New Developments of the Computer Language Classification Knowledge Portal

Aleksandr Akinin
Novosibirsk State University,
Novosibirsk, Russia
Email: akinin3113@gmail.com

Alexey Zubkov
Novosibirsk State University
Novosibirsk, Russia
Email: ortoslon@gmail.com

Nikolay Shilov
Institute of Informatics Systems,
Novosibirsk, Russia
Email: shilov@iis.nsk.su

***Abstract*—During the semicentennial history of Computer Science and Information Technologies, several thousands of computer languages have been created. The computer language universe includes languages for different purposes (programming, specification, modeling, etc.). In each of these branches of computer languages it is possible to track several approaches (imperative, declarative, object-oriented, etc.), disciplines of processing (sequential, non-deterministic, distributed, etc.), and formalized models, such as Turing machines or logic inference machines. Computer language paradigms are the basis for classification of the computer languages. They are based on joint attributes which allow us to differentiate branches in the computer language universe. We have presented our computer-aided approach to the problem of computer language classification and paradigm identification in a recent paper *Development of the Computer Language Classification Portal* (Proc. of Ershov Informatics Conference PSI-2011, Lect. Not. in Comp. Sci., v.7162). In the present paper we discuss new developments of our project: (1) the pre-alpha version of the Portal is online, and (2) the reasoner is a model checking engine for a paraconsistent (inconsistency-tolerant) description logic.**

***Keywords*: computer languages, computer paradigm, classification, knowledge portal, description logic, paraconsistency, inconsistency-tolerance, model checking.**

I. Introduction: The Problem of Computer Language Classification

Let us start with a sketch of motivation for our research. Please refer [12] for more details.

We understand by a computer language any language that is designed or used for automatic information processing, i.e. data and process representation, handling and management. A classification of some universe (the universe of computer languages in particular) consists in means of identification and separation of items/entities/objects, classes and their roles, and navigation between them.

The *History of Programming Languages* poster by O'REILLY is well known [15]. It represents chronological and influence relations between 2500 programming languages. Due to the number of existing computer languages alone, there is a necessity for their systematization or, more precisely, for their classification. At the same time, classification of already developed and new computer languages is a very important problem for Computer Science, since software engineers and information technology experts could benefit by a sound framework for computer language choice of components for new program and information systems.

Drawing an analogy between Computer Science and other sciences, one may assume that classification of computer languages could be done in the style of Linnaeus (i.e., a taxonomy like: Kingdom - Phylum - Class - Order - Family - Subfamily - Genus - Species). For example, look at *Taxonomic system for computer languages* [17].

However, there is a great difference between domains of natural sciences and Computer Science since the former is static while the latter is highly dynamic. In the last decade of the twentieth century everyone can see rapid growth of existing and new branches of computer languages (knowledge representation languages, languages for parallel/concurrent computing, languages for distributed and multi-agent systems, etc.). Each of these new computer languages has its own, sometimes very particular syntax, a certain model of information processing (i.e., semantics or a virtual machine), and its pragmatics (i.e., the sphere of its application and distribution). And though there were rather small groups of computer languages (e.g., Hardware Description Languages), many groups had already been crowded (e.g., Specification Languages) and some of them went through the period of explosion and migration (e.g., Markup Languages). Sometimes computer language experts have difficulties in putting some languages into one definite group. For example, the programming language Ruby: "Its creator, Yukihiro "matz", blended parts of his favorite languages (Perl, Smalltalk, Eiffel, Ada, and Lisp) to form a new language that balanced functional programming with imperative programming"[18]. Rapid generation of new computer languages will continue while new spheres of human activities will be computerized.

We think that a modern classification of the computer languages universe can be built upon the flexible notion of *computer language paradigms*. In the general methodology of science, *paradigm* is an approach to the formulation of problems and their solutions. The contemporary meaning of the term is due to the well-known book [5] by Thomas Kuhn. Robert Floyd was the first who had explicitly used the term

"paradigm" in the Computer Science context. In particular, he addressed "Paradigms of Programming" in his Turing Award Lecture [3]. Unfortunately, R. Floyd did not define this concept explicitly.

