Web Framework for Dynamic Programming Languages

Ana Stankovic[1], Dragan Stankovic[2], and Dusan Tatic[3]
[1]The Faculty of Information Technologies at Metropolitan University Tadeusa Koscusca 63, 11000 Belgrade, Serbia
{ana.stankovic@metropolitan.ac.rs}

[2]The Faculty of Technical Sciences, University of Pristina Kneza Milosa No. 7, Kosovska Mitrovica, Serbia
{sfsgagi@gmail.com}

[3]The Faculty of Electronic Engineering, University of Nis Aleksandra Medvedeva 14, 18000 Nis, Serbia
{dusan@dragongroup.org}

ABSTRACT: *Dynamic programming languages have been gaining popularity in recent years, along with the popularity of web 2.0 apps. A lot of web frameworks have been created based on dynamic programming languages, such as Rails (developed in Ruby) Grails (in Groovy) and Django (in Python). These frameworks have been designed to improve the productivity of web application development by encouraging Agile methodologies of work and keeping the code simple and maintainable. This paper has analysed whether the same gains can be made in multiagent system (MAS) development. We have compared the code quality and size in Groovy (asynchronous dynamic programming language), Python (asynchronous programming language) and Ruby (static-typed programming language) using independent implementations of asynchronous dynamic programming algorithms in all four languages. The results of this analysis can be generalized for other MAS algorithms.*

Keywords: Multi-Agent Systems, Dynamic Programming Languages

Received: 26 April 2023, Revised 19 July 2023, Accepted 28 July 2023

DOI: 10.6025/jio/2023/13/4/106-111

1. Introduction

Dynamically typed programming languages have recently turned out to be really suitable for specific scenarios such as Web development, application frameworks, game scripting, interactive programming, rapid prototyping, dynamic aspectoriented programming and any kind of runtime adaptable or adaptive software. The main benefit of these languages is the simplicity they offer to model the dynamicity that is sometimes required to build high context-dependent software. Common features

of dynamic languages are metaprogramming, reflection, mobility and dynamic reconfiguration and distribution [1]. Out of special interest to us when considering dynamically typed programming languages in the context of multi agent systems is rapid prototyping with additional benefit of adaptability which is not the subject of this paper.

Russel and Norvig in [2] define artificial intelligence as a scientific study of agents that are able to perceive the environment and perform actions. Since the research of multiagents is still in its infancy, there is no universal consensus on an unequivocal definition of the concept of agent.

Nevertheless, definition provided by Wooldridge and Jennings [3], [4] is becoming widely accepted by a growing number of researchers, which is why it can be regarded as being one of the most complete definitions. According to this definition, an agent is:

"a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives"

Interest in studying multi-agent systems usually stems from the interest in artificial (software or hardware) agents, such as the agents living on the Internet, for example. Examples of those agents are trading agents, game-playing agents that assist or replace human players in multi-player games, autonomous robots in multi-robot environments and the like.

Software agents can be regarded as a natural extension of the concept of software objects. Object-oriented programming has introduced abstraction entities – objects to the structural programming paradigm. Similarly, agent-based programming introduces new entities – agents, which, in contrast to objects, have an independent execution thread. Therefore, in comparison to objects, agents have the ability of acting in a goal-directed manner, for example, by interacting with other agents, reading sensors or sending commands to effectors, while objects only passively respond to procedure calls. In short, it can be stated that agents represent intelligent, adaptable software applications, designed with the purpose of meeting different, user-defined requirements.

In most of the cases, even separate action of agents can be useful. Nevertheless, agents achieve their fullest potential by interacting with other agents, thus making multi-agent systems. Most of these systems are heterogeneous because they consist of different types of agents that have different functions within the observed system. Agents act either in synergy with the purpose of achieving the common goal or competitively with the purpose of achieving contradictory goals.

What follows are detailed definitions of important concepts associated with agents and multi-agent systems. After that, comparison of one simple MAS-used search algorithm implemented in dynamic programming languages Ruby,Python and Groovy with statically typed Java programming language will be performed.

