S-Modeling: an Introduction to Updated Theory

V. D. Ilyin

Federal Research Center «Computer Science and Control» of the Russian Academy of Sciences, Moscow, Russia

Abstract. The review describes the basics of updated theory of symbolic modeling of arbitrary objects in a human-machine environment (S-modeling). The theory of S-modeling includes languages for a formalized description of an extensible system of S-modeling notions, a description of the core of this system and classes of basic tasks for constructing and manipulating S-models. The theory of S-modeling is considered as a methodological platform for the scientifically based development of information technologies and the human-machine environment of S-modeling and digitalization of various types of activities (S-environment). S-modeling uses all kinds of symbols (audio, visual, etc.) implementable in the S-environment. S-models are studied as entities having three interrelated representations in the S-environment: symbolic, code and signal. The construction of S-models is carried out according to the rules corresponding to the classes of basic S-modeling tasks. The typing of s-modeled objects is defined. Refined definitions of classes of basic S-modeling tasks are given.

Keywords: Symbolic Modeling (S-modeling), S-modeling Theory, S-symbol, S-code, S-signal, S-environment, Basic Tasks of S-modeling.

DOI 10.14357/20718632220406

Introduction

Without effective symbolic modeling of the studied entities, the development of science, technology and other types of intellectual activity is impossible [1–2]. Achieving a goal in the course of some activity implies a clear idea of the problems to be solved. When achieving non-trivial goals (to design a machine, develop information technology, etc.), the symbolic representation of the idea in the human-machine environment for problem solving (S-environment) is the most productive [3–4].

This approach has a number of proven advantages. Firstly, by analyzing the symbolic model of the idea (device scheme, task specification, etc.), we can check whether the model corresponds to the idea, and if it does not, make corrections to the model. Secondly, on a model recognized as corresponding to the idea, it is possible to verify the validity of the idea itself. And if verification is successful, it is possible to make a decision on the feasibility of implementing the idea. Otherwise – to engage in a change of idea. The idea of achieving a goal using symbolic models is embodied in many technologies for various types of activities. The most successful technologies prohibit behavior that does not comply the rules (attempts of unacceptable behavior have as much chance of success as attempts to play poker against a chess program).

A distinctive features of *S-objects* that exist in the *S-environment* (files of books, articles, videos, electronic maps, computer programs, etc.) are easy copying without distortion, easy distribution and storage of copies (in comparison with physical models, layouts of scientific and technical facilities, etc.) [3–4].

Markuping text fragments and writing formulas. For markuping text fragments, the

S-modeling language *TSM* (*Textual Symbolic Modeling*) is used:

□<text fragment>□ means a definition;

:=: is a separator which is placed after the defined notion (before the definition);

≈ is a substitute for the phrase "the same as";

∘ <text fragment>∘ means an example;

◊<text fragment>◊ means a note.

Italics are used to highlight the concepts and sentences, to which the author wants to attract the attention, and also for variables names [3–4].

The presented results. The results were obtained in the performance of research work "Modeling of social, economic and environmental processes" (№ 0063-2016-0005) under the state task of FANO of Russia for the Federal research center "Informatics and control" of the Russian Academy of Sciences.

1. S-Modeling: Basic Notions

□ ***S-symbol*** :=: a substitute for a natural or invented object, denoting this object and being an element of the system for constructing symbolic messages (texts, computer programs, etc.), designed for human or robot perception. □

∘ In S-modeling, the Russian alphabet together with punctuation marks is considered as *a system of text symbols* for constructing messages according to the rules of the grammar of the Russian language (each element of the alphabet is a substitute for the sound used in *speech messages*); Braille for the blind – as *a system of textured symbols* for constructing text messages designed to be perceived with the touch of the fingers; musical notation, *a system of musical symbols* – as a means of constructing graphically presented *musical messages*; *a system of chess graphic symbols* – as a means of visual representation of chess positions. ∘

In computers, smartphones, and other *S-machines* [3–4], the S-symbol is represented in the form of *S-code*, intended for constructing, storing, transmitting and interpreting symbolic messages.

