ABIS: A Language of Intelligent Systems

A.Poletykin, M.Byvaykov and A.Baybulatov
Laboratory of distributed information/analytical, and control systems
V.A. Trapeznikov Institute of Control Sciences of the Russian Academy of sciences (ICS RAS)
Moscow, Russia
poletik@ipu.rssi.ru, ipu31@mail.ru

***Abstract*—The high-level language ABIS has been developed at the ICS RAS for artificial intelligence systems. The basis of the language is the relational data model and forward chaining logical inference based on rules. At present, the language is a corporative software product and is successfully applied to create software of large control systems of nuclear power plants. However, the language is universal and may be applied to develop a large spectrum of software codes in different branches.**

***Keywords*—progranming language, inference engine, data base**

I. INTRODUCTION

There exist many problems whose solution is based on a logical analysis of different-type data via complex heuristic algorithms which are not based on strict mathematical methods while being sequences of looks of data bases, matching data from different sources, logical inferences, information conversion, its filtering, etc. These problems involve an analysis of data archives of an arbitrary nature, maintaining a dialogue with a human, processing flows of measured data, data mining, and many others.

Frequently, to create software codes intended to solve such a kind of problems an approach is used under which the access to the data is implemented by a relation type DBMS with the SQL language, wile as a toll of the analysis algorithms implementation a universal programming language is used, for instant C++ or an other one.

We do not criticize such an approach, although we have a negative experience of its applying within problems when a guaranteed system reply is to be assured. We propose another way, using the ABIS language, which organically combines tools of storing and manipulating data with powerful logical constructions.

The basis of the ABIS language is the mathematical apparatus of conformances proposed by M.A. Zuenkov at the middle of 1980s. But in the sequel, the language has been developing independently and has received its logical completeness at the middle of 1990s.

One should underline that the purposes of creating the ABIS language were increasing the programmers labor productivity and increasing the software code quality. Primordially, it has been being developed as a tool of crating complex software codes in large control systems, where an intelligent data processing, their generalization, and filtering are required.

The problem of creating a commercial version of the language has become actual recently, after proving that the ABIS enables one to develop complex software products possessing high reliability, high operation rate, and low cost. Such an inference has been done in accordance to results of a successful development of a SCADA-system for nuclear power engineering, which enables one to implement intelligent processing hundreds of thousands values of signals and data on-line [1].

II. DATA STRUCTURE OF THE ABIS LANGUAGE

The language supports simple data types: logical variables, strings, integers, floating point numbers, arrays, etc.

But these types are auxiliary. The basic type is sets of facts (factsets). These are sets of relations tuples which resemble those ones used in the relational data bases. In entity, the factsets are data bases. However, the ABIS language uses an "extended" relational data model.

The basic concepts of the model are: *relation*, *attribute*, *domain*, and *tuple*. Let us introduce these concepts by the following simple example. Let be a table entitled as "Supplier" consisting of the following columns: "Enterprise" "Merchandise", "Volume", "Measurement unit", in which two records are:

TABLE I. "SUPPLIER"

Enterprise	Merchandise	Volume	Measurement unit
Electric power plant	Electric power	30	Megawatt
Metallurgical plant	Rolled metal	1000	Thousand of tonnes

In the terms of the relational data models, the table is a *relation*, the columns are *attributes*, a range of values of the records in a column is an attribute *domain*, and a total string of the table is a *tuple* of the relation. The data model is a set of the relations:

$R_1(A_{11},\ldots,A_{1n})$

$R_2(A_{21},\ldots,A_{2m})$

…

$R_k(A_{k1},\ldots,A_{kl})$

where A_{ij} is the attribute *j* of the relation R_i.

Each string (tuple) of the relation $R(A_1,\ldots,A_n)$ may be represented in the form:

$R(A_1=s_1,\ldots,A_n=s_n)$,

where s_i is a concrete value of the attribute A_i.

