A find out about of a dialogue and making a Logtalk for the sketch of an Object Oriented Logic Programming Language

By Dr. Paulo Jorge Lopes de Moura, Assistant Professor of the Department of Informatics of the University of Beira Interior

Dr. Abel João Padrão Gomes, Assistant Professor of the Department of Informatics of the University of Beira Interior

Abstract

This thesis describes the design, implementation, and documentation of Logtalk, an object-oriented common sense programming language. Logtalk is designed as an extension to the Prolog common sense programming language offering encapsulation aspects primarily based on object-oriented concepts. Logtalk primary points encompass help for each prototypes and training in the equal application, integration of event-driven programming with object-oriented programming, and category-based code reusing.

Keywords: Logtalk, Prolog, Logic programming, Object-oriented programming, Event-driven programming

Introduction

Regarding good judgment programming, Prolog, its most full-size language, born in 1972, nevertheless lacks a fashionable library and a function set appropriate for programming in the large. The first Prolog ISO standard, regarding the core factors of the language, used to be published in 1995, and is being adopted at a gradual tempo with the aid of the Prolog community. The 2nd ISO standard, regarding the module system, was once posted in 2000 and is being essentially left out via most of the Prolog community. Long gone are the days of the Japanese fifth Generation Computer System Project, based on the guarantees of common-sense programming as the silver bullet. Today, Prolog is an area of interest language. Logic used to be dropped as a programming paradigm from the first draft of the 2001 ACM/IEEE Computer Science Curricula, and used to be solely re-

introduced in the curricula closing document after a great deal stress from the common-sense programming community.

There are, of course, many motives for the present-day country of good judgment programming in general, and Prolog in particular. Nevertheless, sociologic and political motives aside, Prolog technical handicaps, such as the inexistence of fashionable libraries, widespread overseas language interfaces, and effective encapsulation mechanisms, make it an uphill struggle to use the language and the common-sense programming paradigm for teaching, researching, or industrial software program development. Regarding object-oriented programming, the subject is strongly biased toward class-based structures and the specific implementation of object-oriented standards in a few languages, notably, C++ and Java. These languages, like any different language, signify a particular set of plan selections on how to enforce vast object-oriented concepts.

Integration of common sense and object-oriented programming

Logtalk pursuits to carry collectively the principal blessings of these two programming paradigms. On one hand, objects enable us to work with the identical set of ideas in the successive phases of utility development. On the different hand, good judgment programming lets in us to represent, in a declarative way, our know-how of every object. Altogether, objects and declarative programming permit us to shorten the distance between an utility and its hassle domain. Adding objects to Prolog approves us to follow high-level object-oriented improvement methodologies and metrics to common sense programming. Objects additionally furnish common sense programming languages, and Prolog, specifically, with various points wished in large-scale software program projects. In particular, objects add namespaces to the standard Prolog flat database, offering predicate encapsulation and information hiding, beautify code reusing trough inheritance and composition, and, coupled with protocols, supply separation between interface and implementation.

Support for each prototype-based and class-based systems

Almost all object-oriented languages reachable today, are both class-based or prototype-based with a sturdy predominance of class-based languages. In particular, all modern-day mainstream object-oriented languages are class-based. However, prototypes furnish an awful lot higher alternative for Prolog modules than classes. A prototype can be a stand-alone object, no longer connected to any hierarchy, and consequently a handy answer to encapsulate code that will be reused with the aid of a number of unrelated objects. Prototypes are for this reason a herbal improve to the use of modules in functions the place the ideas of instantiation and specialization of class-based languages do no longer make sense. For different applications, the types of code reuse underlying the ideas of category and instance, are the exceptional solution. Each variety of system, with its strengths and drawbacks, is equally beneficial in the context of an object-oriented common sense programming language. As such, Logtalk targets to furnish equal aid for lessons and prototypes, inclusive of runtime guide for each prototype and classification hierarchies in the identical application. In fact, many Logtalk examples and purposes make use of prototypes, classes, and situations simultaneously.

Separation between interface and implementation

This is a predicted function of any contemporary high-level programming language. Logtalk need to furnish help for keeping apart interface from implementation in a bendy way, enabling an interface to be applied by using more than one objects, and an object to put into effect more than one interfaces. Surprisingly, this is no longer viable in the modern ISO trendy for Prolog modules, the place a module consists of a single module interface and zero or greater corresponding module bodies.

