# Development of the Computer Language Classification Knowledge Portal

**Nikolay Shilov, Renat Idrisov, Aleksandr Akinin, Aleksey Zubkov**

Nikolay Shilov
A. P. Ershov Institute of Informatics Systems
Lavrentev av. 6, Novosibirsk 630090, Russia
shilov@iis.nsk.su

Renat Idrisov
Novosibirsk State University, Kaptug av. 2, Novosibirsk 630090, Russia
ren@ngs.ru

Aleksandr Akinin
Novosibirsk State University, Kaptug av. 2, Novosibirsk 630090, Russia
akinin3113@gmail.com

Aleksey Zubkov
Novosibirsk State University, Kaptug av. 2, Novosibirsk 630090, Russia
ortoslon@gmail.com

## 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 languages, specification languages, modeling languages, languages for knowledge representation, 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, parallel, distributed, etc.), and formalized models (from Turing machines up to 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. Currently the number of essentially different paradigms is close to several dozens. The study and precise specification of computer language paradigms (including new ones) are called to improve the choice of appropriate computer languages for new Software projects and information technologies. This position paper presents an approach to computer languages paradigmatization (i.e. paradigm specification) and classification that is based on the unified approach to formal semantics, and an open ontology (wiki-styled) for pragmatics, formal syntax and informal "style".

## *1 The Problem of Computer Language Classification*

We understand by a computer language any language that is designed or used for automatic "information processing", i.e. data (including process) representation, handling and management. A classification of some universe (the universe of computer languages in particular) consists in means for class(es) identification, separation and navigation.

During the semicentennial history of development of programming and information technologies, several thousands of computer languages have been created[1]. Due to the number of the 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 it could benefit software engineering and information technology by a sound framework for computer language choice for components of new program and information systems.

At the initial stage of programming and information technology history (years 1950-65), it was possible to classify computer languages chronologically with annotations à la Herodotus' "History", i.e. including lists of authors, their intentions, personal stories, etc. (Refer to [11] for a story of this kind.) The matter is that at the first stage all (almost) computer languages were languages of imperative programming for von Neumann's computers.

But since the late 1960-s the approach in style of the "Father of History" became unacceptable. Since this time the variety of computer languages included not only programming languages, but also specification languages, data representation languages, etc. Some of these branches since the late 1960-s include not only imperative, but also declarative languages (functional in particular).

Between the middle of 1970-s and the early 1980-s, new approaches to computer language design appeared (logical and object-oriented for example). Drawing an analogy between Computer Science and other sciences, we could say that since late 60s, the classification of computer languages could be done either in "Linnaeus style" (i.e. a taxonomy like: Kingdom - Phylum - Class - Order - Family - Subfamily - Genus - Species) or in "Mendeleyev style" (i.e. as a chemical periodic table where the

[1]The *History of Programming Languages* poster by O'REILLY (http://www.oreilly.com/news/graphics/prog_lang_poster.pdf) is well known. The full-scale version of the poster is about 6 m long and contains 2500 entries. The chronology is represented by the temporal axis placed at the top of the poster, version history of individual languages are shown with colored lines, and the influences of programming languages are depicted by weak grey lines. Please refer Appendix B for scaled copy of the poster and zoomed fragment of it.

rows represent periods, and the columns represent groups). For example, look at[2] *Taxonomic system for computer languages* at http://hopl.murdoch.edu.au/taxonomy.html.

However, the 1990-s and the beginning of a new millennium became the time of rapid growth of existing and new branches of computer languages. For example, 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 syntax (sometimes a very specific), a certain model of information processing (i.e. semantics or a virtual machine), and its pragmatics (i.e. the sphere of application and distribution). And though there were rather small groups of computer languages (e.g., Hardware Description Languages), many groups have been already "crowded" (e.g., Specification Languages) and some of them went through the period of explosion and migration (e.g., Markup Languages). Sometimes computer language "Gurus" have difficulties in putting some languages into the one definite paradigm[3] or to any paradigm (e.g. Language of Temporal Ordering Specification LOTOS, Business Process Modeling Notation BPMN). Rapid generation of new computer languages will continue while new spheres of human activities will be computerized. Thereby the situation in computer languages radically differs from that of the natural sciences: in biology or chemistry the situation is much more static, while in computer languages the situation is rather dynamic. Due to these arguments alone, the natural sciences analogies cannot be adequately applied to the classification of computer languages.

