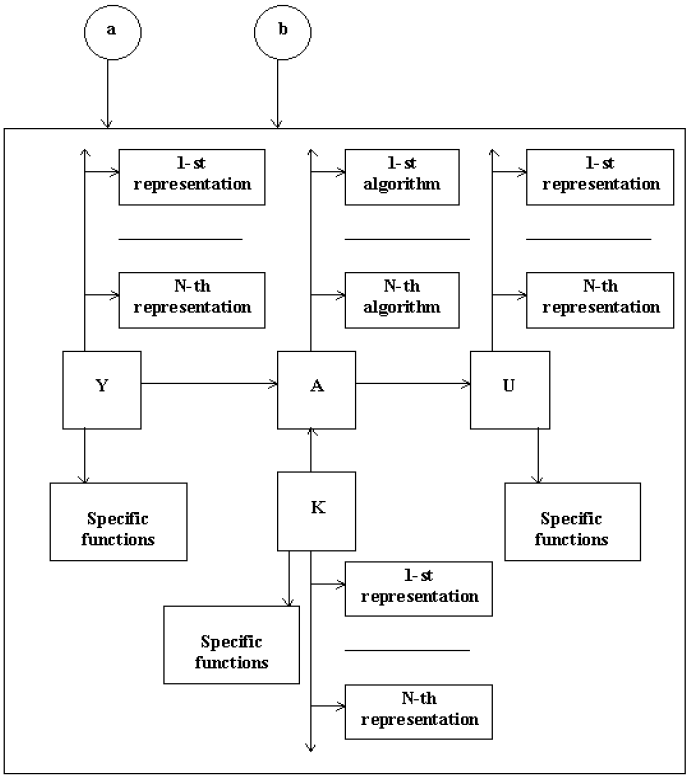

Fig. 1. *General architecture of the functional network*

The functional network, describing the whole model, can be seen as the top-level object. It consists of three basic objects *Y*, *A*, *U* and *K*. They describe $y(t)$, transformation operator, $u(t)$ and the kernel $K(t, s)$ respectively.

It is important, that the objects *Y*, *U*, *K* demand two lines of descendants. The first line is the representation line. It describes the different representa-

tions of the functions such as the values in the discrete points, spectral coefficients of Fourier, Karhunen-Loeve etc. The second line is the line of specific functions. Eventually, the descendants of the object *A* are the programs which perform the specific algorithms for solving direct and reverse tasks.

Functions $y(t)$, $u(t)$, $K(t, s)$ can be represented in the following way:

$$Object = IS\_A(FUNCTION, Role, M, ID), \quad (7)$$

where *M* is a certain model space. Moreover, here we can introduce the new class *VECTOR* which describes functions in some model spaces.

The nodes of the functional network are interconnected connected. The links are the following:

- links between different representations of the same function;
- links between input and output functions; these links can be realized by specific methods.

Then the accomplishment of the user's task can be considered as the searching of a path through the graph from an initial object to the target object.

This can be specified for each individual case. Thus, if the user wants to solve a direct task, he/she should specify the descendants of the *Y* by pointing them in interactive way. These descendants are the specific function and its representations. Then the user should specify the kernel. The linked objects are activated automatically until this reaches descendants of *U*.

If user wants to solve the reverse task, he should specify $u(t)$ and $K(t, s)$ in the same way. The activation is transmitted through the network in the reverse order.

Thus, this provides the selection of the proper method and its parameters. The environment must enable the user to specify the algorithm for solving the problem. The situation when the selected algorithm demands another representation of the input function is very typical, then the links between different representations of the same function should be involved.

Another general situation arises when there are different algorithms for solving the problem, or the user doesn't know which parameters are to be applied for the method. So, this demands another part of the knowledge base — an expert system. This expert system should accumulate an experience of solving integral equations and contain recommendations about applying specific algorithms. This may be, for example, a production system with the sort of rules as the following: «If the function has representation *G* with parameters *H*, the algorithm *X* with parameters *Y* should be applied».