□ ***S-code*** :=: a substitute for an S-symbol or symbolic message, designed to construct, store, transmit and interpret symbolic messages using S-machines. □ ∘ Morse codes, codes of The Unicode Standard. ∘

□ ***S-signal*** :=: a physically realized representation of S-symbol, designed for perception by human senses (or robot sensors), or an S-code representation, designed to be received by hardware of S-machines.

In S-modeling, the definition of a system of notions is considered as a description of its S-model, accompanied by an indication of the application area.

□ ***S-model of a system sc of notions*** :=: < collection set^{sc} of notions >, < family *rel* (set^{sc}) of dependencies given on set^{sc} >.

For each notion of system *sc*, a set of values is defined.

The application area of the definition is specified by the description of the types:

– *the correspondent* (for whom the definition is intended to be interpreted);

– *the purpose*, in the process of achieving the one the definition is advisable to be applied (∘ tasks, in the study of which the definition can be useful ∘);

– *the stage* where it makes sense to use the definition (∘ problem statement, development of a solution method ∘). □

∘ System *tr* of notions named *a triangle* :=: < set^{tr} ≈ collection of notions >, < *rel* (set^{tr}) ≈ family of dependencies defined on set^{tr} >.

In *tr*, the elements of set^{tr} are the sides (a, b, c), angles (α, β, γ), perimeter p, etc. The family *rel* (set^{tr}) includes $p = a + b + c$; $\alpha + \beta + \gamma = \pi$, etc.∘

∘ The system $tr^{\pi/2}$ of notions of a right triangle can be defined as a specialization of *tr*: $tr^{\pi/2} \approx tr$ [:: $\alpha = \pi/2$] (by adding the restriction $\alpha = \pi/2$, which distinguishes a subset of triangles where the value of one of the angles is equal to $\pi/2$). ∘

The representation of dependencies between notions in the form of solvable tasks is a necessary condition for constructing quantitative S-models of notions systems.

□ ***S-task*** :=: { *Formul*, *Rulsys*, *Alg*, *Prog* }, where *Formul* is the statement of the task; *Rulsys* is the set of systems of *mandatory and orienting requirements for solving the task*, aligned with *Formul*; *Alg* is the union of sets of algorithms, each of which corresponds to one element from *Rulsys*; *Prog* is the union of sets of programs, each of which is assigned to one of the elements of *Alg*.

The task statement *Formul* is a pair { *Mem*, *Rel* }, where *Mem* is the set of notions of the task, where the partition *Mem* = *Inp* U *Out* (*Inp* ^ *Out* = 0) and the set *Rel* of dependencies between notions are specified, where *Rel* defines the binary relation *Rel* < *Inp* * *Out*. The set *Mem* is named *the task memory*; *Inp* and *Out* are its input and output, the values of which are supposed to be set and found, respectively. □

□ *S-algorithm* :=: a system of rules for solving a task (corresponding to one element of *Rulsys*), which allows, in a finite number of steps, to put the resulting set belonging to *Out* to one-to-one correspondence with a given data set belonging to *Inp*. □

□ *S-program* :=: S-algorithm implemented (in a high-level programming language, machine-oriented language and/or in a system of machine instructions), presented in the form of a message that determines the behavior of S-machine task solver with specified properties. The S-algorithm exists in symbolic, code and signal forms connected by the translation relations]. □

◊ In general case, the sets *Rulsys*, *Alg* and *Prog* can be empty: the number of their elements depends on the degree of knowledge of the task. ◊

Descriptions of the use of set elements:

– *Rulsys* includes a specification of the task solver type (autonomous S-machine, networked S-machines cooperation, human-S-machine cooperation, etc.); requirement for information security, etc.;

– *Alg* includes data on the admissible modes of the task solver (automatic local, automatic distributed, interactive local, etc.), requirements for the result obtained, etc;

– *Prog* includes data on programming languages, operating systems, etc.

