International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 16 (2017) pp. 5984-5989

Intelligent Learning Agents Construction by Context-Redefined Language Technology (Resource Consumption Behavior Prediction Tasks)

V. N. Podsvirov

Scientific and Technical Center "Technocenter" of Southern Federal University, Rostov-on-Don, Russia.

Abstract

This work deals with design and application questions of context-redefined computer lan-guages for new information technologies. Realization problems of such languages are discussed for intelligent learning agents (ILA), which applied for solving of resource consumption behavior pre-diction tasks in communal services. Additional attention for implementing functions conversion methods was charged. The same approach can be used for another task solution too. Intelligent agent has complicated actions, which is defined by environment interaction and by internal states forming. As a result there are problems of intelligent agents design: search and formation of learning algorithms for intelligent agent; variation and invasion of these algorithms into the own intelligent agent. The article deals with the second problem. The approach is in the application of context-redefined language and it support system for problem solution. We concentrate attention to princi-pal unpredicted changing of source function algorithms. The main part of the intelligent learning agent is performance element. There are six compo-nents of agent performance element [1]. All of them are discussed from the context-redefined lan-guage use point of view. We interested in the conditions and methods context forming for every component of per-formance element. And we pay extra attention to methods of constructive function interpretation, which can be varied or can be also changed. Main idea is to extract changing parts of component algorithms and organize proper interac-tion between every part and the context which can change it directly or indirectly. As a result, re-quired adaptive algorithm variation takes place on the base of obtained knowledge. One of the proper tasks is in forming this process during real time functioning of ILA.

Keywords: Context-redefined languages, intelligent agents, intelligent learning agents, computer languages

INTRODUCTION

An agent [1, 2] is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

Discuss the structure of a general learning agent (Fig.1).

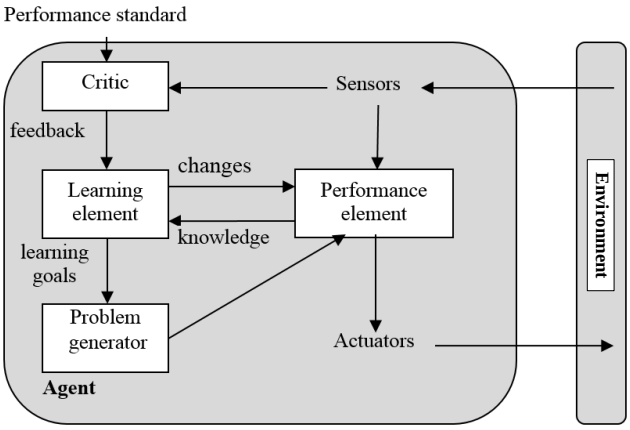

Figure 1: A general learning agent

Learning agent in general consists of:

- performance element (PE), which define action,

- learning element (LE), which modify PE to reach most applicable results.

Today we can find different types and variants of LE. Structure of LE depends on: PE design, tutorial feedbacks and styles of component definition. We suggest control system for LE, which can be useful for learning task solutions.

Discuss typical components of PE [1]:

1. A direct mapping from conditions on the current state to actions.

2. A means to infer relevant properties of the world from the percept sequence.

3. Information about the way the world evolves and about the results of possible actions the agent can take.

4. Utility information indicating the desirability of world states.

5. Action-value information indicating the desirability of actions.

6. Goals that describe classes of states whose achievement maximizes the agent's utility.

Context-redefined language (CRL) is defined as a language with the self-change properties, which can be varied by means of added context [16-21]. As a result, we watch not only the change of some sentence meanings, but new syntactical and semantic constructions are appeared.

THE MAIN PART

Design of CLR support systems demands additional efforts, because of not every traditional compiling and interpreting system can support this kind of task. Sure, the elements of CLR and the constrains of maintain system are inserted and designed into modern programming languages as a rule. But the main goal, which define it appearance, is not so complex and more simple.

