Extension of Agent-oriented Domain-specific language ALAS as a support to Distributed Non-Axiomatic Reasoning

Dejan Sredojević*, Milan Vidaković**, Mirjana Ivanović***, Dejan Mitrović***

* Novi Sad Business School, Novi Sad, Serbia
** University of Novi Sad/Faculty of Technical Sciences, Novi Sad, Serbia
*** University of Novi Sad/Faculty of Sciences, Novi Sad, Serbia
drsredojevic.vps@gmail.com, minja@uns.ac.rs, {dejan, mira}@dmi.uns.ac.rs

Abstract—Development of Agent-based languages is the natural extension of the research in the area of Agent-based systems. This paper deals with adding the support for the Distributed Non-Axiomatic Reasoning into the ALAS agent-oriented language. This support has been added into the Siebog agent middleware. Siebog is a distributed multiagent system based on the modern web and enterprise standards. Siebog has built in support for reasoning based on the Distributed Non-Axiomatic Reasoning System (DNARS). DNARS is a reasoning system based on non-axiomatic logic (NAL) and general principles of development of non-axiomatic reasoning systems. So far, the DNARS-enabled agents could be written only in Java programming language. To solve the problem of interoperability within different Siebog platforms, an agent-oriented domain-specific language (AODSL) ALAS has been developed. The main purpose of the ALAS is to support agent mobility and implementation and execution of agents on heterogenous platforms. This paper describes the extended version of the ALAS language which supports DNARS. The conversion process of ALAS code to Java code is also described in this paper. The latest version of ALAS has been developed by using textX framework and Arpeggio parser.

I. INTRODUCTION

This paper deals with the improvement of the support for the Distributed Non-Axiomatic Reasoning System (DNARS [1]) by extending the ALAS language [2][3][4], to support the DNARS. The ALAS language is the agent-oriented language built for the Siebog agent middleware [5][6][7]. Siebog middleware is an agent middleware which supports both server-side [5][7], (application server-based) and client-side [8], (browser-based) agents.

Distributed Non-Axiomatic Reasoning System (DNARS) is a novel architecture for reasoning which can be employed for the intelligent agent development. It extends the Non-Axiomatic Logic Reasoning [9][10][11], by introducing the capability for distributed processing which allows large amounts of data to be processed. The main advantage of DNARS, when compared to all other existing reasoning and cognitive architectures, is that it leverages state-of-the-art techniques for large-scale, distributed data management and processing. This approach allows DNARS to operate on top of very large knowledge bases, while serving large numbers of external clients with real-time responsiveness.

Instead of the popular Belief-Desire-Intention (BDI) model for the development of intelligent agents, DNARS is based on the so-called non-axiomatic logic, formalism developed in the domain of artificial general intelligence [11]. The term of non-axiomatic means that the logic is suitable for the development of systems that operate in conditions of insufficient knowledge and resources. For example, it can effectively manage to inconsistent knowledge bases and summarize the existing knowledge and thus reduce the total number of statements in the database.

Within Siebog multiagent platform, there was a problem of interoperability (movement of the agents between servers and clients that can be implemented in different programming language) [7]. This problem has been solved by developing the ALAS programming language, which has been described in our previous papers [2][4][12]. By upgrading of the Siebog Middleware and by introduction of new DNARS architecture for agents management, there was a need to upgrade the ALAS to support this reasoning framework.

For the development of AODSLs, developers use a variety of tools. One of the most used tools in recent years that offers great possibilities in the development of DSL is Xtext framework for the development of programming languages and DSL [13]. Initially, the ALAS language has been developed using Xtext framework [4]. However, the Xtext is a complex environment and it is often necessary to know in detail how it works to develop of some advanced language features. Therefore, we have switched to use the textX framework [14][15], and Arpeggio parser [16][17]. In this paper we will present results obtained with this change.