Private, protected, and public object predicates

Logtalk ought to guide statistics hiding via imposing private, protected, and public object predicates, and via adopting scope regulations frequent to different object-oriented languages: personal predicates can solely be known as from the container object, covered predicates can solely be known as from the container object and its descendants, and public

predicates can be referred to as from any object. Note that the present-day ISO well known for Prolog modules does now not aid facts hiding. By the use of specific module qualification, we can name any module predicate as lengthy as we be aware of its name.

Private, protected, and public inheritance

This is a frequent characteristic of present-day object-oriented languages such as C++ and Java, and a herbal extension of the predicate scope rules. Logtalk ought to guide a generalized implementation of private, protected, and public inheritance, enabling us to hinder the scope of inherited, implemented, or imported predicates.

Compatibility with most Prolog compilers and the ISO standard

Logtalk must be designed to be well matched with most Prolog compilers and, specifically, with the ISO Prolog standard. It ought to run on most laptop structures assisting a present-day Prolog compiler. The language sketch has to decrease implementation-dependent points to make certain vast portability of Logtalk packages across Prolog compilers and working systems. In fact, Logtalk need to be considered as a perfect device for writing transportable programs: any operating-system based code can be encapsulated inner objects imposing in reality described cross-platform protocols.

Logtalk protocol concept

Logtalk protocols encapsulate predicate declarations. Ideally, the declarations ought to correspond to a functionally cohesive set of predicates. Protocols allow the separation between interface and implementation: a protocol can be applied with the aid of numerous objects, and an object can put in force various protocols. Note that, even though summary lessons and a couple of inheritance, each aspect supported by way of Logtalk, can grant some of the performance of protocols, this answer can solely be utilized to class-based hierarchies. In contrast, Logtalk protocols can be applied with the aid of each lessons and prototypes. An object implements a protocol through offering definitions for the declared predicates.

However, the implementation of a protocol by using an object must be interpreted in a free sense. An object is no longer required to grant a definition for each and every protocol declared predicate. As mentioned in a message to an object is legitimate if the corresponding predicate is declared for the object and in the scope of the sender. If no predicate definition is determined to reply the message, it actually fails. This semantics can be considered as a Logtalk reinterpretation of the Prolog closed-world assumption.

Documenting language

Automatic era of documenting archives implies guide each at the language stage and at the compiler level. At the language level, we have to be in a position to signify arbitrary data about an entity and its predicates. At the compiler level, all applicable data need to be parsed, formatted, and written out to a documentation file.

Two frequent examples of automated documenting equipment are Javadoc and Lpdoc. The first is a well-known device protected in the Java SDK. The 2d is a device for documenting (C)LP systems, along with Prolog programs. Javadoc makes use of specifically formatted application feedback to permit the person to specify documenting information. This ability that two languages will be used when writing programs: one language for code and every other language for documentation. This is an method shared with the aid of most literate programming equipment such as CWEB and its offspring's. Lpdoc defines a state-of-the-art announcement language that can be considered as an extension of Prolog, made of a set of directives. In each case, we stop up the usage of two specific tools: a language compiler and a separate device for extracting software documentation.

In Logtalk, an essential format choice was once to use the identical language for each code and documentation. After all, Logtalk is a declarative language. All documenting data in Logtalk supply documents is expressed the usage of Logtalk directives.

Discussion and Conclusions

This affords the most applicable contributions of this thesis, some supplementary concerns on Logtalk guide for reflection and on the usage of Logtalk in the classroom, as nicely as the deliberate future improvement of the Logtalk language. Complete and greater designated conclusions about every language characteristic of are supplied.

Logtalk can be described as a multi-paradigm language that helps common sense programming, object-oriented programming, and event-driven programming. However, Logtalk purpose used to be now not to solely guide these programming paradigms however to combine them. The integration used to be made by, first, reinterpreting object ideas in the context of common-sense programming and, second, with the aid of reinterpreting match ideas in the context of object-oriented programming.

This starts through evaluating Logtalk as a Prolog object-oriented extension and as an interpreted, interactive object-oriented programming language. Secondly, the relevance of event-driven programming in the context of object-oriented languages is described. Thirdly, category-based composition, and its guide for component-based programming, is summarized. Fourthly, Logtalk native guide for reflection is examined. Then, the Logtalk assist for computerized software documentation is presented. Next, the trip of the usage of Logtalk in the school room is described, observed by means of some facts on the Logtalk distribution numbers. Finally, the roadmap for future improvement is presented.