We define paradigms of computer languages as specifications of alternative approaches to information processing, accumulated and fixed in the form of computer languages. Computer language paradigms should base on joint attributes, which allow us to differentiate classes in the computer language universe. Paradigmatization is an approach to the specification of paradigms.

Let us remark that Robert Floyd was the first who had explicitly used the concept of "paradigm" in Computer Science, but in a different context and with another meaning. He discussed "paradigms of Programming" in his Turing Award Lecture in 1978 [5]. Floyd referred to Thomas Kuhn's well-known book [7], published just 8 years before. According to T. Kuhn, a paradigm is a method, an approach to the formulation of problems and the ways to solve them. R. Floyd had a similar understanding of programming paradigms. In particular, he wrote: "To the designer of programming languages, I say: unless you can support the paradigms I use when I program, or at least support my extending your language into one that does support my programming methods, I don't need your shiny new languages" [5]. Currently, the number of essentially different paradigms of programming is already several dozens (see, for example, the list of "programming paradigms" at [13]).

[2]Please refer Appendix C for scaled copy of the poster and zoomed fragment of it.

[3] For example, 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" (http://www.ruby-lang.org/en/about/).

## *2 Introducing Syntactic-Semantic-Pragmatic View (approach)*

For natural and artificial languages (including computer languages), the terms "syntax", "semantics" and "pragmatics" are used to categorize descriptions of language characteristics. 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 based on "specific features" of syntax, semantics and pragmatics, could be natural for the specification of paradigms and the classification of computer languages.

The syntactic aspect of computer language classification should take into account formal syntax as well as 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 formal characteristics of computer languages could be attributes for their classification. These attributes can be brought from formal language theory in accordance with Chomsky hierarchy or any other formal classification of formal languages. But the classification based on formal syntax has lost its original value by virtue of development of effective and powerful parsers. In contrast, informal annotations (attributes or characteristics) like "flexibility", "naturalness", "style" (supported by a library of good-style examples), "understandability" from a human standpoint (including a portion of "syntactic sugar") become much more important.

The role of formal semantics for the classification of computer languages is also well known. The major problem with semantics of computer languages is different formalisms and different level of formalization that is adopted for particular computer languages. In programming languages, for example,

- functional language LISP is based on very precise denotational semantics in terms of $\lambda$-calculus,
- the structured subset of a high-level imperative language Pascal has operational and axiomatic semantics [6],
- but formal semantics for imperative languages of C-family is still a research challenge [9, 10].

This difference makes it extremely hard to compare semantics of different computer languages. Nevertheless, we believe that sound unification of approaches to semantics is essential for better classification of computer languages and a proper paradigm definition. We think that the problem can be solved by two-dimensional stratification of paradigmatic[4] computer languages. Each of these languages should be stratified into levels and layers.

- Level hierarchy is the human-friendly semantic (and partially syntactic) representation. It should comprise 2-3 levels that could be called as elementary, basic, and full. Elementary level should be an educational dialect of the language for the first time study of its basic concepts and features. Basic level should be a subset for regular users of the language which requires skills and experience. Full level is the language itself, it is for advanced and experienced users.

[4] We discuss what is "paradigmatic computer language" in the next section. To be short, paradigmatic languages are the most typical ones for a particular paradigm (class).

- Layer hierarchy is the formal-oriented semantic representation. It could comprise up to 3 layers for basic level and (optionally) for other levels. These layers could be called kernel, intermediate and complete. The kernel layer should have an virtual machine semantics and provide tools for the implementation of the intermediate layer; the intermediate layer in turn should provide tools for the complete layer. Implementation of intermediate layer should be of semantics-preserving transformation. Please refer [9] for an example of 3-layer hierarchy for programming language C#.

In contrast to highly mathematical formal semantics, pragmatics relies upon highly informal expertise and experience of people that are involved in the compute language life cycle  (i.e. design, implementation, promotion, usage and evolution). In other words, we need to formalize expert "knowledge" (views) about computer languages, related concepts, and relations between computer languages. It naturally leads to an idea to represent this knowledge about computer language pragmatics as ontology.

"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)" [4]. Roughly speaking, an ontology is a partial formalization of a "knowledge" about a particular problem domain (computer languages for instance). This "knowledge" could be an empirical fact, a mathematical theorem, a personal belief, an expert resolution, a shared viewpoint of a group.