◊ Each program is accompanied by links to sets of test cases. ◊

The notion of *task constructive object* (*tco*) is introduced along with the number of other notions derived from it [2, 4–7]. They form the complex of notions used in *s*-modeling of the process of transition from the problem to the programmed problem. This complex should allow the software developer to formulate his ideas in *tco*-terms and to advance step by step towards the producing of the specified program system, keeping in mind the meaning of every transition. The *tco*-concept is used to support the «stage to stage» transition from a task formulation (*what*-representation of task is being programmed) to a program (final *how*-representation).

TCO is presented by the finite tree, where task formulation corresponds to the tree root and programs correspond to the leaves. The intermediate (on the way from formulation to program templates) are necessary for taking into account the implementation specification.

□ ***The S-model of the knowledge system sk*** :=: < *ca* ≈ S-model of the notions system *sc* >, < set^{lng} ≈ S-model of the set of message languages interpreted on *ca* >, < set^{intr} ≈ S-model of the set of interpreters of *ca*-messages composed in languages from set^{lng} >. □

Interpreting the message on *ca*:

1. constructing an output message based on a given input message (messages are presented in languages from the set set^{lng});

2. analysis of the output message (whether changes are required in *ca*);

3. if required, initiate changing the *ca*; if not, end.

Typing of s-modeled objects. □ *Type X* :=: a set *X* whose elements have a fixed set of attributes and a family of acceptable operations. It can have subtypes called *type X specializations* and supertypes called *type X generalizations* □.

□ *Specialization of type X* :=: generating of the *subtype X* [::r*ule*] (here the double colon "::" is the symbol of *specialization*) with a family of relations, expanded by the addition of the relation *rule*. Allocates a subset *X* [::*rule*] of the set *X*. *Specialization* is also called the result *X* [::*rule*] of this generating (*X* > *X* [::*rule*]). □

A type *specialization* defined by a sequence of added relations *X* [::(r*ule*1)::*rule*2] is a *specialization* of type *X* [::r*ule*1] on the relation *rule*2. The number of specializing relations in the sequence is unlimited. In this case, the names of the relations preceding the last one are enclosed in parentheses, and before the opening bracket of each pair of brackets is a double colon.

□ *Generalization of type Z* :=: generating of its *supertype Z* [#r*ule*] by *weakening* (here # is the *weakening* symbol) a relation *rule* from a family of relations corresponding to type *Z*. The exclusion of a relation is considered its ultimate weakening. □

2. About OBRAZ Language for Task Knowledge Representation

The conception and formalization of task knowledge representation are developed to allow the building of the task knowledge base of the system of computer aided program construction [2, 5–7].

The central element of *OBRAZ* is a *notion* defined by its name and specification. A notion specification is a set of other subordinate notions (attributes) and the relations between them. A language statement usually contains a notion definition including such relations as a*ffiliation* and *inheritance*. Language statements can also describe restrictions for a notion value set and equivalence of notions as a special case of restriction.

The notion can have several definitions that express various points of view. The language interpreter considers different definitions as separate notions. A user can set up his own point of view to the problem area, selecting a single definition among multiple possible ones.

The program construction environment has no means of evaluating task relations stored in its database. This determines the absence of variables in *OBRAZ*, because there is no temporary data to store.

◊ The *notion specification* is close to the concept of *object class* in object-oriented languages. ◊

To be brief, we'll sometimes skip in subsequent paragraphs words «specification» or «description», so one should consider terms «notion» and «task» as «notion specification» and «task specification». Text enclosed in /* and */ (multiple lines) is a comment. Text between // and end of line is also a comment.

A notion representation. Some notions in *OBRAZ* have built-in specifications. There are the most general notions such as «text», «number», «set» for which *OBRAZ* language supports constants and special syntax extensions that increase readability and compactness of notation. There are also notions that are used to build the task based models, such as «task relation», «task memory element», «input», «output», «task graph», «query». All these notions are called predefined.