II. DNARS IMPLEMENTATION

Non-axiomatic logic is a formalism for the specification of the system for reasoning within artificial general intelligence [9][10][11]. NAL includes grammar i.e. alphabet, a set of rules for the execution, and semantic theory. Unlike many other formalisms used in computing, NAL is a logic of terms [18][19][11]: the statements are given in the form of subject-copula-predicate, where subject and predicate are terms. The term of non-axiomatic means that the logic is suitable for the development of systems that operate in conditions of insufficient knowledge and resources [10][11]. It means that the knowledge inside the system could be incomplete.
and possibly not consistent. New proofs can appear at any time, can have any content, and can change the truth of any existing statement. Additionally, the system may not have sufficient resources (in terms of time, storage space, etc.) to consult its entire knowledge to solve the problem. Therefore, it is possible that the system does not apply the full set of rules for executing logic. The whole logic is organized into nine levels [9]:

- NAL-1: Inference rules for heritage.
- NAL-2: Similarity, instance, and property copulas.
- NAL-3: Compound terms.
- NAL-4: Arbitrary relations among terms.
- NAL-5: Higher-order statements.
- NAL-6: Variables.
- NAL-7: The concept of time.
- NAL-8: Support for operations, i.e., procedural statements.

Each level introduces additional grammar and execution rules, and extends expressivity of logic and the ability of the system that are based on it. The current version of Siebog implements first four levels of NAL within DNARS [1].

DNARS is the reasoning system that is based on the NAL and the general principles of the development of non-axiomatic reasoning in combination with contemporary approaches and standards for the processing of large amounts of data [1]. DNARS distributes reasoning among cluster of computers in order to improve performance. It incorporates an efficient knowledge management system, and a set of inference engines for answering questions and deriving new knowledge. DNARS also uses a graph-based representation of knowledge [20][21].

When defining the architecture of DNARS, a special attention is dedicated to the organization of the knowledge base. The base is designed as a distributed, scalable architecture which consists of three levels. At the lowest level, all knowledge is fragmented and distributed to multiple computers in a cluster. So-called horizontal scaling is used: to increase the amount of knowledge, the system performance is maintained by simply adding new computers to the cluster [22]. This approach has two main advantages:

- The knowledge base can contain and manage of large amounts of data. By introducing appropriate rules for the data distribution, fast finding and obtaining the relevant statements is possible.
- Fault-tolerance is enabled because the data is copied to the computer cluster, i.e., there is no single point of failure, and the system can continue to operate regardless of the hardware or software faults.

The entire knowledge is then divided into one or more knowledge domains. Domains organize data from a lower level into logical categories. During runtime, the system may consult one or more domains. This organization also supports sharing data between clients. The knowledge that belongs to one client can be placed in one domain. However, more clients may work with the same domain (one or more), and thus can share knowledge and acquired experience.

Finally, the short-term memory is located on the third, the highest level of abstraction. It contains knowledge that is directly available to rules for executing and it is the starting point for reasoning in DNARS. The main task of short-term memory is to serve as a module for runtime optimization. Its content should be fully fit in the working memory of one machine, providing maximum performance [1].

The knowledge bases based on NAL statements can actually be represented using of the so-called property graph. Property graph is a directed, multi-relational graph with any number of properties attached to vertices and edges [20][21]. That is, it is a graph which can include different types of edges (e.g. for representing inheritance and similarity), and in which each vertex or an edge can have any number of key → value pairs attached to it.

III. SIEBOG AGENT MIDDLEWARE

Siebog multiagent middleware combines modern principles of server and client development system into a single software framework for agents. On the server-side [5][7], Siebog offers load-balancing agents per cluster nodes, as well as resistance to hardware and software faults (fault-tolerance). On the client-side [8], Siebog functions as a platform-independent system that is designed so that it can be executed on a number of devices, such classical desktop computers, smartphones and tablets, smart TVs, etc. Server and client side are integrated into a single framework that delivers cross-platform agents communication, heterogeneous mobility, as well as code sharing. Finally, Siebog is connected to a DNARS, in order to enable the development of intelligent multiagent systems with unique capabilities.