The most popular computer language for ontology representation is Web Ontology Language OWL. It provides an opportunity to use Description Logic (DL) reasoners for automatic consistency checking. Consistency is very important for open evolving temporal ontology. Openness means that the ontology is open for access and editing. Temporal means that the ontology change in time, admits temporal queries and assertions, and that all entries of the ontology have time-stamp. Evolving means that the ontology tracks all its changes. Wikipedia, the free encyclopedia [14], is a good example of open, evolving, and temporal ontology.

## *3 Towards Open Temporal Evolving Ontology for Classification of Computer Languages*

Formalization of expert knowledge for pragmatics of computer languages should be open evolving temporal ontology that includes syntactic and semantic (formal and informal) knowledge in the form of annotations and attributes.

Let us remark, that the *History of Programming Languages* poster by O'RELLY (see Appendix B) already defines an ontology of programming languages in which the class differentiation and navigation method is explicit enumeration of languages, "influence lines" and chronology. The same holds for HOPL (History of Programming Languages at http://hopl.murdoch.edu.au/, see Appendix C). This project represents historical and implementation information about an impressive number (8512) of programming languages. Unfortunately HOPL is not open for editing, is not evolving since 2006, and does not deal with any other inter-language relations than language–dialect–variant–

implementation. Situation is different with the Progopedia the wiki-like encyclopedia of programming languages (http://progopedia.ru/). It is open for editing, traces its history. But both HOPL and Progopedia does not deal with programming paradigms, have restricted temporal navigation. While the HOPL provides some taxonomy instruments, the Progopedia has only a trivial one (language–dialect–variant–implementation). In comparison with HOPL and O'REILLY poster, Progopedia is relatively small. At present it contains information about 51 language, 80 dialects, 187 implementations, and 485 versions. All these "ontologies" do not have means for constructing classes by users and support only manual navigation among the classes. We believe, that a more comprehensive ontology could solve the problem of computer languages classification, i.e. identification and separation of classes of computer languages and navigation among them.

In the proposed ontology for computer languages, objects should be computer languages (including their levels and layers also as objects), concepts/classes (in terms of DL/OWL) – collections of computer languages that can be specified by concept terms (in DL), ties (DL-roles or OWL-properties) – relations between computer languages. For example, Pascal, LISP, PROLOG, SDL, LOTOS, BPMN, UMLT, as well as C, C-light and C-kernel, OWL-Lite, OWL-DL and OWL-full should be objects of the ontology. Since we understand computer language 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 concept terms as paradigms of computer languages. In these settings computer language paradigms and classification will not be taxonomy trees based on property inheritance from sup-class to sub-class. Objects (i.e. computer languages) of the proposed ontology could be attached by 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 open for modifications and extensions. Nevertheless it makes sense to provide certain attributes and annotations for all objects (e.g., an attribute "date of birth" with different time granularity, or annotation "URL of external link" for references) but allow indefinite value for them. In contrast, some other attributes and annotations will be very specific to objects. For example, "try" annotation with a link to easy to install or web-based small implementation (that can be freeware or shareware) makes sense for elementary levels or kernel layers.

We have already discussed a number of examples of concepts/classes in the proposed ontology: "has context-free syntax", "functional languages", "specification languages", "executable languages", "static typing", "dynamic binding", etc. (Other examples can be found at [17].) All listed examples should be elementary DL-concepts/OWL-classes. All elementary DL-concepts/OWL-classes should be explicitly annotated by appropriate attributes ("has context-free syntax", "is functional language", "is specification language", etc.). Nonelementary concepts/classes should be specified by means that are supported by OWL and DL (by standard set-theoretic operations "union" and "intersection" in particular). For example, "executable specification languages" is the intersection of "executable languages" and "specification languages". We have some doubts about "complement", since the proposed ontology will always be open-world ontology with incomplete information. For example, if a language has no explicitly attached attribute "has context-free syntax", it does not mean that the language has no CF-syntax. At present we adopt a temporary solution to use explicit positive (e.g., "has context-free syntax", "is functional language", "is specification language", etc.) and negative attributes (that are counterparts of positive one, e.g., "has NOT context-free syntax", "is NOT functional language", "is NOT specification language", etc.) and use of positive DL concept terms for paradigm specification (i.e. concept terms without complement).