In the conditions of CRL, one of the typical tasks is in suitable selection and correct interpretation of the context. Some sets of syntactic rules are used as predicate for solving result, which value is: "true" or "false". According to value: "true" starts further stage of recognition, but "false" returns process back to the next attempt of source text recognition.

According to the context agreements (common case), the beginning of the context change can be found in arbitrary positions of the string, not only in the first. The context can occupy the whole string or part of it, may spread for some strings and so on (according to recognition algorithm).

Imagine the input text in the form for the simplest illustration:

String 1

String 2

. . .

String i

. . .

String n

Assume for the simplest case (presented above), that the context may take one or two strings entirely. As a result, the analysis of the text (for main principals illustration) might be:

String 1

Context 1 (String 2)\{

String 3

String 4

\}

Context 2 (String 5, String 6)\{

String 7

\}

. . .

String n

Brackets: "\{", "\}" - define the scope of context. They are not included in the source text, but illustrate recognition system action. "String 2" is recognized as "Context 1", but strings: "String 5" and "String 6" are recognized as "Context 2". "Context 1" and "Context 2" are conditionally independent, because of their coverage do not intersect.

But it can be as:

String 1

Context 1 (String 2)\{

String 3

String 4

Context 2 (String 5, String 6)\{

String 7

\}

String 8

\}

. . .

String n

In this case, the "Context 2" can be seen as complementary to "Context 1" or alternative action depending on the context implementation algorithm. Similar to conventional procedural programming languages, it can be loops nested in each other. In the common case, CRL allows us to leave "Context 2" with the aftereffect to "Context 1": for example, to forbid execution of a region nested "Context 2" in a loop formed by "Context 1" and to offer in terms of "Context 1" to execute the aftereffect of "Context 2". This fundamentally changes the traditional approach towards generalization of it.

Illustration to the above may be the text in c - oriented language:

```
1. int j=5;                          //String 1
2. for(int i=0; i<3; i++)            //Context 1 (String 2)\{
3. {                                 // String 3
4.      j+=i;                        // String 4
5.      once                         // Context 2 (String 5, String 6)\{
6.      after
7.      init(j);                     // String 7
                                     // \}
8. }                                 // String 8
                                     // \}
. . .
n.                                   //String n
```

It is expected that after the completion of "Context 1" will be executed method "init(j)", which was generated by "Context 2". As a parameter meaning it will be passed the value of j, which was formed before the time of "Context 2" execution. Methods to ensure the aftereffect can be transferred not only for covering contexts, but also for contexts, which follow one after the other.

For instance:

```
String 1
Context 1 (String 2)\{
String 3
String 4
Context 2 (String 5, String 6)\{
        String 7
\}
String 8
Context 3 (String 9, String 10)\{
        String 11
\}
String 12
\}
. . .
String n
```

"Context 3" can access the aftereffect method from "Context 2". For "Context 3" and "Context 1", it may be different methods.

The system distinguishes the context from plain text, can contain contradictory and conflicting rules of recognition. In addition, it may change sets of the rules and the sequence of their application. As a result, the current state of the recognition system is defined by the tuple {A, R}, which contains references to proper sequence selection algorithm of the rules "A" and an array of references to recognition rules "R". Depending on the selected algorithm the final choice of a suitable recognition rules can be arbitrarily complex.

Rules of recognition in the General case can affect the overall rules selection algorithm and select the rules that can generate an infinite loop in the case of incorrect use.

It is suitable to define the first stage of recognition as predicate stack, if syntactic and recognition rules sets are given. Every element of the stack is defined as own predicate, which conditionally (data connection takes place) is not connected to a further control source up to the action end.

Stack is designed by the loading of predicates into the top. Predicates analysis goes in the invert sequence, during the source text constraints recognition. This process, in some aspects, similar to functioning of communication protocol stack, which is widely used in modern operating systems. It is possible to consider communication protocol stack functioning as simplified variant of suggested system.