Recently Peter van Roy has published the taxonomy *The principal programming paradigms* [19] with 27 different paradigms and advocated it in the paper [10]. Surprisingly, the cited paper does not provide a convincing and concise definition of the notion *Programming Paradigm*. We can refer to the following quotation only: "A programming paradigm is an approach to programming a computer based on a mathematical theory or a coherent set of principles. Each paradigm supports a set of concepts that makes it the best for a certain kind of problem." [10]

In our recent paper [12] we suggested more comprehensive definition for computer paradigm that (we believe) is coherent with the general concept of *paradigm*:

1) Computer paradigms are alternative approaches (patterns) to formalization of information problem formulation, presentation, handling and processing.
2) They are fixed in the form of formal (mathematical) theory and accumulated in computer languages.
3) Every natural class of computer languages is the extent of some paradigm, and vice versa, every computer paradigm is the intent of some class of computer languages.
4) A paradigm can be characterized by a set of problems/application areas that the paradigm fits better than the other ones.
5) The educational value of paradigms is to teach to think different about information problems and to choose the best paradigm to solve them.

II. Methodology: the Syntactic-Semantic-Pragmatic Approach

Categories *syntax*, *semantics* and *pragmatics* are used to characterize natural and artificial languages (including computer languages). Syntax is the orthography of the language. The meaning of syntactically correct constructs is provided through language semantics. Pragmatics is the practice of use of meaningful, syntactically correct constructs. Therefore the approach that is based on features of syntax, semantics and pragmatics could be natural for specification of paradigms and classification of computer languages.

The syntactic aspect of computer language classification should reflect both the formal syntax and the human perspective. Certainly, it is very important for the compiler implementation whether a particular language has regular, context-free or context-sensitive syntax. Thus, syntactic properties of computer languages could be attributes in the classification. These attributes can be brought from formal language theory. But informal annotations (attributes) like *flexibility*, *naturalness*, *style* (supported by a library of good style examples), *clarity* from a human standpoint (including a portion of *syntactic sugar*) become much more important.

The role of semantics for computer languages is well known. But there are several problems with the use of *formalized semantics* in classification of computer languages, the major problems are listed below.

- Poor acquaintance with formal semantics among computer languages users, *more experts, but fewer general users*.
- Prejudice that formal semantics is too *pure in theory* but too *poor in practice*.
- Too many individual semantic systems and notations with different level of formalization are adopted for different computer languages.

Nevertheless, we think that these problems can be solved by development of multidimensional stratification of "paradigmatic" computer languages[1].

For example, educational semantics and formal semantics are two particular semantic dimensions. They can be stratified into *levels* and *layers* as follows.

- The layer hierarchy is an educational, human-centric semantic representation. It should comprise 2-3 layers that could be called *elementary*, *basic*, and *full*. The elementary layer may be an educational dialect of the language for the first-time study of primary concepts and features. The basic layer may be a subset for regular users of the language which requires skills and experience. The full layer is the language itself, it is for advanced and experienced users.
- The level hierarchy is a formal-oriented semantic representation. It should comprise several levels for the basic layer of the language and optionally for some other layers. The levels of the basic layer could be called *kernel*, *intermediate*, and *complete*. The kernel level would have executable semantics and provide tools for the implementation of the intermediate level; the intermediate level in turn should provide implementation tools for the complete level. Implementation of intermediate level should be of semantics-preserving transformation. Please refer to [9] for an example of a three-level hierarchy for the programming language C#.

In contrast to syntax and semantics, pragmatics relies upon highly informal *beliefs* (i.e. expertise and experience) of people that are involved in the computer language life cycle (i. e. design, implementation, promotion, usage and evolution). In other words, we need to represent formally *expert "knowledge"* (i.e. their views and beliefs) about computer languages, related concepts, and relations between computer languages. It naturally leads to the idea of representing this "knowledge" with an ontology. It is just a tradition to call experts' *beliefs knowledge*, since this expertise can be just an authoritative opinion, but not true, while (according to Plato) *knowledge is true belief*. Nevertheless we will follow this tradition in spite of inconsistency with epistemology.

[1] *Paradigmatic languages* are the most typical ones for a particular paradigm (class).

Formal "ontology is the theory of objects and their ties. Ontology provides criteria for distinguishing various types of objects (concrete and abstract, existent and non-existent, real and ideal, independent and dependent) and their ties (relations, dependencies and predication)" [20]. A formal ontology (simply *ontology* in the sequel) of a particular *problem domain* is a formalization of knowledge about objects (entities) of the domain (computer languages for instance), their classes and ties (relations). This knowledge could include empirical facts, mathematical theorems, personal beliefs, etc.