For instance, the first string of the relation "Supplier" may be written as follows:

Supplier (Enterprise = Electric power plant,
Merchandise = Electric power,
Volume = 30,
Measurement unit = Megawatt)

or in a brief record form (omitting the attributes names):

Supplier (Electric power plant, electric power, 30, megawatt).

Consider now those values which the attributes may take. Here, three basic variants are possible:

- Terminal (definite) value;
- Non-definite (unknown or indifferent) value;
- A reference to a child tuple of a relation of the data model.

In the example "Supplier" considered above, all the attributes values presented are the terminal ones. But if one will consider such a fact:

"There is an enterprise producing 30 megawatts of the electric power",

than to represent it, one would require a tuple of the relation "Supplier" of the following form:

Supplier (_ , electric power, 30, megawatt).

Here, as a value of the attribute "Enterprise", the character (_) is used representing the non-definite value.

Let us consider now another case. Let the "Merchandise" produced by enterprises may not be described by a single terminal value, and one requires to introduce the relation "Merchandise" of, for instance, the following form:

TABLE II. **"MERCHANDISE"**

Name	**Production date**	**Delivery type**
Rolled metal	December	Rail way

In that case, the fact:

"In December, the metallurgical plant produced 10 thousand tones of the rolled metal and shipped it via the rail way"

may be written in the following form:

Supplier (Metallurgical plant,
Merchandise (rolled metal, December, Rail way),
10, thousand tonnes)

Here, the value of the attribute "Merchandise" of the relation "Supplier" is the direct record of the corresponding tuple of the relation "Merchandise" with, in accordance to the terminology of the ABIS-system, is referred as *reference* since in the internal representation (after compiling) it is indeed transformed into an address reference.

III. STRUCTURE OF THE SOFTWRE CODES IN THE ABIS LANGUAGE

The basic software code units of the language are rules. For simplicity, we will not use the formal syntax and will be limited by commonly used concepts.

Each rule contains a condition (IF) and a corollary (THEN) which are processed by an inductive system (inference engine).

The conditions contain a number of sentences which are to be met. The sentences may be simple (comparison for equality, inequality, or their logical combinations), or contain checking contents of certain factsets.

The corollaries also contain a number of sentences, each of them assumes implementing an action: assigning a value to a simple variable, modifying a factset, transfer to another rule, etc.

The sentences of a condition and corollary may contain variables which are assigned within the process of processing the condition and are used under implementing the corollary. This gives rice to the situation that the rules are met under different combinations of the variables values, i.e. multiply.

Let us consider the following example. Let, besides the relation "Supplier" already introduced, be a relation "Consumer" with the same attributes names: "Enterprise", "Merchandise", "Volume", and "Measurement unit".

Then the rule

"If: there is a consumer of a merchandise, then: there should be a supplier of the merchandise"

may be written only by explicit indication both for the supplier and consumer that the speech is on the same merchandise. This enables one to make the mechanism of the variables

If : consumer *(_, _T, _, _)*

Then : supplier *(_, _T, _, _)*.

Here, availability of the same variable *_T* both in the condition and corollary parts of the rules tells that the speech is about the same merchandise. Thus, in the rules of the knowledge-base there is admitted to use also variables as values of the attributes.

Under performance of the deductive system, the basic procedure is matching the facts from the data base and the conditional parts of the knowledge-base rules. The process of matching is reduced to the feature that if in the conditional part of the rule there is a tuple of a relation *R*, then the rule may be successfully matched only with the facts being tuples of the relation *R*, at that the matching process is to be implemented attributewise. The process of matching fact tuples and a rule is referred as *conformance*. Table 3 contains results of conformance for different types of attributes and a rule implemented in the ABIS language. A successful conformance is designated via 1, and unsuccessful, via 0.