For reflecting possible uncertainties, these rules may be fuzzy. Let us consider the case with one parameter only, other cases are similar. Then, the rules of the production system may be formulated, for example, in the following way:

$$G\,[a_1, a_2] \rightarrow (\mu_1(H)), (\mu_2(c))\,H(c). \quad (8)$$

This means that if a function is represented in a model space G and its coordinate in this space varies from a_1 to a_2, then the fuzzy function μ_1 is a degree of certainty that the algorithm H should be applied; the fuzzy function μ_2 is a degree of certainty that this algorithm should be applied with the parameter *c*. Then the method and its parameters may be determined by minimizing respective fuzzy functions. Here also should be considered the function which determines the rules of changing fuzzy functions while accumulating the experience.

On the data structures level, all structures of the classes should be specified in detail.

According to the general object-oriented approach proposed, we consider at least two categories of users dealing with the environment:

- end-users who apply the environment for solving their practical tasks;
- experts who determine rules of applying specific algorithms and links between the nodes of the functional network. Experts can also create programs for specific algorithms; the special language should be suggested for this purpose.

Now we can summarize the general structure of the intellectual modelling environment, which includes the following main modules:

- end-user's interface;
- expert's interface;
- interpreting system, which should provide translation of fuzzy formulated queries to specific mathematical models as well as return translation;
- knowledge-based system for mathematical modelling which is the combination of functional network (such network for the particular case was described above) and expert production system determining the use of the specific algorithms;
- simulation system for experimental investigations;
- archive database for storing results of experiments.

Conclusions. Thus, the concept and principles formulated above, particularly, proposed object-oriented approach and general architecture of intellectual software, introduced knowledge-based architecture for modelling the linear integral equations described systems as a composition of specific functional network and expert system, creates fundamentals for development of intellectual simulation program environment for solving practical problems of dynamic energy objects' computer modelling (using suggested program languages C++, Java and Matlab/Simulink).

References:

1. Do & Nguyen Hien & Mai. A Method of Ontology Integration for Designing Intelligent Problem Solvers. *Applied Sciences.* 2019. DOI: 9.3793.10.3390/app9183793.

2. Wazwaz A. Linear and Nonlinear Integral Equations: Methods and Applications. 2011. DOI: 10.1007/978-3-642-21449-3.
3. Verlan A., Sterten Jo. Implementation of Integral Explicit Macromodels by means of Quick-Acting Algorithms. *Mathematical and computer modelling. Series: Technical sciences.* 2018. Vol. 18. P. 26-33. DOI: 10.32626/2308-5916.
4. Burton T. A. Volterra integral and Differential Equations. 2nd ed. *Mathematics in science and Engineering*, 202. Elsevier, 2005.
5. Jerri A.J. Introduction to Integral Equations with Applications. Seconded. Jhon Wiley and Sons, 1999.
6. Nadir M. Solving linear integral equations with Fibonacci polynomial. *Malaya Journal of Matematik.* 2018. Vol. 6. P. 711-715. DOI 10.26637/MJM0604/0001.
7. Adawi A. Fadi A., Husein J. A Numerical Method for Solving Linear Integral Equations. *International Journal of Contemporar Mathematical Sciences.* 2009. Vol. 4. P. 485-496.
8. Kameda T. A general method for solving linear integral equations. II. *Proceedings of the Physico-Mathematical Society of Japan.* 2020.
9. Mirceski V. Modification of the step-function method for solving linear integral equations and application in modelling of a voltammetric experiment. *Journal of Electroanalytical Chemistry — J ELECTROANAL CHEM.* 2003. Vol. 545. P. 29-37. DOI 10.1016/S0022-0728(03)00086-X.
10. Verlan A., Sizikov V. Integral equations: methods, algorithms, programs. Kiev : Naukova dumka, 1986.
11. Brosy N. Fredholm integral equation. 2020. DOI: 10.1002/9783527809080.cataz07129.
12. Abdul-Majid W. Fredholm Integral Equations. 2011. DOI: 10.1007/978-3-642-21449-3_15.
13. Georgiev S. Generalized Fredholm Integral Equations. 2016. DOI: 10.2991/978-94-6239-228-1_5.
14. Altürk A. On multidimensional Fredholm integral equations of the first kind. *Journal of Inequalities and Special Functions.* 2017. Vol. 8. P. 85-95.
15. Machado M., Margotti F., Leitao A. On Nonstationary Iterated Tikhonov Methods for Ill-Posed Equations in Banach Spaces. 2018. DOI: 10.1007/978-3-319-70824-9_10.
16. Argyros I.K., Santhosh G., Shobha E. Discretized Newton-Tikhonov method for ill-posed hammerstein type equations. *Communications on Applied Nonlinear Analysis.* 2016. Vol. 23. P. 34-55
17. Favini A., Pandolfi L. Multiscale Lavrentiev method for systems of Volterra equations of the first kind. *Journal of Inverse and Ill-posed Problems — J INVERSE ILL-POSED PROBL.* 2008. Vol. 16. P. 221-238.
18. Wang L. Karhunen-Loeve expansions and their applications. 2008.
19. Hartley R. An Overview of Conceptual Programming I. 2020.

