Automating Software Design System DESTA

Vladimir A. Lovitsky
Associate Professor
Software Engineering Department
Institute of Radioelectronics
Kharkov, Ukraine
(0572) 409 113 (Fax)

Patricia D. Pearce
Professor, Head of Computing Department
University of Plymouth
Drake Circus, Plymouth
Devon, PL4 8AA, UK
pat@uk.ac.psw.cd
(0752) 232 541 (Office)
(0752) 232 540 (Fax)

Abstract

"DESTA™ " is the acronym for the *D*ialogue *E*volutionary *S*ynthesizer of *T*urnkey *A*lgorithms by means of a natural language (Russian or English) functional specification of algorithms or software being developed.

DESTA represents the computer-aided and/or automatic artificial intelligence "forgiving" system which provides users with software tools support for algorithm and /or structured program development.

The *DESTA* system is intended to provide support for the higher levels and earlier stages of engineering design of software in contrast to conventional CAD systems which provide low level tools for use at a stage when the major planning and structuring decisions have already been taken.

DESTA is a knowledge-intensive system. The main features of the knowledge are *procedures, functions, modules, operating system commands, batch files, their natural language specifications* and *their interlinks.*

The specific domain for the *DESTA* system is a high level programming languages likes Turbo Pascal 6.0.

The *DESTA* system is operational and runs on a IBM PC computer.

1. Introduction

At present software development is the biggest obstacle to major new breakthroughs in computing. The biggest limitation in software development is the failure of imagination that people tend to project: *"A user only really knows what he wants when he sees a finished attempt"*.

How we develop software at present. We tend to develop software in the same way we did it in the 1960s i.e. it's one instruction after the other. We really haven't yet got to the point where CAD system or CASE-type tools help out very much. In order to move toward what we call higher levels of software automation in the future, we are going to be using more standardized systematic-type modules for developing software systems.

The end aim of automatic programming is a complete system without the need to write any code. At present you don't see automatic programming, where you simply say to the computer: *"OK, I need a program to do this, and lo and behold, out it comes"*.

This paper describes aspects of applied research related to the development of an intelligent system *DESTA*. The idea is very simple: we must get lots of different software from lots of different places that must work together and must talk to each other and the output of one can be used as input of the other. Obviously we need to have vast knowledge base for it. The software should be developed from *reusable* software components: ***"software chips"***. To do it we need to consider some general issue:

- Knowledge content, structure, representation, acquisition, and maintenance.
- Inference engine.
- Human-computer interaction, natural language interface, integrated program development environment.

2. Knowledge Base

Software development is an intensely knowledgebased activity. The functioning or activity of any intelligent system (natural or artificial) can be reduced to solving a set of suitable problems, 93% of which belong to so-called "ill-defined" problems whose solution cannot be expressed by formulae or by means of using classical or modern mathematics. In this case it is more convenient for the end user to specify their requirement to the computer by means of natural language (NL).

By an *intelligent system* we shall understand a system which enables us to solve *intelligent problems*.

2.1. Intelligent Problem

Intuitively under the problem **T** they will understand the four **<X,Q,F,Y>**, in which **X** stands for the finite set of input data and their specification; **Q** represents the goal descriptions, and **F** is the finite sequence (or *set*) of rules transforming **X** into **Y**. Thus **Y** is the finite set of output data.

Proceeding from a given definition it is easy to single out at least three classes of problems, which are characterized by the following relations:

X & Q & F |-- Y, (1)
X & Q & Y |-- F, (2)
X & Q |-- F => Y, (3)

where symbols "**&**", "**|--**" and "**=>**" stand for "**and**", "**give**" and "**implication**" respectively.

Intelligent problems are characterized by relations **(2)** and **(3)**. In this case the problem to define **software chips** can be represented by:

T = <Sp(x),Dt(x),Nm(F),As(F),Sp(F),Cnd(F),Dc(F),Pr(F),Sp(y),Dt(y)>,

where **Dt(x)** and **Dt(y)** are the *"input"* and *"output"* data, respectively;

Sp(x) and **Sp(y)** are their *"specification"*;

Nm(F) is the *"algorithm name"* coinciding with the problem name **Nm(T);**

Dc(F) represents the *"declarative description"* of the finite sequence (or set) of rules called the "**algorithm**". Having available **Dc(F)** the system **Knows How** to solve the problem **T**, but it **Cannot** (is not able to) solve this problem;

Pr(F) is the *"program representation"* of **F** called "**program**" (or "**module**"). Having available only **Pr(F)** the system **Does Not Know How** to solve the problem **T**, i.e. it cannot describe declaratively the course of its solution, but because the description of **Pr(F)** is *"intelligible"* to the system it **Can Execute** F (i.e. **Can Solve** the problem);

The description of *"functionality"* - **As(F)**, the *"condition"* of its execution - **Cnd(F)** and its *"specification"* - **Sp(F)** including the language for the description of **F**, the method of solving, the required computational resources for its implementation etc are brought to conformity with every **F**.

2.2. Content and Structure of KB

The activity of any natural (or artificial) intelligent system is just connected with solving different problems. Hence, knowledge of these systems must be **predisposed** to realize such activity. In our opinion the KB should consist of three components:

(1) Knows WHAT,
(2) Knows HOW,

(3) CAN DO Something.

According to the traditional approach to knowledge representation, the knowledge is divided into *"Declarative"* and *"Procedural"* that does not permit one to realize the main capability of the human mind: ***"bootstrapping principle"***.