Operations on notion specifications. The concatenation operator «+» adds to its left-side operand all attributes, relations and constraints that are contained in the right-side specification. A notion specification defined by concatenation of some base notions contains a specification of its base. Those attributes that appear in a result specification from specifications of other notions are called *inherited* ones, unlike own attributes that were defined inside curly braces. The exclusion operator «–» selects only those attributes and relations from the left-side specification that are not defined in the right-side specification. The latter can contain some notions that are unknown in the left-side specification – these notions are ignored. A specification to be excluded can be represented by a notion name or as an explicit list of attributes and relations enclosed in curly braces.

The group of two or more specifications separated by intersection operator «*» defines a new specification containing only those attributes and relations that are defined in every member of this group. Relations defined using operators «=» and «<>» are more than restrictions on the value sets. The operator «=» also means synonymy, mutual concatenation of specifications and establishing of links between memory elements of *tco*-construction. The operator «<>» prohibits the concatenation and equalizing of the notions it connects and similarly for their derivatives.

A task relation. To specify some notion *x* as a task relation one should derive it from *trel*:

x: *trel* + {…};

Concatenation of *x* attributes {…} with *trel* specification gives to *x* inherited attributes *x.mem*, *x.input* and *x.output*.

The t*ask relation* notion has special built-in derivatives: *function*, *equation*, *program*.

◊ The built-in *OBRAZ* program notion is a template and cannot specify any program code generation process. ◊

There are some notions in the construction system knowledge base that are derived from «program» and define proper attributes to describe construction in various special environments.

A task relation graph (*tr-g*raph). Every notion can be considered as a *tr*-graph if it contains some task relations or other *tr*-graphs that are *tr*-graph vertices. The union set of memory elements of all tasks in *tr*-graph forms *tr*-graph memory. Memory element equivalence of *tr*-graph vertices forms vertices memory intersections that are called *tr*-graph edges. Every group of equivalenced notions with at least one task memory element in the group, forms one *tr*-graph memory element. All members of such a

group are synonyms. Every task memory element that isn't connected with any other notion also forms one *tr*-graph memory element.

3. S-(Message, Data, Information)

□ *S-message* :=: a finite sequence of S-symbols designed for recognition and interpretation by the recipient, or its S-code that meets the requirements for basic S-tasks solvers. □

∘ S-models of systems of notions and knowledge systems, which present the results of the study of certain entities (objects of research); programs that determine the behavior of S-machines; web pages and document files – all these are S-messages. ∘

□ *S-file* :=: a named unit of storage of the S-machine message code (data or program) on a drive (SSD, hard disk, etc.) of computer device (desktop, laptop, smartphone, digital camera, etc.). □

□ *S-data* :=: S-message required to solve a certain task or a family of tasks, presented in a form designed for recognition, transformation and interpretation by a task solver (a program or person). □

□ *S-information* :=: the result of interpreting a message on the S-model of a system of notions. To extract information from a message, it is necessary to have:

– an accepted message presented in a form designed for recognition and interpretation by the message recipient;

– models of systems of notions stored in memory, among which – the necessary one for interpretation of the received message;

– mechanisms for finding the necessary model, interpreting the message, presenting the result of interpretation in the form of a message and writing it to memory. □

∘ The result of interpretation of the m^a message presented in language a, received by the translator (human or robot) – translated to the m^b message in language b, is the information extracted from the m^a message. ∘

4. The General Method and Classes of Basic S-Modeling Tasks (S-Tasks)

□ *The general method of S-modeling :=:* a constructive proof of the existence of S-model of an arbitrary object, representable in S-environment. □

Studying the properties and regularities of S-modeling at each stage of the S-modeling theory development allows to determine the classes of basic S-modeling tasks. Nowadays, the S-tasks presented in Table 1 are the most relevant.