2. Concept of an Intelligent Agent

It has already been pointed out that there is no universally accepted definition of the term agent. Debates concerning this issue are still under way. In fact, while there is a general consensus that autonomy is something that is always associated with agents, not all the details have been cleared up. Perhaps the main reason for which one universally accepted definition is difficult to find lies in the fact that agents are used for finding solutions to problems within various domains. Therefore, for some applications, the ability of agents to learn on the basis of their previous experience is very important. However, for some other applications learning is not only unnecessary but also undesirable at times (an example supporting these facts is found in the air-traffic control system; the passengers would probably not like the situation in which the system modified flights schedule at run time on the basis of previously learned facts).

Agents can be regarded as an approach to structuring and development of software that offers certain advantages and that is suitable for certain types of applications (some papers see agents as evolutionary in relation to objects) [5]. Agents' characteristics to reduce the interdependence of application components can represent their most advantageous characteristics. Agents are autonomous, which can be regarded as some kind of encapsulation [5]. While objects have their own methods that are controlled by external entities, agents do not allow external entities to control them. When an agent gets a message, being autonomous, it decides what is to be done with that message by itself. Interdependence of application components is reduced not only by the agents' autonomy, but also by their robustness, reactivity and proactiveness. For example, when an

agent enters goal-directed phase, agent itself is responsible for the process of realization of that goal. It is not necessary to perform constant supervision and checking. Analogously, object can be regarded as a reliable employee that has no initiative or sense of responsibility; supervision of that employee requires increased level of communication. On the other hand, agent can be regarded as an employee that takes initiative and has the sense of responsibility. Therefore, supervision of that employee does not require increased level of communication, which is why it can be stated that there exists lower level of interdependence.

Reduced interdependence leads towards software systems that are more modular, more decentralized and more easily changeable. This resulted in the fact that agents started being used in wide specter of applications, especially in applications that are regarded as open systems, that is, applications which have been designed and written by different authors without their mutual communication. Of course, this entails the introduction of certain standards. Examples of these systems include semantic web and grid computing.

The fact that some agents are proactive and reactive makes their mode of problem solving similar to human. That feature resulted in a great number of applications in which agents are used as substitutes for humans within some limited domains. One such example is an application in which software agents are used to replace human pilots in military simulations [6]. Another example are computer games. The game Black & White uses agents that are based on BDI (Belief-Desire- Intention) model. Another field within which agents have been practically applied is the film industry. Producers of the film Lord of the Rings used the software package Massive to generate armies of orcs, elves and humans. Each individual character was modeled as an agent. Other types of applications where agents show their advantages include intelligent assistants, e-trade, production and modeling of business processes [7], [8].

3. Dynamic Programming Languages

In the last few years, the development of Web 2.0 applications has brought about the increase of interest in dynamically typed programming languages. Great number of frameworks that enable efficient development of web applications and promote Agile application development methodology have been developed. Terms such as *DRY* (don't repeat yourself), *KISS* (keep it simple but not simpler) and convention over configuration have been adopted by the programmers and these stand for the major directions the programmers follow in the process of application development. Growing interest in programming languages that increase productivity of web developers has resulted in further expansion of their use in the desktop applications domain making languages such as Ruby, Python and Groovy extremely popular nowadays. What follows is a short description of these programming languages and their important features that improve efficiency of rapid prototyping of multi-agent systems algorithms.

3.1. Ruby

Ruby is a dynamic programming language which is characterized by a complex but very expressive grammar and a good core class library with a rich and powerful *API*. Ruby is based on elements of Lisp, Smalltalk and Perl, but its grammar is such that C and Java Programmers find it easier to learn. Ruby is a programming language that is completely object-oriented, but it is also suitable for procedural and functional programming styles . Ruby includes powerful metaprogramming mechanisms and can be used for the creation of new languages which are suitable for certain domains or for the creation of *DSLs* (Domain Specific Languages). [9]

3.2. Python

Python is a dynamic, object-oriented programming language which can be used for various forms of software development. It offers strong support for integration with other languages and tools, huge range of standard libraries and can be learned in a few days. Many programmers who had had experience with Python programming reported substantial productivity gains and easier and more maintainable code development. [10]

3.3. Groovy

Groovy is a developing dynamically typed programming language for the Java Virtual Machine. It builds upon the strength of Java but it also possesses additional features that are inspired by languages such as Python, Ruby and Smalltalk. It supports *DSLs* and test driven development. Its main advantage lies in the fact that it smoothly integrates with Java objects and libraries. In fact, "Groovy is Java and Java is Groovy". Groovy is the second referent language for Java. platform (Java programming language being the first), which further explains its relation with Java [11].