But the proposed ontology should have a special elementary concept/class for "paradigmatic computer languages" that comprises few (but one at least) representative for every elementary concept/class. Of course, all elementary concepts/classes (including "paradigmatic languages") should be formatted on base of expert knowledge and be open for editing. A special requirement for the proposed ontology should be the following constraint: every legal ("well-formed") non-empty concept/class must contain a paradigmatic language (one at least). A background intuition is straightforward: if expert can not point out any representative example of a paradigm, then paradigm should be empty.

Roles/properties in the proposed ontology could be very natural also: "is dialect of", "is layer of", "uses syntax of", etc. For example: "REAL is dialect of SDL", "C-light is layer of C", "OWL uses syntax of XML". All listed examples are elementary DL-roles/OWL-properties. Standard (positive) relational algebra operations "union", "intersection", "composition" and "transitive closure" can be used and are meaningful for construction of new roles/properties. For example, "uses syntax of dialect of" is the composition of "uses syntax of" and "is a dialect of": REAL [8] executional specifications "uses syntax of dialect of" SDL [12]. Again we have some doubts about use of "complement" and "inverse" and, maybe, we will adopt use of explicit complement and inverted for elementary DL-roles/OWL-properties.

Let us remark that computer language domain has four domain-specific ties between languages: "is dialect of", "is variant of", "is version of", and "is implementation of" [15]. Of course these ties must be presented in the proposed ontology as elementary DL-roles/OWL-properties. But, unfortunately, there is no consensus about definition of these ties. For example, some people [18] consider that an implementation can have a version, while some other [17] have an opposite view that a version can have an implementation. Detailed discussion of this issue is a topic for further research, but currently we adopt the following definition. Dialects are languages with joint elementary level. Variants are languages with joint basic level. Version line is linearly ordered collection of variants such that later every earlier "version" is a compatible subsets of all later versions[5]. Implementation is platform-dependent variant of a language.

[5] Here "version" is any element of any version line? i.e. is defined with respect to a particular line. In principle, several independent version lines can coexists. For example, it is possible to say that Object C and C++ are object-oriented variants of C, but for sure these two languages launch different version lines.

Universal and existential quantifier restrictions that are used in OWL and DL for construction of new classes/concepts also could get a natural and useful meaning. An example of existential restriction (in DL notation): a concept. (uses syntax of : ((markup language) □ □{XML}) consists of all computer languages that are markup languages but do not use syntax of Extensible Markup Language XML; an example of a language of this kind is LaTeX. An example of universal restriction and terminological sentence (in DL notation also) follows: a sentence XML: (is dialect of) . (□{ML}) expresses a fact that XML is a dialect of any computer language but a functional programming "Meta Language" ML.

We would like to emphasize that a proposed ontology for pragmatics of computer languages should be open evolving ontology. Openness of the ontology will be supported by wiki technology for editing. Evolution will be supported by the automatic timestamping and history of all edits. Temporality will be supported by temporal extensions of Description Logic for paradigm specification.

Recently we have started implementation of a prototype of a computer languages classification knowledge portal that eventually (we hope) will evolve into Open Temporal Evolving Ontology for Classification of Computer Languages (see Appendix A). We believe that it will provide Computer Language researchers with a sound and easy to maintain and update framework for new language design and language choice/selection tools for new Software engineers and Information Technology managers.

## *References*

[1] Anureev I.S. Ontological Transition Systems. Joint NCC&IIS Bulletin, Series Computer Science, 2007, v. 26. (Accepted for publication.)

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

[3] Borger E. and Stark R. Abstract State Machines: A Method for High-Level System Design and Analysis. Springer-Verlag, 2003.

[4] Corazzon R. Ontology. A Resource Guide for Philosophers. At http://www.formalontology.it/.

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

[6] Hoare, C.A.R. and Wirth N. An Axiomatic Definition of the Programming Language PASCAL. Acta Informatica, 2, 1973, p.335-355.

[7] Kuhn T.S. The structure of Scientific Revolutions. Univ. of Chicago Press, 1970.

[8] Nepomniaschy V.A., Shilov N.V., Bodin E.V., Kozura V.E. Basic-REAL: integrated approach for design, specification and verification of distributed systems. Springer Lecture Notes in Computer Science, v.2335, 2002.

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

[10] Norrish M. C formalised in HOL. PhD Thesis. University of Cambridge, Computer Laboratory, Technical Report 453, 1998.

[11] Ritchie D.M. The development of the C language. ACM SIGPLAN Notices, v.28, n.3, 1993, p.201-208.