Tab. 1. Classes of basic S-tasks

Name of the class of basic S-tasks	Basic S-tasks
S-representation	Creation of interconnected systems of S-(symbols, codes, signals), specification languages, programming languages, queries languages; representation of S-(messages, data, information), S-models of systems of notions and knowledge systems [4–22].
S-transformation	Converting S-messages (∘ speech ↔ text; analog ↔ digital; uncompressed ↔ compressed; *.doc ↔ *.pdf, etc. ∘) [2–7].
S-recognition	A necessary but insufficient condition for recognition - is the presentation of S-message in a format known to recipient. When this condition is met, the tasks of matching the sample models or matching the properties of the recognized model with the properties of sample models are solved [2–7].
S-construction	Construction of new S-objects from previously created S-objects (∘ tasks specifications, programs, systems of notions, knowledge systems, etc. ∘), presented as constructive S-objects [2–7].
S-interpretation	Interpretation supposes the existence of an accepted S-message, a S-model of the system of notions on which it should be interpreted, and an interpretation mechanism. ∘ To interpret a web page presented on a monitor screen, a person uses systems of notions stored in his memory. For S-machine microprocessor, the S-messages to be interpreted are the codes of the S-machine commands and data; for the compiler, the source code of the program is the S-message to be translated [2–7]. ∘

Tab. 1. Classes of basic S-tasks (end of the table)

Name of the class of basic S-tasks	Basic S-tasks
S-interaction	In this class, the tasks of interaction in S-environment (man – S-machine; S-machine – S-machine) are studied. Senders and recipients of S-messages, means of sending, transmitting and receiving messages are classified. Systems of messaging rules (S-network protocols), S-network architectures, service-oriented architectures, document management systems are being developed [3–24].
S-(saving, accumulating and searching)	This class includes the related S-tasks of saving, accumulating, and searching for S-messages. Memory devices, their management mechanisms, forms of storage and accumulation, methods of accumulation and search, databases and program libraries are studied here and typed [2–7].
S-security	S-tasks of this class are designed to prevent and detect vulnerabilities, perform access control, protect against unauthorized use, malware, and message interception [4, 25–26].

References

1. Newell, A., and H. Simon. 1976. Computer science as empirical inquiry: symbols and search. Communications of the ACM. 19(3):113–126. doi: 10.1145/360018.360022.
2. Ilyin, V. D. 1989. Sistema porozhdeniya programm [The system of program generating]. Moscow: Nauka, 264 p.
3. Ilyin, V. D. 2017. Simvol'noye modelirovaniye (S-modelirovaniye) [Symbolic modeling (S-modeling)]. Bol'shaya rossiyskaya entsiklopediya – elektronnaya versiya [The Great Russian Encyclopedia – electronic version]. Available at: https://bigenc.ru/technology_and_technique/text/4010980 (accessed July 21, 2022).
4. Ilyin, A. V., and V. D. Ilyin. 2009. Osnovy teorii s-modelirovaniya [Fundamentals of the theory of S-modeling]. Moscow: Institute of Informatics Problems of the Russian Academy of Sciences. 143 p. Available at: https://www.elibrary.ru/item.asp?id=25784971 (accessed July 21, 2022).
5. Ilyin, A. V., and V. D. Ilyin. 2012. S-modelirovaniye zadach i konstruirovaniye programm [S-modeling of tasks and construction of programs]. Moscow: Institute of Informatics Problems of the Russian Academy of Sciences. 146 p. Available at: https://www.elibrary.ru/item.asp?id=25816802 (accessed July 21, 2022).
6. Ilyin, V. D. 1995. A methodology for knowledge based engineering of parallel program systems. In: Forsyth, G., Moonis, A. (eds.) The Eighth International Conference Industrial and Engineering Applications of Artificial Intelligence and Expert Systems , IEA/AIE 95, Melbourne, Australia, 6–8 June, 1995. Gordon and Breach. 805–809.
7. Ilyin, A. V., and V. D. Ilyin, 2021. Updated Methodology for Task Knowledge Based Development of Parallel Programs. In: Silhavy R., Silhavy P., Prokopova Z. (eds.) Data Science and Intelligent Systems. CoMeSySo 2021. Lecture Notes in Networks and Systems. Vol 231. Springer, Cham. 319–328. doi: 10.1007/978-3-030-90321-3_25.
8. Licklider, J., and W. Clark. 1962. On-line man-computer communication. In: AIEE-IRE '62 (Spring) Proceedings of the May 1-3, 1962, spring joint computer conference. 113–128.
9. Cerf, V., and R. Kahn. 1974. A Protocol for Packet Network Intercommunication. IEEE Transactions on Communications. 22(5):637–648. doi: 10.1109/TCOM.1974.1092259.
10. Jamsa, K. 2013. Cloud computing. Jones & Bartlett, Learning Burlington. 322 p.
11. Kay, A. 1975. Personal Computing. Palo Alto: Learning Research Group. Xerox Palo Alto Research Center. 30 p.
12. Berners-Lee, T. 1989. Information Management: A Proposal. CERN. Available at: https://www.w3.org/History/1989/proposal-msw.html (accessed July 21, 2022).
13. Berners-Lee, T. 2010. Long live the Web. Scientific American 303(6). Available at: https://www.scientificamerican.com/article/long-live-the-web/ (accessed July 21, 2022).
14. Kim, R. 2011. Efficient wireless communications schemes for machine to machine communications. Comm. Com. Inf. Sc. 181(3):313–323.
15. Lien, S., T. Liau, C. Kao, and K. Chen. 2012. Cooperative access class barring for machine-to-machine communications. IEEE T. Wirel. Commun. 11(1):27–32.
16. Pereyra, C., C. Liu, and S. Jayawardena. 2015. The emerging Internet of Things marketplace from an industrial perspective: A survey. IEEE T. Emerging Topics Computing 3(4):585–598. doi: 10.1109/TETC.2015.2390034.
17. Kravchenko, V., and D. Shirapov. 2018. Logic-Functional Modeling of Nonlinear Radio Engineering Systems. 2018 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon). IEEE. 1–6. doi: 10.1109/FarEastCon.2018.8602769.
18. Rojek, I., D. Mikołajewski, et al. 2020. Digital Twins in Product Lifecycle for Sustainability in Manufacturing and Maintenance. Applied Sciences 11(1):31. DOI: 10.3390/app11010031.
19. Semeraro, C., M. Lezoche, et al. 2021. Digital twin paradigm: A systematic literature review. Computers in Industry 130:103469. DOI: 10.1016/j.compind.2021.103469.