Add definitions:

- beginning Point (P_b) - marks the first symbol of recognizing text.
- current Point (P_c) - marks current symbol of recognizing text.
- end Point (P_e) - marks the end symbol of recognized text.
- stop Point (P_s) - marks the last symbol of recognizing text.

Every predicate from the text achieves P_b, when starts. We assume, the text, which lies before this point, is successfully recognized and processed. In the current predicate limits P_c serves as P_b (functionally). In other words, it indicates the beginning of the current predicate constraint. If searching construct is absent, then rollback returns recognition process to P_c point. P_e marks the end symbol of performed text standard part. If predicate recognition is successful (no rollback), then P_e value will be stored into P_c with the shift to the next symbol.

Particular interest is in the point P_s, which marks the last symbol for current predicate searching and processing text limits. It is useful for recognition procedures amount minimization, because of searching and processing area cuttings. For P_s calculation, it is convenient to start specially oriented lexical analyzer, which, in common case, can be designed for every predicate type.

As instance, we can suggest loop statement from procedure oriented programming language. Specially oriented lexical analyzer highlights (from P_b) the entire block down to the end of the loop (P_s) and solves reference to the next statement (P_s+1), which lies just after the current block. If undefined alien text (outer insertion) for proper translation takes place, then the end of alter text invasion will be found.

Suitable result of certain analysis is in tags system. Each tag is connected with a private object or group of objects. Some tags can be local, accessible for current predicate only, but other - global with total access for another pretender groups.

Text object can be defined as text symbol or certain group of symbols. It is not limited by the text symbol elements sequence, but can be represented as separated symbols or separated groups of the text symbols.

In common case, a stack of tags is useful. The blocks, which are searched out from the source text, can be processed more simply. Stack convenient for own CLR debug system organization too. In this stack must be defined operations: "read", "write" and "familiarize" stack content. "Read" and "write" operations are classic for stack, but "familiarize" provides entire stack content to read (without delete and modify

functions). In addition, "familiarize" operation provides index stack reading (as "read" without delete) from top to bottom.

Thus, every predicate from the current predicate stack can act with the data, which organized as separate stack. The data designed as objects with appropriate tags, which stored to them as list elements. Data stack has layer by layer design. Every layer, at first, is stack element, but it has references to proper outer objects, at second. However, references to added proper layers tags and to generating changes predicate from predicate tag are inserted into entire objects.

According to this stack structure, generating changes predicate from the stack and recognition step, as predicate result, can be always identified, because one predicate run generates single recognition step and not more one layer in data stack. In rollback case, there is "false" as a result of predicate computation, top layer stack references are cleared adequately for correct system state recovery. In the process, generated objects in the layer are dropped, but added elements in the previously generated objects are null.

From the point of supporting system view, additional convenience of step by step operation track takes place.

Predicate stack is read, until one of predicates returns "true". After that, semantic and next analysis, which are closely connected with recently ended predicate, take place. Changes: $P_b = P_e + 1$ and further recognition is continued. This continuation is native for ordinary recognition without context changes, however, we have no limits to use another predicate stack or modify current one.

Discuss another interesting instance, the side effect of "true" result predicate can be the text, which interprets previously recognized portion. This kind of tasks are typical for computer translation systems or for proper macros extensions, when indirect recursive use takes place. In this case, it is early to go to the next source text symbol. At first: proper macro extension must be formed; at second: it must be processed; at third: if no recursive step, next source text portion recognition must be started.

In the suggested approach, this task is solved by simple use of new predicate stack, which deals with the another source text instead of previous. This text is formed by the current predicate solution. Thus, the task is converted into the simple source text stream switch with the further return to the previous one. As a result, the idea of source text stream stack comes to us. "true" predicate result can direct to source text stream stack design and source text stream switch with return. It means, that current stack (source text stream) element exhaustion makes source text stream recovery (from the stack) and previous predicate state from the predicate stack.