[12] Turner K. J. (Editor) Using Formal Description Techniques: An Introduction to Estelle, LOTOS and SDL. John Wiley and Sons, 1993

[13] Programming paradigm. From Wikipedia, the free encyclopedia. At http://en.wikipedia.org/wiki/Programming paradigm.

[14] Wikipedia, the free encyclopedia. At http://en.wikipedia.org.

[15] Pigott D. HOPL: an interactive Roster of Programming Languages. 1995-2006. At http://hopl.murdoch.edu.au/

[16] OWL Web Ontology Language Guide, W3C Recommendation, February 10, 2004. At http://www.w3.org/TR/owl-guide/

[17] Kinnersley W. The Language List. At http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm.

[18] Progopedia. Encyclopedia Yazykov Programmirovaniya. (In Russian) At http://progopedia.ru/.

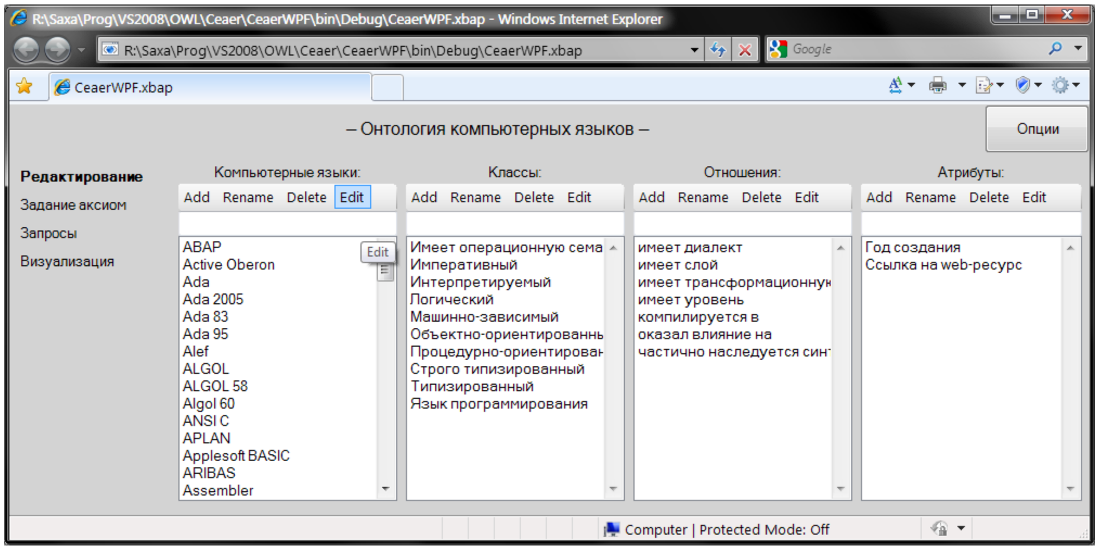
## *Appendix A: a prototype of a computer languages classification knowledge portal*

A prototype of a knowledge portal for computer languages classification has been designed for small-scale experimentation purposes. So it does not support full functionality. The prototype is implemented as a web application, so “experts” (i.e. members of the laboratory) can enter it with a web browser. The interface allows users to observe and edit the information represented by the portal, which is formed as an ontology.

The main elements of the 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 axioms (Description Logic statements).

Each of these types of entities form a list which can be viewed and modified directly by the user. The pictures below illustrate this process.

1. Example of list of ontology elements:


R:\Saxa\Prog\VS2008\OWL\Ceaer\CeaerWPF\bin\Debug\CeaerWPF.xbap - Windows Internet Explorer
R:\Saxa\Prog\VS2008\OWL\Ceaer\CeaerWPF\bin\Debug\CeaerWPF.xbap
Google
CeaerWPF.xbap
– Онтология компьютерных языков –
Опции
Редактирование
Задание аксиом
Запросы
Визуализация
Компьютерные языки:
Классы:
Отношения:
Атрибуты:
Add Rename Delete Edit
Edit
Computer | Protected Mode: Off