Logtalk as a Prolog object-oriented extension

Logtalk reinterprets the idea of object as a set of predicate directives (declarations) and clauses (definitions). Thus, message sending is reinterpreted as proof development the usage of the predicates described for the receiving object. Inheritance mechanisms enable us to outline the whole database of an object. A technique is then virtually the predicate definition chosen from an object whole database in order to answer a message. By reinterpreting the standards of object, message, and approach in good judgment programming terms, a easy mapping is set up between Logtalk semantics and the acquainted Prolog semantics.

In widely wide-spread terms, this reinterpretation of object principles is shared by means of most Prolog object-oriented extensions. To consider and evaluate Logtalk with different object-oriented extensions and Prolog module systems, the following standards will be used: compatibility with Prolog compilers, language syntax, interpretation of the thought of object, function set, and working environment.

Logtalk compatibility

Logtalk is the solely Prolog object-oriented extension handy nowadays that has been de-signed from scratch for compatibility with most compilers and with the ISO Prolog standard. This graph purpose units it aside from different Prolog extensions. The preprocessor answer adopted for the Logtalk implementation lets in it to run on most computer systems and running structures for which, a contemporary Prolog compiler is available. Specifically, the modern-day Logtalk model is well matched with thirty-one variations of twenty Prolog compilers.

Logtalk syntax

Logtalk uses, each time possible, general Prolog syntax, and defines dependent language constructs, in line with the cutting-edge exercise and expectations of Prolog programmers. This helps to easy the mastering curve for Prolog programmers. This is greater than a syntactic sugar issue. For example, Logtalk allows current Prolog code to be encapsulated in objects besides any changes. Only when a predicate wants to name different object predicates, would minimal modifications be required. Thus, handy conversion of ancient Prolog packages is ensured.

The position of objects in common sense programming

The foremost reason of objects in Logtalk is the encapsulation and reusing of code, consequently decoupling this performance from the theoretical troubles of dynamic kingdom alternate in common sense programming. As such, Logtalk offers a realistic view, as a substitute than a theoretical view, of the function of objects in common sense

programming in general, and in Prolog in particular. By focusing on the encapsulation and code reuse proprieties of objects, Logtalk targets to be a fine device for fixing software program engineering issues in Prolog programming.

Implementation options for object-oriented concepts

Logtalk indicates how to enforce the foremost object-oriented standards in Prolog. These consist of principles no longer located in my view on most Prolog object-oriented extensions such as: guide for each training and prototypes; metaclasses; protocols and protocol hierarchies; public, protected, and non-public predicates; and public, protected, and personal inheritance. Thus, Logtalk is probably one of the most entire Prolog object-oriented extension on hand today. In addition, Logtalk suggests how to put into effect different essential ideas that are now not accessible on different object-oriented languages, such as classes and event-driven programming. Unlike different object-oriented extensions that are both proprietary or rely closely on the specifics of the native module systems, Logtalk implementation options are wholly well matched with any compiler that complies with the Part I of the ISO Prolog standard.

Working surroundings and different realistic matters

Logtalk working surroundings is restrained to the subset of frequent aspects of the compatible Prolog compilers. For example, there is no frequent set of predicates for working gadget get right of entry to (so limiting the performance of the Logtalk compiler) or a frequent standard for developing graphical person interfaces. Proprietary object-oriented extensions, developed to work with a single Prolog compiler, are in a position to grant a richer working environment, taking benefit of special elements of a compiler and its internet hosting operating system. Nevertheless, inside its compatibility restrictions, Logtalk tries to grant a getting to know and working surroundings comparable to different Prolog compilers and object-oriented extensions that use text-based improvement tools. For enhancing supply files, the present day Logtalk distribution consists of syntax-coloring configuration archives for famous textual content editors, alongside with textual content

templates for defining new entities and entity predicates. This improves the programming experience, in particular by using supporting to avoid syntax blunders when writing entity and predicate directives. Also covered are a quantity of programming examples and widespread documentation, which involves a person manual, a reference manual, and programming tutorials.