Expert knowledge for pragmatics of computer languages should be formalized in an *open*, *evolving* (i.e. versioned and temporal) ontology that includes syntactic and semantic (both formal and informal) knowledge in the form of annotations and attributes. The *openness* means that the ontology is open for access and editing. Temporality means that the ontology changes in time, admits temporal queries and assertions, and that all entries in the ontology are timestamped. Versioning means that the ontology tracks all its changes. Wikipedia, the free encyclopedia, is a good example an of open and evolving ontology.

III. Towards an Open Temporal Evolving Ontology for the Classification of Computer Languages

A. Existing Ontologies of Programming Languages

History of Programming Languages poster by O'REILLY [15] can be considered as a primitive ontology of programming languages that is neither open nor evolving. Programming languages are the objects in this ontology, but, unfortunately, the poster does not provide any information about classes of objects. The navigation method in this ontology is represented by *influence lines* and *chronology*.

History of Programming Languages (HOPL) [16] is a much better-developed ontology of programming languages, but, unfortunately, it is, too, neither open for editing nor evolving. HOPL represents historical and implementation information about an impressive number (>8500) of programming languages, but hasn't been updated since 2006, and does not deal with any inter-language relations other than *language-dialect-variant-implementation*.

The situation is different with *Progopedia* [21], a wiki-like encyclopedia of programming languages. It is open for editing and is tracing its history. But Progopedia has poor temporal navigation means. While HOPL provides some taxonomy instruments, Progopedia only has a trivial one *language-dialect-variant-implementation*. In comparison with HOPL and the O'REILLY poster, Progopedia is relatively small. At present it contains information about ∼130 languages, ∼70 dialects, ∼300 implementations, and ∼660 versions.

None of the three listed ontologies have means for constructing classes by users or deriving classes, and only manual navigation among the classes is supported. We believe that a more comprehensive ontology is needed to solve the problem of computer languages classification, i.e. identification and differentiation of classes of computer languages and navigation among them.

B. Outlines of our Approach

We develop ontology for computer languages, based on Description Logic (DL) [1], [11], [14]. The objects of our ontology are computer languages (also their levels and layers), concepts/classes (in terms of DL/OWL) — collections of computer languages that can be specified by concept terms (in DL terms), ties (DL-roles or OWL-properties) — relations between computer languages. For example, Pascal, LISP, PROLOG, SDL, LOTOS, UMLT, as well as C, C-light and C-kernel, OWL-Lite, OWL-DL and OWL-full should eventually become objects of the ontology.

Since we understand computer paradigms as specifications of classes of computer languages, and we consider classes of computer languages as DL-concepts (OWL-classes), then we have to adopt DL concepts as paradigms of computer languages: *Every (syntactically correct) DL concept term defines a paradigm that is the concept specified by the term*. In this setting, computer language paradigms and classification is not a taxonomic tree based on property inheritance from sup-class to sub-class, but a formal ontology with navigation by DL means.

Objects (i.e. computer languages) of the ontology could be described with different formal attributes (e.g., formal syntax properties) and informal annotations (e.g., libraries of samples of good style). Let us remark that the list of formal attributes and informal annotations is not fixed but is open for modifications and extensions. Nevertheless, we fix certain attributes and annotations for all objects (but allow to assign an indefinite value for them). For example, we provide the following attributes:

- *date of birth* with various time granularity,
- *URL of an external link* for any non-specified references,
- *try-version* for a link to an easy to install or web-based small implementation (that can be freeware or shareware).

Some elementary concepts/classes in the ontology are also fixed, for example: *has context-free syntax*, *functional languages*, *specification languages*, *executable languages*, *static typing*, *dynamic binding*, etc. A special elementary concept/class is *paradigmatic computer languages*, it comprises few (but one at least) representatives for every elementary concept/class. We expect to borrow more ideas for elementary concepts from [22]. Elements of elementary concepts/classes must be explicitly annotated by appropriate attributes (*has a context-free syntax*, *is a functional language*, *is a specification language*, etc.).

Non-elementary concepts/classes should be specified by DL concept terms. For example, *executable specification languages* is the intersection of *executable languages* and *specification languages*. Since our ontology is an open-world ontology with incomplete information then some problem occurs with *class-complement*. For example, if a language has no explicitly attached attribute *has a context-free syntax*, it does not mean that the language has no CF-syntax, it just

means that *the information is not available*. To resolve the problem, we provide every *positive* attribute (e.g., *has context-free syntax*) by the corresponding *negative* attribute that is the counterpart of positive one (e.g., *DOES NOT have a context-free syntax*).