Siebog multiagent platform has been extended to include support for reasoning based on DNARS [1]. Since annotations are the standard meta-programming constructs of Java EE, the same approach can be applied for defining server-side DNARS agents. A set of annotations for marking of agent goals are offered to software developers. Annotation based development of intelligent agents has also been used elsewhere. Currently, Siebog agents that rely on DNARS reasoning are programmed in terms of beliefs and actions. The three main annotations for belief and action management are @Beliefs, @BeliefAdd, and @BeliefUpdated. In the background the system then automatically translates the signals sent from DNARS’s Event manager, in calls of appropriate methods. The integration of client-side Siebog with DNARS was implemented the same way as the integration with the server-side of the Siebog, i.e. through the communication with the appropriate web services. In terms of support for intelligent agents, Siebog left the traditional BDI architecture. Instead, it offered the possibility of development of intelligent multiagent systems which involve agents with innovative reasoning skills.

In this way, Siebog and DNARS represent a unique multiagent framework for the development and execution of intelligent agents in Web environments, and offer new and interesting ways for the practical application of agent technology.

Agent-oriented Domain-specific language ALAS is extended with new language constructs to support new
distributed system for non-axiomatic reasoning. The original purpose of ALAS constructs was to hide much of the complexity of the agent code, but, on the other hand to be sufficient to show how agent works. In this way, the developer can focus on solving a specific problem [4]. The goal of the language is to allow mobility of agents i.e. moving and executing agents on different platforms and different virtual machines. The idea is that all agents (client-side and server-side) can be created in the same programming language and then be transformed into the appropriate language of the target platform, during the migration. This paper describes one example of transformation of the ALAS code to the Java code (Listing 2) because server-side of the Siebog is implemented in Java. The process of upgrading the ALAS to support client-side (JavaScript) is in progress, too. Reference [23] describes previous version of ALAS which was implemented in Xtext and the process of conversion of the ALAS code to the JavaScript using Google Web Toolkit (GWT).

The latest version of ALAS was developed with the support of the textX framework [15]. TextX is a meta-language and a tool for building Domain-Specific Languages in Python. It is built on top of the Arpeggio PEG parser (Parsing Expression Grammar) [17], and takes away the burden of converting parse trees to abstract representations from language designers. From a single grammar description, textX constructs Arpeggio parser and a meta-model in run-time. The meta-model contains all the information about the language and a set of Python classes inferred from grammar rules. The parser will parse programs/models written in the new language and construct Python object graph a.k.a. the model conforming to the meta-model. Notable features of textX are: (1) construction of the meta-model and parser from a single description, (2) automatic resolving of model references, (3) meta-model and model visualization (using GraphViz dot tool3), (4) repetition expression modifiers, (5) grammar rule modifiers, (6) case sensitive/insensitive parsing (configurable per meta-model), (7) whitespace handling control (per meta-model and per grammar rule), (8) direct support for language code comments (comments are treated as whitespaces), (9) object and model post-processing, (10) grammar modularization, (11) extensive error reporting and debugging support.

The syntax of the language is defined using textX’s grammar that consists of a set of rules [14]. Since textX is based on the Arpeggio parser, PEG notation for describing of grammar is at the base of the ALAS grammar. The main advantage of the PEG grammars compared to some other grammars, for example, context-free grammars (CFG) is the use of ordered choice operator that enables uniformity of parsing. If the input text belongs to a language that describes the given PEG then there is only one valid tree that describes it, i.e. grammar can’t be ambiguous. Extension of the language grammar is .tx. There are several .tx files that together form the grammar of ALAS language.

Besides the modification of existing properties and rules to accommodate the textX, in the language grammar are added both new rules that allow agents to support DNARS and rules to program the flow control. Two basic rules that are used to support DNARS are DnarsBeliefsAnnotation and DnarsBeliefAUAnnotation. Listing 1 shows a part of the definition of these rules.

With these rules the appropriate ALAS constructs are defined: beliefs, beliefadd i beliefupdated. In the Java code, these constructs will be converted into the following annotations: @Beliefs, @BeliefAdd i @BeliefUpdated as you can see in Listing 2.

Language model (agent), like language grammar, can be written in a plain text editor without dependence on any development environment. By using the appropriate commands, the syntax can be checked from the command prompt (shell) to see if the model corresponds the grammar and if there are syntax errors in the written agent. Besides the command prompt (shell), the execution can be achieved by writing and by running the appropriate Python script.

During the execution of the appropriate command, it creates a parser, parse tree and meta-model of language. Parser checks the grammar validity and compatibility model (agent) with meta-model (grammar). If the model corresponds to the meta-model, the console will print messages that model and meta-model are OK; if there is an error, the console will print the error and the exact location of the error.