TABLE III. A TABLE OF SUCCESS/UNSUCCESS OF CONFORMANCE FOR DIFFERENT TYPES OF FACT ATTRIBUTES AND A RULE

Fact attribute / **Rule sentence attribute**	**Non-definite value**	**Terminal value**	**Reference**
Non-aforsaid variable	1	1	1
Non-definete value	1	1	1
Terminal value	0	Checking for equality	0
Refernce	0	0	Recursion

IV. Input-output functions of the ABIS language

In the ABIS language, all the functions envisioned in the C language are implemented. Besides that, there are used input-output functions of the factsets in the text and binary forms, as well as functions of their transmission through network channels.

V. Current implementation of the ABIS language

The ABIS language is simple. Its description involves 150 text pages with examples and comments. In totality, the Pseudocompiler, interpreter, and debugger involve 14 thousand lines in the C language.

The existing version performs under the operation system Linux.

The current version is oriented to professional programmers and contains a minimally sufficient list of possibilities and services.

VI. Experience of applying the ABIS language

The most considerable application of the language is a SCADA-system applied for organization of monitoring and control of power units of Russian nuclear power plants. Let us list its basic characteristics:

- Number of processed signals is 250 thousands;
- Information processing cycle is not more than 1.7 seconds (in average, 0.7 seconds);
- Automatic reconfiguration under faults of backup equipment;
- Service of 25 workplaces;
- Availability of computer-aided design tools.

The system has been created under a limited budget (USD 300 thousands) during 3 years. The system is maintained by 4 persons jointly with implementing other job duties.

The SCADA-system is an example of a software which solves tasks of processing information flows on-line. To achieve the indexes of speed of operation and reliability, the system has been created as a distributed set of computing processes in the ABIS language which implement parallel information processing as a pipeline.

A typical example of implementing algorithms of the logical analysis of static data in the ABIS language is the program code of monitoring integrity of the file system in order to detect un-authorized deviations. It performs in two modes: the model of learning and the mode of monitoring. In the learning mode, the program code composes a data base of characteristics of important files. In the main mode, the program code monitors availability of deviations. The programs implement rather complex heuristic algorithms, and the number of processed files reaches several dozens thousands. At that, the program code itself is very simple, evident, takes not more than 1 Mb of the RAM and performs fast.

Possible branches of applying the ABIS language are not defined yet. However, with a great confidence one may affirm that their list is not limited by processing data flows and logical analysis of data bases only.

In accordance to results of application, one may emphasize several features of the source code in the ABIS language and particularizes of program codes debugging.

The first feature is “seamlessness” of the internal data with may be easily looked at the stage of debugging and even within the program codes operation.

The second feature is a possibility of simple creating highly effective highly tailored languages to implement certain works. This enables one to create programmable, configurable systems in the ABIS language.

The third feature is the shortness of the program code and small size of the load images. This enables one to run the software codes at any modern computing devices up to home automation and cellular phones.

And, finally, the forth feature is a possibility of creating computing pipelines of a different topology, in which processing algorithms are decomposed and implemented in independent computing processors connected by network channels to transmit intermediate results in the form of the factsets.

VII. Perspectives

We assume that basing on the ABIS language one may create a self-sufficient instrumental platform to develop a software of different intention. For that, one should elaborate a standard of the language, add required libraries, created suitable and modern debugging tools and implement all other necessary works.

VIII. Conclusions

The ABIS language has passed all the required stages in its development and has proven its right for the future.

We believe that it should become available for a broad community of software developers.

At present, we implement preparation works and look for partners and sponsors to create a public version of the ABIS language.

References

[1] M. E. Byvaikov, E. F. Zharko, N. E. Mengazetdinov, A. G. Poletykin, I. V. Prangishvili, and V. G. Promyslov, “Experience from Design and Application of the Top-level System of the Process Control System of Nuclear Power-plant,” Automation and Remote Control. Moscow, vol. 67, no. 5, pp. 735–747, 2006 [Avtomatika i Telemekhanika, no. 5, pp. 65-79, 2006].