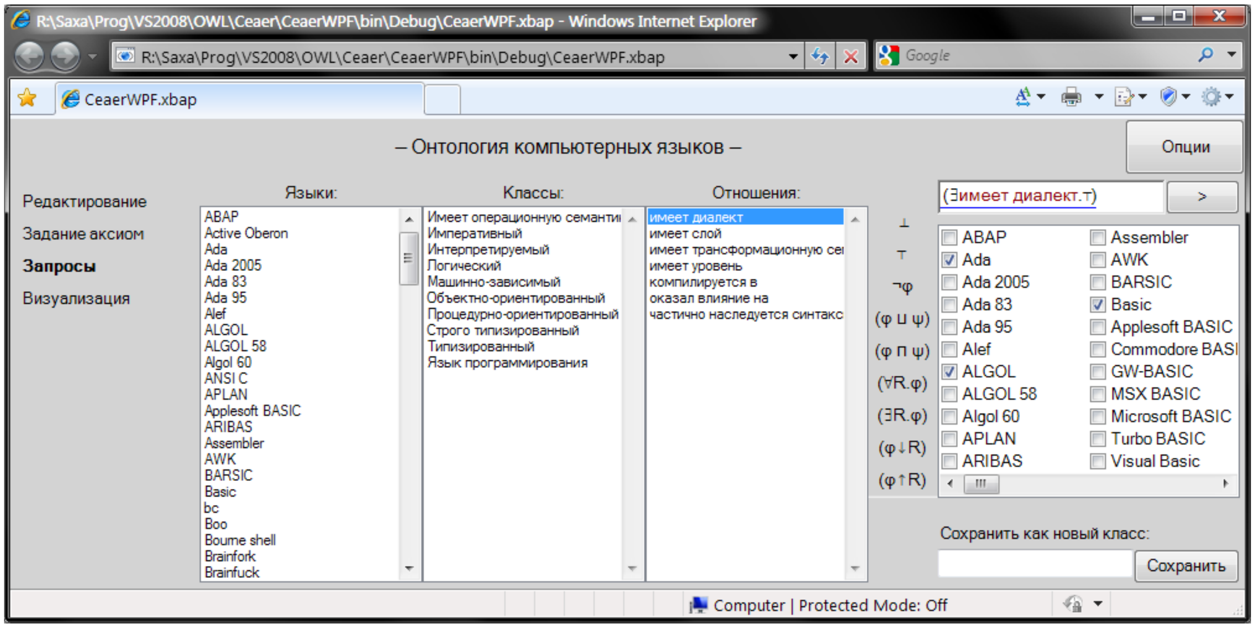
Компьютерные языки:
ABAP
Active Oberon
Ada
Ada 2005
Ada 83
Ada 95
Alef
ALGOL
ALGOL 58
Algol 60
ANSI C
APLAN
Applesoft BASIC
ARIBAS
Assembler

Классы:
Имеет операционную сема
Императивный
Интерпретируемый
Логический
Машинно-зависимый
Объектно-ориентированнь
Процедурно-ориентирован
Строго типизированный
Типизированный
Язык программирования

Отношения:
имеет диалект
имеет слой
имеет трансформационнук
имеет уровень
компилируется в
оказал влияние на
частично наследуется син

Атрибуты:
Год создания
Ссылка на web-ресурс

2. Example of query processing:


R:\Saxa\Prog\VS2008\OWL\Ceaer\CeaerWPF\bin\Debug\CeaerWPF.xbap - Windows Internet Explorer
R:\Saxa\Prog\VS2008\OWL\Ceaer\CeaerWPF\bin\Debug\CeaerWPF.xbap
Google
CeaerWPF.xbap
– Онтология компьютерных языков –
Опции
Редактирование
Задание аксиом
Запросы
Визуализация
Языки:
Классы:
Отношения:
⊥
⊤
¬φ
(φ ⊔ ψ)
(φ ⊓ ψ)
(∀R.φ)
(∃R.φ)
(φ↓R)
(φ↑R)
>
Сохранить как новый класс:
Сохранить
Computer | Protected Mode: Off

Языки:
ABAP
Active Oberon
Ada
Ada 2005
Ada 83
Ada 95
Alef
ALGOL
ALGOL 58
Algol 60
ANSI C
APLAN
Applesoft BASIC
ARIBAS
Assembler
AWK
BARSIC
Basic
bc
Boo
Bourne shell
Brainfork
Brainfuck