3.4. Support for DSL creation

Common feature of all languages that have been described so far is that they can create *DSLs*, that is, new programminglanguages that enable more efficient development of applications for specific domains. This feature is important because it enables, for example, the creation of a specific *DSL* for the domain of multi-agent systems, which, in turn, enables more efficient modeling and development of multi-agent applications without losing the interoperability of the code written in that new *DSL* with standard libraries.

4. Distributed Path Finding Problem

Majority of problems that occur within multi-agent systems are focused on how to meet some global constraints in a distributed way, that is, how the agents can optimize some objective function in a distributed manner. In most cases, it is achieved with the help of four families of techniques and specific problems. Those techniques are:

• Distributed dynamic programming (applied here to the path planning problem)

• Distributed solutions for Markov Decision Problems (MDP ÀÛ±Ü Markov Decision Problems)

• Algorithms of optimization algorithms of economic functions (matching and scheduling problems)

• Coordination on the basis of social laws and conventions (example of traffic regulations)

With the purpose of illustration, distributed dynamic programming will be applied to the path planning problem. Path planning problem consists of a weighted directed graph with a set of n nodes N, directed links L, a weight function w: $L \rightarrow R^+$ and two nodes s, $t \in N$. The goal is to find a directed path from s to t that will have minimal possible total weight. Generally speaking, a set of goal nodes $T \subset N$ can be considered, and the shortest path from s to any of the goal nodes $t \in T$ can be looked for.

This kind of abstract framework can be applied in various domains. It can certainly be applied in cases of some specific networks (for example, transportation or telecommunication network). Nevertheless, it can be applied to other problems as well. For example, in a planning problem the nodes can be states of the world and the arcs can be the actions that the agent performs. In that case, the weights stand for the cost of each action (for example, the time needed for the action) (37) (38).

```
procedure ASYNCHDP (node i)
if i is a goal node then
        h(i) ← 0
else
        initialize h(i) arbitrarily (e.g. to ∞ or 0)
repeat {
        forall neighbors j do
                f(j) ←w(i, j) + h(j)
        h(i) ← min_j f(j)
}
```

Figure 1. Asynchronous dynamic programming algorithm

5. Asynchronous Dynamic Programming

The problem of finding the best path is the problem that has been thoroughly studied in computer science. Distributed solution will be considered here, in which each node performs a local computation with insight only into the state of neighboring nodes. The principle of optimality underlies the solutions that will be illustrated: "if node x belongs to the shortest path from s to t, then the part of the path from s to x (or from x to t) must also be the shortest path between s and x (that is, x and t). This principle enables an incremental divide- and-conquer procedure, also known as dynamic programming.

Let $h^*(i)$ represent the shortest distance from any node i to the goal node t. In that case, the shortest distance from i to t via node j neighboring i is shown as: $f(i,j) = w(i,j) + h^*(j)$, and $h^*(i)$-$minj f(i,j)$. Having these facts in focus, *ASYNCHDP* algorithm has each node perform the procedure shown in Figure 1. Within this procedure, each node i maintains a variable $h(i)$ that stands for an estimate of $h^*(i)$.

It can be proved that *ASYNCHDP* procedure always converges to the true values, that is, that h will converge to h^*. In this case, convergence will require additional step for each node in the shortest path, which means that in the worst case convergence will require n iterations. However, this is not so good for realistic problems. Not only will convergence be slow, but this procedure also assumes the existence of agent for each node. In typical search spaces it is not possible to enumerate all nodes in an efficient way and allocate each of them a separate process. (For example, chess has approximately 10^{120} positions). For that reason, programmers often turn to heuristic versions of the procedure that require smaller number of agents.

6. Results

We have implemented the above mentioned algorithm in Java, Groovy, Python, and Ruby and used the number of lines of code as a measure for evaluating their rapid prototyping abilities. It can be further discussed whether the number of lines of code is a measure that can be suitable for the estimate of efficiency of some programming language in a specific domain (multi-agent systems). The lines of code will depend on developer's experience with certain programming language and the applied code style rules. The code that we used here and the way it was generated are sufficient for the process of drawing general conclusions. One such conclusion is that there is a big difference between statically typed Java and dynamically typed languages that were considered here. The graph shown in Figure 2 illustrates this difference in the best possible way.