As the goal, we need for suggested approach realization in: some amount of predicate stacks, stack of global data for predicate functioning, source text stream stack for source text stream switch. We pay attention to: predicate stack change is practical equivalent to recognizing language switch or proper language variation use.

As a remainder, one system more must be designed to support source text point back return, because the further context can transform it value and result to rollback previous recognized part of the text as wrong.

Problem can't be solved by simple text "back step", because text interval between current symbol and "back step" result was successfully recognized, already. For proper rollback, data stack can be used, which fix all data changes. The proper Clearing of the stack layers is equivalent of global rollback.

Every stack layer forming can be defined as transaction, which will be fixed or rolled back in future. In the case, principal difference from the classic transaction system is in dynamical roll back of some transactions amount, which were fixed previously with success. In addition, these transactions can generate side effect, which is present whenever to transaction fixing or rolling back. It is reached by partial clearing of the layer or the layers.

For each layer: clearing method (rollback), fixing method, **next using method** are formed. In regular linear layer form, the fixing method is started just after data layer creation. In the layers, positioning between current and desired layer, clearing method is started for repeat recognition need and back return to the previous text stream point.

There are two variants of further processing:

- The first: method supports pre saving of cleared elements from the stack (just before using of every stack element, **next using method** is started).

- The second: clearing method destructs proper part of the stack by direct elements delete from the stack.

Obviously, semantic and other recognition analyzers have to obtain appropriate methods for the correct functioning.

For the first glance, this approach is seemed to be complex, but object-oriented programming and correct object model formation demonstrate opposite result. Mind JavaScript as paradox parallel. In it: Inserted elements can be defined as editing text with the further objects design under the text base.

Proposed approach supports not only the switch to the new formed text, but it supports changing source text with previously recognized old text rollback.

In reality, we have a chance to transform syntactic constructions in the source text by proper choice from the current predicate stack and/or add new items in it.

Naturally, discussed approach is not total for all application fields of CLR. But with the proper technical constraints, the coverage of the task fields subsequent investigations in the recognition field is considerable.

Suggested approach deals with structural insertion of learning algorithms into the feedback functions. Technically, CRL system obtain context defined learning variations and algorithms, which vary them. It means that adaptation of varying (learning) functions of intelligent agent to dynamically changing environment takes place. In principal, variation inserting process into previously defined components can be applied enormously times. Naturally, next typical insertion is defined by practicability and limits of obtained total intelligent agent algorithm complication.

There is possibility of simple context extraction from functions, which vary learning elements and generate flexible algorithm variation of these functions. Using of that possibility can be recommended not in every case, but for complicate intelligent agents is effective and useful control problems solution.

Since technical methods of algorithms insertion are suggested, but not own algorithms. All typical decisions and theoretical results of research in the field of intelligent agents can be applied as a general part, which is formed by CRL structure of agent.

This approach was applied for solving of resource consumption behavior prediction tasks in communal services. Intelligent agent was inserted into resources consumption environment.

Main tasks of intelligent agent were:

- adaptation for environment (prediction functions forming);
- predictions for resources consumption system;
- predictions for damage control system.

CONCLUSIONS

As a result, we can compare predictions and behavior of the systems. Preliminary outcome: deviation is <5% for the model. Result can be improved in future.

The research results outlined in this paper were obtained with financial support from Ministry of Education and Science of the Russian Federation, as part of the execution of the project entitled " Establishment and creation of high-tech production for manufacturing of innovation system of complex account, registration and analysis of energy and water consumption by industrial enterprises and objects of housing and communal services", pursuant to decree of the government of the Russian Federation № 218 issued on April 09, 2010. Research was conducted in FSAEI of HE of the SFU.

REFERENCES

[1] Russell, S., Norvig, P. 2010. Artificial Intelligence. A modern Approach.