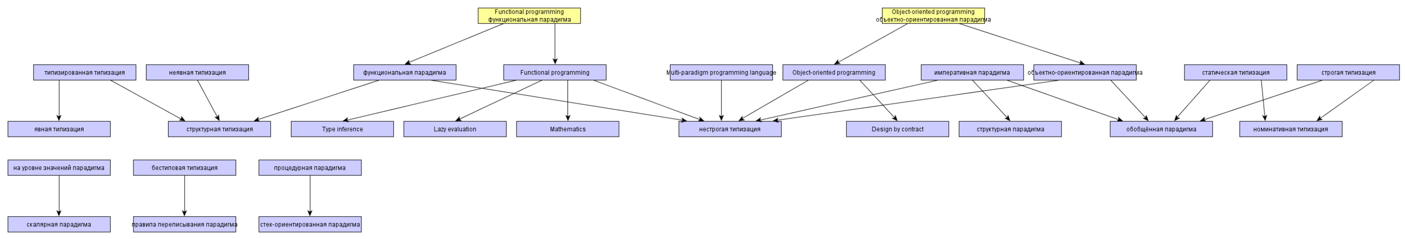
Классы:
Имеет операционную семанти
Императивный
Интерпретируемый
Логический
Машинно-зависимый
Объектно-ориентированный
Процедурно-ориентированный
Строго типизированный
Типизированный
Язык программирования

Отношения:
имеет диалект
имеет слой
имеет трансформационную се
имеет уровень
компилируется в
оказал влияние на
частично наследуется синтакс

(∃имеет диалект.⊤)

| | |
|---|---|
| ☐ ABAP | ☐ Assembler |
| ☑ Ada | ☐ AWK |
| ☐ Ada 2005 | ☐ BARSIC |
| ☐ Ada 83 | ☑ Basic |
| ☐ Ada 95 | ☐ Applesoft BASIC |
| ☐ Alef | ☐ Commodore BASI |
| ☑ ALGOL | ☐ GW-BASIC |
| ☐ ALGOL 58 | ☐ MSX BASIC |
| ☐ Algol 60 | ☐ Microsoft BASIC |
| ☐ APLAN | ☐ Turbo BASIC |
| ☐ ARIBAS | ☐ Visual Basic |

3. Example of data visualization:

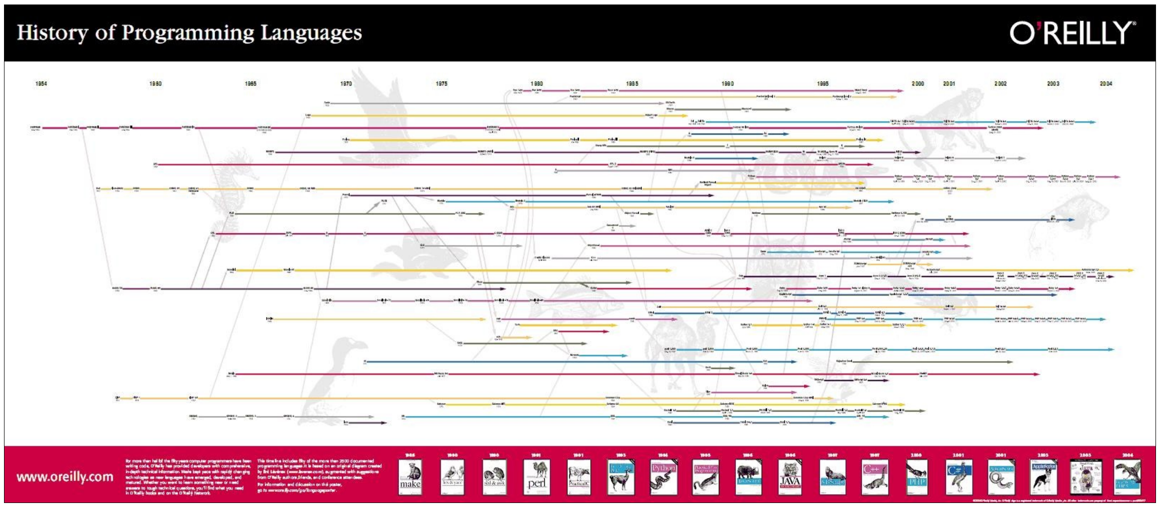


The portal prototype also allows to visualize selected relations between languages. It draws a graph where the vertices are computer languages marked by their names and the edges are the relations between the languages marked by their color.