Logtalk as an object-oriented programming language

Logtalk extends Prolog, in the equal way as CLOS extends LISP or Objective-C extends C. As a programming language in its very own right, Logtalk shares points with frequent object-oriented languages. However, there are additionally some necessary variations due to the fact of its Prolog roots. The most massive one is that Logtalk eliminates some dichotomies deeply mounted on most object-oriented languages. These dichotomies are frequently used as a way of characterizing and classifying object-oriented languages. Specifically, Logtalk makes no difference between variables and methods, lets in most language factors to be both dynamic or static, and integrates instructions and prototypes in the equal language. Despite the first two factors are frequent to some Prolog object-oriented extensions, they are well worth summarizing here. Unlike Logtalk, nearly all object-oriented languages are strictly described as both class-based or prototype-based languages. Most of them are characterized with the aid of a clear difference between variables and methods, what is static and what is dynamic, and what ought to be finished at assemble time or can be carried out at runtime.

Predicates as each variables and methods

Logtalk object predicates unify the principles of object techniques and object variables, consequently simplifying the language semantics. Predicates cast off the dichotomy between kingdom and behavior: a predicate sincerely states what is genuine about an object. Predicates can also be used to put in force each variables and methods, however such a difference is usually optional. It follows that we no longer want separate definition, inheritance, and scope guidelines for kingdom and behavior. In addition, each

nation and conduct can be without problems shared alongside inheritance hyperlinks or described locally. This permits us, for example, to outline techniques in cases and to share object country effortlessly amongst descendant objects barring the want of first formalizing principles such as shared occasion variables.

Static and dynamic language elements

Logtalk objects, protocols, categories, and predicates can be both dynamic or static. Predicates may additionally be asserted into, and abolished from each static and dynamic object at runtime. Objects, protocols, and classes can be described both in a supply file or created dynamically at runtime. If contained in a supply file, they can be described as both static or dynamic entities. As such, we are now not constrained, for example, to outline training as static entities and situations as runtime-only objects. We might also outline an occasion in a supply file in the identical way as a classification may additionally be defined. This is constant with Logtalk essential view of objects as encapsulation units.

Event-driven programming

Logtalk provides a conceptual integration of event-driven programming into the object-oriented programming paradigm. The key for this integration is the interpretation of message sending as the solely match that takes place in an object-oriented program. Thus, Logtalk reinterprets the principles of event, monitor, match notification, and match handler in phrases of objects, messages, and methods. This approves us to continue to be inside the object-oriented programming paradigm when writing event-driven programming code.

Two essential consequences emerge from Logtalk programming exercise in the usage of activities to put into effect complicated dependency members of the family between objects. These effects are now not particular to Logtalk. Instead, they follow to most object-oriented programming languages. First, event-driven programming is an necessary characteristic of object-oriented languages for accomplishing a excessive degree of object brotherly love and averting useless object coupling on purposes the place object members of the family mean constraints on the kingdom of taking part objects.

Dependency mechanisms, as located in Smalltalk and in different languages, grant solely a partial solution, which can solely be used when object strategies include calls to the dependency mechanism methods. Second, activities and video display units have to be supported as language primitives, built-in with the message sending mechanisms. This is an vital requirement from a overall performance factor of view, which precludes the implementation of event-driven programming at the utility layer or thru language libraries. Native language aid is essential in making event-driven programming an high-quality device for hassle solving.

Category-based composition

Categories are the groundwork of component-based programming in Logtalk. Categories pro-vide finer, functionally-cohesive gadgets of code that can be imported through any object. Thus, classes play a position dual to that one performed by means of protocols for interface encapsulation. In addition, classes supply various improvement advantages such as incremental compilation, updating an object through updating its imported categories, and refactoring of complicated objects into extra manageable and reusable parts. Category-based composition, inheritance, and occasion variable-based composition grant complementary types of code reuse. Categories put into effect a composition mechanism the place a class interface will become section of the interface of an object importing it. This is in contrast to occasion variable-based composition, however comparable to what takes place with inheritance. Logtalk aid for public, protected, and personal class importation presents similarly flexibility. By making use of the ideas of separation of issues frequent to component-based programming approaches, classes grant choice options to multi-inheritance designs that can be utilized in the context of single inheritance languages.