All elementary concepts/classes (including paradigmatic languages) should be created on the basis of expert knowledge and be open for editing. A special requirement for the proposed ontology should be the following constraint: every legal (i.e. *well-formed*) non-empty concept/class must contain a paradigmatic language. This is common sense: if experts can not point out a representative example of a paradigm, then it should be empty.

Roles/properties in the proposed ontology could also be natural: *is a dialect of*, *is a layer of*, *uses the syntax of*, etc. For example: *C-light is a layer of C*, *OWL uses the syntax of XML*, etc. All listed examples are elementary DL-roles/OWL-properties. Standard (positive) relational algebra operations *union*, *intersection*, *composition*, *role inverse*, and *transitive closure* can be used and are meaningful for construction of new roles/properties. For example, *uses the syntax of a dialect of* is the composition of *uses the syntax of* and *is a dialect of*. Again we have a problem with *role complement*, but we have not fix any solution yet (in contrast to the *class-compliment* problem).

Let us remark that the computer language domain has four domain-specific ties between languages: *is a dialect of*, *is a variant of*, *is a version of*, and *is an implementation of*. Of course these ties must be present in the proposed ontology as elementary DL-roles/OWL-properties. But, unfortunately, there is no consensus about definition of these ties. For example, *Progopedia* [21] considers that an implementation can have a version, while [22] promotes an opposite view that a version can have an implementation. Currently we adopt the following definition.

- *Dialects* are languages with joint elementary level.
- *Variants* are languages with joint basic level.
- *Version series* is a partially ordered collection of variants such that every smaller version is a compatible subset of all later versions.
- *Implementation* is a platform-dependent variant of a language.

Let us remark that several incompatible versions can coexist:Object C and C++ are object-oriented variants of C, but for sure these two languages are incompatible.

Universal and *existential* quantifier restrictions that are used in OWL and DL for construction of new classes/concepts have a natural and useful meaning. An example of existential restriction (in DL notation): a concept $(markup\_language) \sqcap \exists uses\_syntaxof : (\neg\{XML\})$ consists of all computer languages that are markup languages but do not use the syntax of the Extensible Markup Language XML; an example of a language of this kind is LaTeX. An example of a universal restriction and a terminological sentence (in DL notation also) follows: the sentence $\{XML\} \sqsubseteq is\_dialect\_of : (\neg\{ML\})$ expresses that XML is a dialect of any computer language but the functional programming language ML.

IV. Current State of the Project

We started implementation of a prototype of a computer languages classification knowledge portal (that eventually will evolve into an open temporal evolving ontology) for classification of computer languages a year ago [12]. At present, a pre-alpha version of the portal is available online [23].

The prototype does not support full functionality. The prototype is implemented as a web application, so everyone can enter it with a web browser. The interface allows users to view and edit information contained in the portal, which is formed as an ontology.

The main elements of the prototype ontology are computer languages (objects of the ontology), elementary classes of languages (arbitrary, explicitly user-specified subsets of the set of objects), relations between the languages (binary relations over the set of objects), attributes (mappings from the set of languages to some external data types, e.g. text strings, URL's) and the Knowledge Base (Description Logic statements that represent laws of the problem domain of Computer Languages). The data is represented internally as an RDF repository. All these entities can be viewed and modified directly by the user.

Two main services (that are already provided) are the ontology model checker and visualization. The model checker is used for computing classes of objects and ties from specifications (concept and role terms), and for checking consistency of the ontology (data and the Knowledge Base). Visualization is used for displaying classes and ties graphically.

The model checker is an explicit-state model checker for a *paraconsistent* (i.e. inconsistency-tolerant) *description logic* [6], [7], [8] extended by two special constructs for concept terms borrowed from Formal Concept Analysis (FCA) [4], [14], [11]. The underlying paraconsistent description logic uses four-value semantics of Belnap logic [2]. The constructs borrowed from FCA are upper and lower derivatives. (The lower derivative is the same as the *window operator* in DL.) The logic is chosen to handle openness of the ontology and incompleteness and inconsistency of data in the ontology.

Why do we use a model checker as a reasoning tool instead of any available DL inference machine (such as Fact++, Kaon2, etc.)? Because our ontology is for empirical expert knowledge about rapidly developing and changing domain of Computer Languages, not a domain with a set of predefined domain-specific laws. We use an *explicit-state* model checker (not a *symbolic* one) since the domain numbers thousands of objects, i.e. it fits explicit-state representation well.