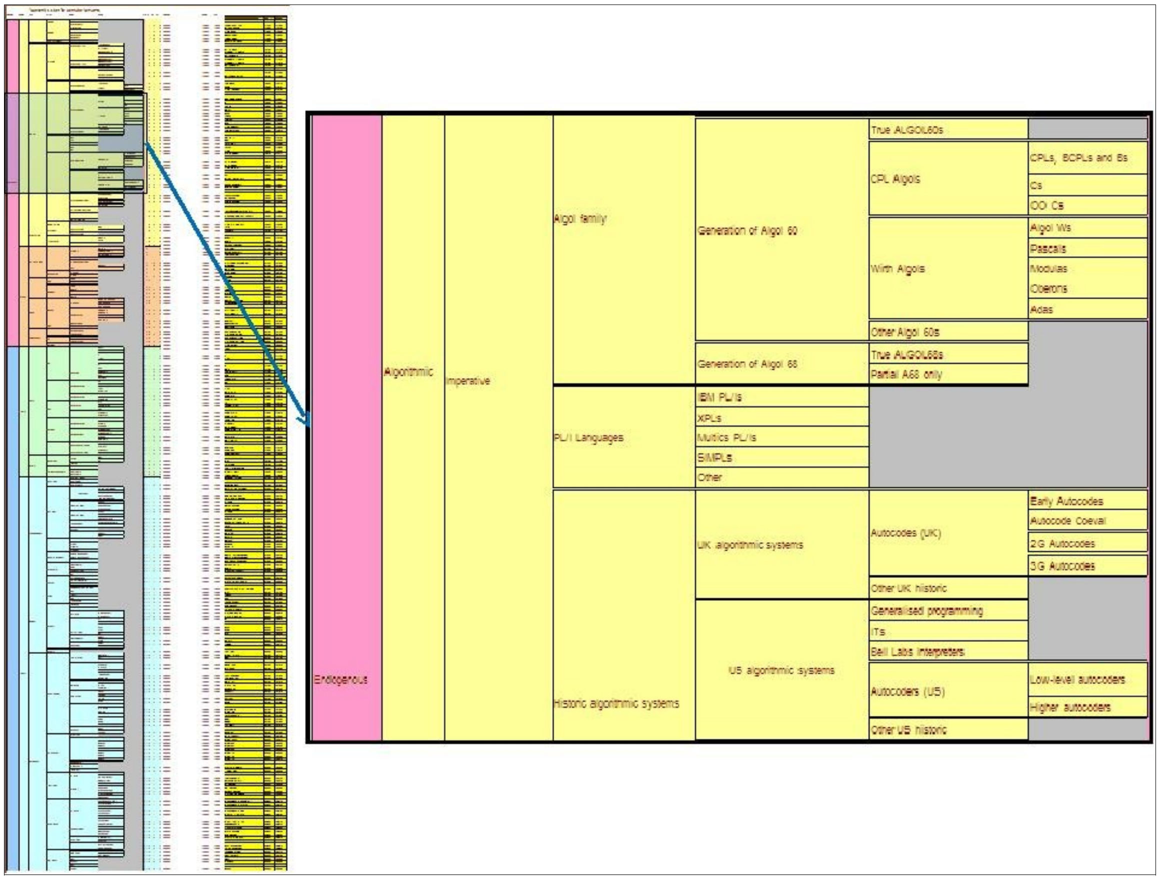
The data in the prototype is represented by an OWL-language database (RDF-repository). In future it will allow us to use some OWL-oriented reasoning tool to perform the consistency control and process the user queries.

The prototype is at the beginning stage of development now. It is planned to expand its functionality by providing a more effective and reliable mechanisms to deal with big quantities of data elements and users, to solve complex queries.

***Appendix B: History of Programming Languages by O'REILLY***
***(**http://www.oreilly.com/news/graphics/prog_lang_poster.pdf**)***



## *Appendix C: HOPL' taxonomic system for computer languages (http://hopl.murdoch.edu.au/taxonomy.html).*



Endogenous
Algorithmic
Imperative
Algol family
Generation of Algol 60
True ALGOL60s
CPL Algols
CPLs, BCPLs and Bs
Cs
OO Cs
Wirth Algols
Algol Ws
Pascals
Modulas
Oberons
Adas
Other Algol 60s
Generation of Algol 68
True ALGOL68s
Partial A68 only
PL/I Languages
IBM PL/Is
XPLs
Multics PL/Is
SIMPLs
Other
Historic algorithmic systems
UK algorithmic systems
Autocodes (UK)
Early Autocodes
Autocode Coeval
2G Autocodes
3G Autocodes
Other UK historic
US algorithmic systems
Generalised programming
ITs
Bell Labs interpreters
Autocoders (US)
Low-level autocoders
Higher autocoders
Other US historic