Logtalk in the classroom

Logtalk is being used at UBI (University of Beira interior) to instruct object-oriented programming and object-oriented extensions to common sense programming to

undergraduate students. Teaching object-oriented ideas, the use of Logtalk has furnished fascinating results. Common object-oriented languages such as C++ and Java are class-based. These languages inherit syntax and standards from essential languages such as C. They re-quire perception ideas such as static versus dynamic allocation, approach argument and return types, library imports, and important methods, in order to write easy programming examples. These standards get in the way of instructing key object-oriented principles like encapsulation, message sending, or inheritance. In contrast, Logtalk objects encapsulate predicates. There is no want to speak about strategies and variables, or characteristic and statistics members, earlier than defining a easy predicate and sending the corresponding message. In addition, the Logtalk guide for each prototypes and lessons potential that we can educate fundamental object-oriented ideas the use of less difficult prototype hierarchies earlier than explaining the distinction between training and cases or between instantiation and specialization mechanisms. There are no main, static, void, encompass or import keywords cluttering and distracting the scholar from the notion that a given instance tries to convey, as it occurs in C++ or Java. There is no rich, integrated, development surroundings whose fundamentals want to be understood and mastered earlier than easy applications can be written as in Smalltalk. A easy textual content editor suffices. For college students with a fundamental information of Prolog programming, Logtalk is an perfect getting to know device for a easy transition from good judgment programming to object-oriented programming, due to the use of acquainted Prolog syntax and semantics and to the guide of a broad vary of object-oriented systems.

Logtalk in numbers

In the final two years, Logtalk was once downloaded an common of 270 copies per month5 (9 copies per day). In addition, Logtalk is dispensed with YAP, an open-source Prolog compiler. New releases are in ordinary introduced solely in the Logtalk mailing listing (that is subscribed with the aid of round 70 users) and in the Freshmeat internet web site (a internet index of cross-platform software, normally open-source products; round 10 customers have subscribed to new Logtalk releases thru this internet site). The Logtalk net website online is linked from round one hundred net sites, ranging from hyperlinks in

private pages to net directories of programming resources. These are modest numbers when in contrast to the estimated dimension of the Prolog person community. They exhibit that there is sufficient pastime on Prolog object-oriented extensions to proceed Logtalk development, however additionally that greater efforts need to be achieved to make bigger the use of Logtalk.

Reference:

[1] ACM/IEEE. Computer Curricula 2001: Volume II — Computer Science — Straw-man Draft. http://www.computer.org/education/cc2001/, March 2000.

[2] Bjarne Stroustrup. *The C++ Programming Language*. Series in Computer Science. Addison-Wesley, 3rd edition, 1997.

[3] James E. Rumbaugh. Relations as semantic constructs in an object-oriented language. In Meyrowitz [122], pages 466–481.

[4] Pattie Maes. Concepts and experiments in computational reflection. In Meyrowitz [122], pages 147–155.

[5] A. Tanenbaum. *Operating Systems — Design and Implementation*. Software Series. Prentice-Hall, 1987.

[6] Henry Lieberman. Using prototypical objects to implement shared behavior in object-oriented systems. In Meyrowitz [124], pages 189–214.

[7] Francis G. McCabe. *Logic and Objects*. Series in Computer Science. Prentice Hall, 1992.

[8] Alan Burning. Classes versus prototypes in object-oriented languages. In *Proceedings of the ACM/IEEE Fall Joint Computer Conference*, pages 36–40, Dallas, Texas, November 1986.

[9] Swedish Institute for Computer Science. Quintus Prolog Home Page. http://www.sics.se/isl/quintus/.

[10] Eric Borgers. OPL User Manual. http://www.amzi.com/download/freedist_opl.htm, 2001.

[11] Bjarne Stroustrup. *The Design and Evolution of C++*. Addison-Wesley, 1st edition, 1994.

[12] Dan G. Bobrow, Linda G. Michiel, Richard P. Gabriel, Sonya E. Keene, Gregor Kiczale, and David A. Moon. Common lisp object system specification. *SIGPLAN Notices*, 23, 1988.

[13] Iain Craig. *The Interpretation of Object-Oriented Programming Languages*. Springer Verlag, December 1999.

[14] Vitor Santos Costa. YAP Home Page. http://www.cos.ufrj.br/~vitor/Yap/.

[15] Sun Microsystems, Inc. Javasoft web site. http://www.javasoft.com/.

[16] Donald E. Knuth. The literate programming paradigm. *Computer Journal*, 27(2):97–111, May 1984.

[17] Lisa Friendly. The design of distributed hyperlinked programming documentation. In *International Workshop on Hypermedia Design '95*, 1995.