ІНТЕЛЕКТУАЛЬНИЙ ОБ'ЄКТНО-ОРІЄНТОВАНИЙ ПІДХІД ДО МОДЕЛЮВАННЯ ДИНАМІЧНИХ ЕНЕРГЕТИЧНИХ СИСТЕМ

Запропонований об'єктно-орієнтований підхід та загальна архітектура інтелектуального програмного забезпечення для математичного моделювання динамічних енергетичних систем. Представлено та де-

тально розглянуто архітектуру баз знань для моделювання систем, що описуються лінійними інтегральними рівняннями. Розглянутий система, заснована на знаннях у вигляді композиції конкретної функціональної мережі та експертної системи.

Ключові слова: *моделювання, динамічні системи, інтелектуальне програмне забезпечення, інтегральні рівняння, база знань, функціональна мережа, експертна система*.

Отримано: 3.09.2020

УДК 004
DOI: 10.32626/2308-5916.2020-21.51-60

О. О. Гордєєв*, канд. техн. наук,
К. П. Леонтієв**

* Університет банківської справи, м. Київ,
** Науково-виробниче підприємство «Радій», м. Кропивницький

МОДЕЛЬ ЖИТТЄВОГО ЦИКЛУ ДЕФЕКТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Процес розробки програмного забезпечення включає в себе обов'язковий додатковий процес забезпечення якості програмного забезпечення, який являє собою сукупність заходів, що охоплюють всі технологічні етапи розробки, випуску та експлуатації програмного забезпечення інформаційних систем, що проводяться на різних етапах життєвого циклу програмного забезпечення для забезпечення необхідного рівня якості програмного забезпечення. Одне з основних завдань такого процесу полягає в знаходженні і усуненні дефектів програмного забезпечення. Дана робота присвячена формальному представленню життєвого циклу дефекту програмного забезпечення. Модель життєвого циклу дефекту програмного забезпечення розглядається як ланцюжок, який починається з помилки розробника і закінчується відмовою програмного забезпечення. У статті подається загальна структура моделі дефекту життєвого циклу програмного забезпечення, яка включає в себе помилку розробника, помилку оператора, прихований дефект у програмному забезпеченні, активний дефект у програмному забезпеченні, помилку обчислення, збій або відмову, породжену вразливість, активовану вразливість, несанкціоноване управління та несанкціонований доступ до даних. Така модель деталізується в набір патологічних ланцюжків, які структурно представляють модифікації життєвого циклу дефекту програмного забезпечення з урахуванням природи виникнення самого дефекту програмного забезпечення. Серед патологічних ланцюжків виділяють наступні: фізичний, проектування, розробки та взаємодії.