20. Liu, K., L. Lei Song, et al. 2022. Time-Varying Error Prediction and Compensation for Movement Axis of CNC Machine Tool Based on Digital Twin. Industrial Informatics IEEE Transactions on 18(1):109–118. doi: 10.1109/TII.2021.3073649.
21. Nguyen, H., R. Trestian, et al. 2021. Digital Twin for 5G and Beyond. IEEE Communications Magazine 59(2):10–15. doi: 10.1109/MCOM.001.2000343.
22. Jia, P., X. Wang, et al. 2021. Digital-Twin-Enabled Intelligent Distributed Clock Synchronization in Industrial IoT Systems. IEEE Internet of Things Journal 8(6):4548–4559. doi: 10.1109/JIOT.2020.3029131.
23. Rathore, M., S. Shah, et al. 2021. The Role of AI, Machine Learning, and Big Data in Digital Twinning: A Systematic Literature Review, Challenges, and Opportunities. IEEE Access 9: 32030–32052. doi: 10.1109/ACCESS.2021.3060863.
24. Zhang, S., C. Kang, et al. 2020. A Product Quality Monitor Model With the Digital Twin Model and the Stacked Auto Encoder. IEEE Access 8: 113826–113836. doi: 10.1109/ACCESS.2020.3003723.
25. Yang, B., R. Huang, et al. 2021. Efficient Lattice-Based Cryptosystems with Key Dependent Message Security. Applied Sciences 11(24):12161. doi: 10.3390/app112412161.
26. Lu, X., F. Wang, et al. 2021. A Universal Malicious Documents Static Detection Framework Based on Feature Generalization. Applied Sciences 11(24):12134. doi: 10.3390/app112412134

Ilyin V. D. Doctor of Science in technology, Professor, Federal Research Center «Computer Science and Control» of the Russian Academy of Sciences, 40 Vavilova str., Moscow, 119333, Russia, e-mail: vdilyin@yandex.ru