[2] Podsvirov, V.N. 2016. Context-Redefined Language Application for the Tasks of Intelligent Learning Agents (Resource Consumption Behavior Prediction Tasks). International Journal of Applied Engineering Research. Vol. 11, No. 15, pp. 8471-8484

[3] Jennings, N. & Wooldridge, M. 1998. Applications of Agent Technology. In N. R. Jennings and M. Wooldridge, editors, Agent Technology: Foundations, Applications, and Markets. Springer-Verlag.

[4] Agent-Oriented Methodologies. 2005. Ed.by B.Henderson-Sellers and P. Giorgini. Idea Group Publishing.

[5] Subrahmanian, V.S., Bonatti, P., Dix, J. et al. 2000. Heterogeneous Agent Systéms. Cambridge MA: The MIT Press.

[6] Bauer, B. 2001. UML Class Diagrams: Revisited in the Context of Agent-Based Systems: Agent-Oriented Software Engineering (AOSE). Montreal.

[7] Bauer, B., Miiller, J. P. & Odell, J. 2001. Agent UML: A Formalism for Specifying Multiagent Interaction. Berlin: Springer-Verlag, pp.91-103.

[8] Wooldridge, M. Jennings, N. & Kinny, D. 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers. Manufactured. Netherlands. 3, 285312.

[9] Xu, D. & Deng, Y. 2000. Modeling Mobile Agent Systems with High-Level Petri Nets. IEEE Conference on Systems, Man, and Cybernetics. Nashville, pp. 3177-3182.

[10] Deloach, S. A. 2001. Multiagent systems engineering. / Wood, M. F. Sparkman, C. H. International Jouranl of Software Engineering and Knowledge Engineering. Vol. 11, No. 3, pp 231-258.

[11] Eiter, T. 2002. Comparing environments for developing software agents. Mascardi, V. AI communications, Netherlands. Vol. 15, pp. 169-198.

[12] Franciso, M. 2005. Integrating Multi-Agent Systems: A Case Study. Raymond, S. Staron, F. IFIP International Federation for information Processing, Springer Verlag. 2005. Vol. 159, pp. 99-108.

[13] Glaser, N. 1997. The CoMoMAS methodology and environment for multi-agent system development. In Zhang, C. Lukose, D. (eds). Multi-Agent Systems - Methodologies and Applications, Springer-Verlag: Berlin. LNAI 1286, pp. 1-16.

[14] Lange, D.B. & Mitsuru, O. 1998. Programming and Deploying Java (TM) Mobile Agents with Aglets (TM). Addison-Wesley.

[15] Barbosa, Fernanda, Cunha, Jos. 2000. A coordination

language for collective agent based systems: GroupLog. ACM Symposium on Applied computing SAC'2000 Villa Olmo, Como, Italy, pp. 189-195.

[16] Podsvirov, V.N. 2016. Context-redefined languages application for intelligent learning agent teaching. Actual problems of modern science: Scientific-practical international conference. Stavropol, pp. 229-232.

[17] Podsvirov, V.N. 2015. Context-redefined languages and intelligent agents. Actual problems of modern science: Scientific-practical international conference. Alushta, pp. 192-194.

[18] Podsvirov, V.N. 2014. Context-redefined languages and productions. ICT in the science, manufacturing and education: Six scientific-practical international conference. Stavropol, pp. 391-393.

[19] Podsvirov, V.N. 2013. Productions and context-redefined languages. Actual problems of modern science: Second scientific-practical international conference. Stavropol, pp. 190-192.

[20] Podsvirov, V.N. 2012. Context-redefined languages implementation, one approach. ICT for science, manufacturing and education: Fifth scientific-technical international conference. Stavropol, pp 54 – 57.

[21] Podsvirov, V.N. 2011. Context-redefined part of modern logical programming languages. Actual problems and innovations in the economy, governance, education, information technology. Science international conference. Stavropol, pp. 150 – 152.