After that, by typing automatically proposed commands, it generates both concrete syntax tree and abstract syntax tree. After these commands the corresponding directory pictures that represent concrete and abstract syntax tree will be generated. Creating the appropriate diagrams is supported by the GraphViz library [24].

One of the main reasons for using the textX is the PEG parser on which the textX is based. By using this parser, it is easy to access any segment of the language model. It is very useful if it is necessary, for example, to transform a text model (ALAS code) to some other programming

```plaintext
Listing 1. A part of ALAS grammar

DnarsBeliefAUAnnotation: annotation = BeliefAU '('
  (body=Body | body=GraphInclude)*
')';
DnarsBeliefsAnnotation:
  beliefs'(' Judgement ',' Judgement')';
BeliefAU:      ann_name = 'belieffadd' |
                ann_name = 'belieffupdated' belief = BeliefAUDeclaration;
BeliefAUDeclaration: '('('belief=Judgement')')'?
  func_param=ID ')'?
Judgement:    term = Term copula = Copula
  term = Term truth = Truth ? ;
Term:         term = AtomicTerm | term = CompoundTerm;
Copula:       copula = Inherit | copula = Similar;
Truth:        '( number = NUMBER ; number = NUMBER )';
AtomicTerm:   ID;
CompoundTerm:
  PrefixCompTerm | InfixCompTerm | ImgCompTerm;
PrefixCompTerm: '('(' term=AtomicTerm ct=AtomTerm ')'?
InfixCompTerm: '>(' term=AtomicTerm ct=CompoundTerm ')'?
ImgCompTerm:  ' image = Image term = AtomicTerm
  Option1 | Option2 ');';
CompTerm:     connector=Connector term=AtomicTerm;
```

370
language. Since the ALAS is just created in order to solve the problem of interoperability, ie. the possibility of executing agents on different platforms implemented in different programming languages, the textX is a very good choice.

During grammar parsing, textX dynamically creates a set of Python classes inferred from grammar rules [14]. Due to this fact, for the transformation of the ALAS code into the wanted programming language, the Python language is used. In order to generate the appropriate code to be maximally facilitated, textX uses the Jinja2 - The Python Template Engine [25]. However, the textX does not depend on Jinja2 Template Engine-a and for the code generation other tools and templates can be used. One of the advantages of Jinja2 Template Engine is the full adaptability to the Python programming language and because of that we have used this template engine. A template engine for conversion ALAS code in Java is implemented and a part of the code is shown in Listing 2.

**CONCLUSIONS AND FUTURE WORK**

This paper describes extension of the ALAS programming language to support the DNARS reasoning system. DNARS is built into the Siebog agent middleware, which supports execution of intelligent software agents. The job of this multiagent middleware is to provide, among other things, efficient and reliable communication channels, a yellow-pages service, mobility, and execution in distributed environments.

Siebog supports DNARS-based intelligent agents. DNARS is based on the NAL formalism for knowledge management. The final result of adding DNARS to the Siebog is a multiagent middleware with a unique architecture and a unique reasoning system for intelligent agents. So far, DNARS supports 4 levels of NAL.

ALAS was created to resolve the problem of agents mobility and their execution on both client-side and server-side of Siebog. DNARS-related extension of the ALAS language include constructs which enable developer to add and query beliefs. Current version of DNARS-enhanced ALAS is transformed to the Java programming language only. Future work will include the full support of the DNARS client-side (JavaScript-based) agents in both ALAS and Siebog. Future work will also include the upgrade of both Siebog and ALAS to support the remaining layers of NAL in DNARS.

---

**ACKNOWLEDGMENT**

Results presented in this paper are part of the research conducted within the Grant No. III-47003, Ministry of Education, Science and Technological Development of the Republic of Serbia.

**REFERENCES**


[12] D. Sredojević, M. Vidaković and D. Okanović, „Integration of agent domain-specific language ALAS into extensible Java-based framework XJAF“, 34th International Conference on
Organizational Science Development internationalization and cooperation, Portorož, pp. 1037-1043, 2015.