Why are we developing a self-contained tool for the ontology instead of using some other ontology tool (Protege for instance)? Because we are developing a tool for a small community-oriented ontology for Computer Language experts, where people would like to use a simple interface instead of studying a manual or a tutorial before using the tool.

We would like to emphasize that at present the ontology is an open ontology already. We expect that the ontology eventually will also become versioned and evolving, i.e. will support automatic timestamping, history of all edits, and temporal queries. We would like to hope that our ontology and portal will provide researchers by a sound and easy framework for language specification as well as software engineers and IT managers by tools for language choice.

ACKNOWLEDGMENT

Research is supported by Integration Research Program n.3 (2012-2013) *Ontology Design and Development on base of Conceptualization by means of Logic Description Languages* provided by Siberian Branch, Russian Academy of Science.

REFERENCES

[1] F. Baader, D. Calvanese, D. Nardi, D. McGuinness, and P. Patel-Schneider (editors), *The Description Logic Handbook: Theory,Implementation and Applications*, Cambridge University Press, 2003.

[2] N.D. Belnap, *How a computer should think*, Contemporary Aspects of Philosophy: Proceedings of the Oxford International Symposium, 1977, p.30-56.

[3] R.W. Floyd, *The paradigms of Programming*, Communications of ACM, v.22, 1979, p.455-460.

[4] B. Ganter, R. Wille, *Formal Concept Analysis. Mathematical Foundations*, Springer Verlag, 1996.

[5] T.S. Kuhn, *The structure of Scientific Revolutions*, Univ. of Chicago Press, 1970. (3rd Ed. – 1996.)

[6] Y. Ma, P. Hitzler, and Z. Lin, *Algorithms for paraconsistent reasoning with owl*, Proc. of European Semantic Web Conference 2007, Lect. Not. in Comp. Sci., v.4519, Springer, 2007, p.399-413.

[7] Y. Ma, P. Hitzler, and Z. Lin, *Paraconsistent reasoning for expressive and tractable description logics*, Proc. of the 21st International Workshop on Description Logic, CEUR Electronic Workshop Proceedings, v.353, 2008.

[8] Y. Ma and P. Hitzler, *Paraconsistent Reasoning for OWL 2*, Proc. of Web Reasoning and Rule Systems, Lect. Not. in Comp. Sci., v.5837, 2009, p.197-211.

[9] V.A. Nepomniaschy, I.S. Anureev, I.V. Dubranovskii, and A.V. Promsky, *Towards verification of C# programs: A three-level approach*, Programming and Computer Software, v.32(4), 2006, p.190-202.

[10] P. van Roy, *Programming Paradigms for Dummies: What Every Programmer Should Know*, in New Computational Paradigms for Computer Music, IRCAM/Delatour, France, 2009, p.9-38.

[11] N.V. Shilov, *Formal Models and Methods for Design and Development of Ontologies*, in Formal Logical and Linguistic Models and Methods for Design and Development of Information Systems, Siberian Devision of Russian Academy of Sciences, 2009, p.11-48 (in Russian).

[12] N.V. Shilov, A.A. Akinin, A.V. Zubkov, and R.I. Idrisov, *Development of the Computer Language Classification Portal*, Proc. of Ershov Informatics Conference, Lect. Not. in Comp. Sci., v.7162, 2012, p.340-348.

[13] N.V. Shilov, *Make Formal Semantics Popular and Useful*, Bulletin of the Novosibirsk Computing Center (Series: Computer Science, IIS Special Issue), v.32, 2011, p.107-126.

[14] S. Staab and R. Studer (editors) *Handbook on Ontologies*, International Handbooks on Information Systems. Springer, 2nd edition, 2009.

[15] *History of Programming Languages*, available at `http://oreilly.com/news/graphics/prog_lang_poster.pdf`.

[16] *History of Programming Languages*, available at `http://hopl.murdoch.edu.au/`.

[17] *Taxonomic system for computer languages*, available at `http://hopl.murdoch.edu.au/taxonomy.html`.

[18] *Ruby. A Programmer's best friend*, available at `http://www.ruby-lang.org/en/about/`.

[19] *The principal programming paradigms*, available at `http://www.info.ucl.ac.be/~pvr/paradigms.html`.

[20] *Ontology. A Resource Guide for Philosophers*, available at `http://www.formalontology.it/`.

[21] *Progopedia*, available at `http://progopedia.ru/`.

[22] *The Language List*, available at `http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm`.

[23] *Computer Language Classification*, available at `http://complang.somee.com/Default.aspx`.