One can distribute among the three part of KB the elements of problem notion:

(1) Knows WHAT: ***Sp, Dt, Nm, As, Cnd.***

Here all the declarative components connected with the specification of the problem being solved are interlinked.

(2) Knows HOW: ***Dc(F),***

(3) CAN DO Something: ***Pr(F).***

At present there can be no doubt that the possibilities of the artificial intelligence system (AIS) are defined to a large degree by the organization of the knowledge store. In the general case, under the **memory organization** one should understand the regularity of data distribution in memory assuring the storage of various links between separate elements of information and representing the main principle of gestaltpsychology, i.e. **"the whole is greater than the sum of the parts"**. In other words, the structure obtained as a result of integration should contain more information than had been used for its creation. Apparently this defines the striking ability of a human being for generating and understanding an endless number of sentences based on the limited experience with a limited number of sentences.

Moreover, at every moment of time both a man and AIS deals only with relatively small fragments of the external world. The corresponding structures are needed to integrate these fragments separated in time into the integral picture.

All kinds of binary relations can be divided into just four classes: ***"one-to-one", "one-to-many", "many-to-one"*** and ***"many-to-many"***. For implementation of these classes of binary relations the four types of elements corresponding to them are suggested: **I-elements, λ-elements, Y-elements** and **X-elements.** Different combination of these elements determine the different attributes of the structures. The KB of *DESTA*-system are provided by the interaction of the different structures as follows:

- **L-tree-structure** (combination of **I-** and **λ-elements**). This is an initial structure which provides the recognition of new words, the normalization of well-known words and the determination of direct links with the corresponding nodes of **Sm-structure, Md-structure** and **Set-structure**.

- **TB-structure** (combination of **I-, Y-** and **λ-elements**). Provides the understanding of new words.

- **Sm-structure** (combination of **I-, Y-, λ-** and **X-elements**). Provides the handling of new or well-known sentences or sequences of words.

- **Md-structure** (combination of **I-, Y-** and **λ-elements**). Provides the mapping of the **In-, Out-parameters** and **Cnd(F)** interaction for different modules and algorithms.

- **Set-structure** (combination of **I-, λ-** and **Y-elements**). Usually each module is associated with several other modules logically including it or included by it. This structure permits the system to map such links.

- **PrRI-structure** (combination of **I-** and **λ-elements**). Provides the storing of production rules and direct access to them.

3. Inference Engine

In the past AIS development was based on the *Logical Paradigm,* the main idea of which was to extract the problem solving from some theorem proof using, for example, first order predicate calculus (or Horn clause logic which is the restriction form of first order predicate calculus).

At present it is understandable that the KB of the real AIS is ***incomplete, inconsistent*** and *should be* ***open.*** In such a case it would be natural to devote more attention to the *human inference process* which is based on *"plausible reasoning"* using maximum "argumentation" about problem solving within the framework of KB.

One can single out at least three relatively independent mechanisms serving the *"natural inference"* source:

- **integration of information;**
- **addition of information;**
- **cognitive transformations.**

The idea of the *structured approach* for natural inference is considered.

4. Natural Language Interface

The natural language (NL) was chosen as:

- an *external* language for knowledge descriptions;
- an *internal* language for knowledge representation;
- a *specification* language of the problem being solved and non-procedural or procedural algorithm;
- a *communication* language between the end users and *DESTA* system.

Any NL-text handling is performed as follows:

- Any NL-sentence is divided into so-called **"nuclear"** (the simplest) sentences. By *nuclear sentence* (NS) is meant:

- a simple or a simple extended sentence with the direct order of words, where the subject group can be expressed only by a noun.

- a simple or a simple extended sentence with the direct order of words, where the subject group is expressed by a verb in the form of the imperative mood.

• Any NS is transformed into a form of the **NL-statement.** The verb of any NS is just a name of **active** or **state** statement. *Active* NS looks like a module description (e.g. ***Remove****(what,from,to)*) and *state* NS - as a specifier (e.g. ***Be****(what,where)*). Every NL-statement consists of the operation name plus respective parameters determined by the valence of this operation or its management model or its role frames.

NL-statements represents **A-statements, S-statements** or **R-statements. A-statement** is the statement of action or the active NL-statement. **S-statement** is a state NL-statement or a statement of relations with a subject of inactive state. **R-statement** represents some relations like *syno-nym, antonym, a part of* and so on (e.g. *"Delete a cursor is a synonym to erase a cursor", "Ascending sort algorithm as opposed to descending sort algorithm", "TB-structure is a part of DESTA knowledge base").*

There is a semantic equality between the initial sentence and the finite set of NL-statements.

• The finite set of NL-statements for initial NL-sentence is represented as a ***"concept".***

5. Conclusion

At present *DESTA* is software implemented to proof the correctness of a paradigm being suggested. In short the keystone of this paradigm is as follows:

• For the end user it is more convenient to specify their requirement to the computer by means of **natural language.**

• The system has to automatically yield a readily comprehensible good structured rapid prototype which *in all stages of structured growth* ***should be executable.***

• Software development is an intensely **knowledgebased activity.** Using NL-specification we can include in the KB functional descriptions of **"software chips"** (SC): *modules, procedures, functions, batch files, operating system commands* and *algorithms.* The SC can be implemented on any programming languages: *module-oriented, object-oriented* and/or *active declarative* languages. The NL-specifications allow us to join them in whole software systems (we are not discussing here about compatibility SC for different programming languages).

• Using NL-specification DESTA either extracts from the KB the suitable SC in accordance with the NL-specification or asks the user for a more detailed description of the problem being solved.