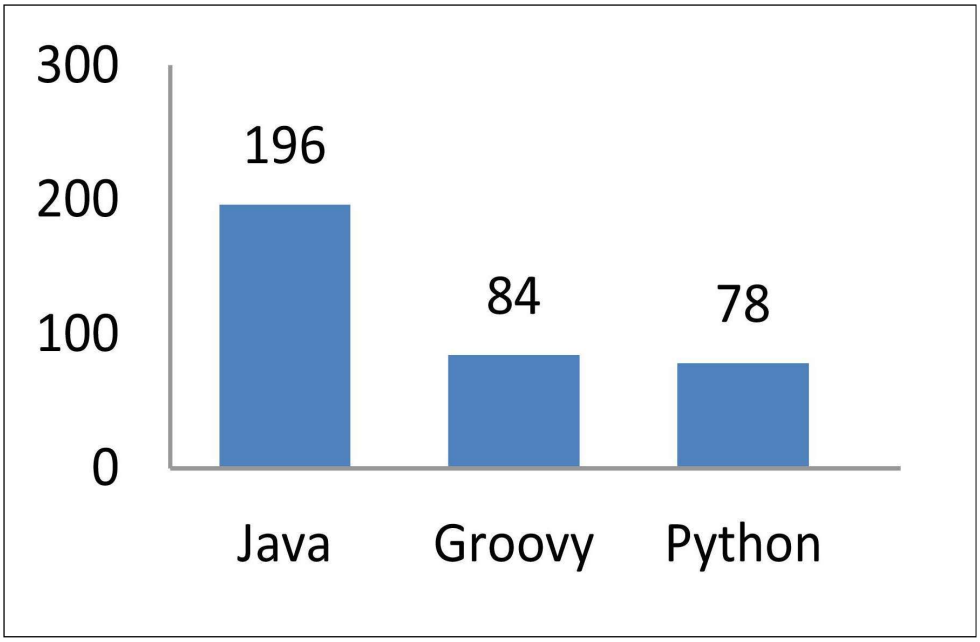

Figure 2. Lines of code (LoC) for the complete test application that relies on AsynchDp algorithm for different programming languages

7. Conclusion

In the domain of rapid prototyping of MAS algorithms dynamically typed programming languages clearly have advantages over classic statically typed languages, such as Java. These advantages, as we have shown, are noticeable even in the simplest examples. Apart from the advantages that are reflected in reduced number of code lines and higher productivity, big advantage is also seen in the increased code readability and subsequent easier influx of broad community of developers in what was previously done, as well as in more efficient code maintenance and iterative improvement.

The conclusion that can be drawn is that dynamically typed programming languages should be given preference over statically typed languages whenever possible.

The future will probably bring increased interest in functional programming languages in the domain of multiagent systems as well. Environments and tools with the most efficient support and broadest community of users will become dominant while the remaining projects will disappear in time.

Acknowledgement

The work presented here was supported by the Serbian Ministry of Education and Science (projects III44006 and III42006).

References

[1] Ortin, F. (2011). Type Inference to Optimize a Hybrid Statically and Dynamically Typed Language. *The Computer Journal, 54*(11).

[2] Russell, S., Norvig, P. (2009). *Artificial Intelligence - A Modern Approach.* Prentice Hall.

[3] Wooldridge, M. (2002). *An Introduction to MultiAgent Systems.* Chichester: John Wiley & Sons.

[4] Wooldridge, M., Jennings, N. (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review, 10*(2), 115–152.

[5] Wooldridge, M. (1999). Intelligent Agents. In G. Weiss (Ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* The MIT Press, Cambridge, Massachusetts, pp. 27-78.

[6] Tidhar, G., Heinze, C., Selvestrel, M. (1998). Flying together: Modelling air mission teams. *Applied Intelligence, 8*(3), 195–218.

[7] Shen, W., Norrie, D. (1999). Agent-based systems for intelligent manufacturing: A state-of-the-art survey. *Knowledge and Information Systems, An International Journal, 1*(2), 129–156.

[8] Jennings, N., et al. (2000). Autonomous agents for business process management. *International Journal of Applied Artificial Intelligence, 14*(2), 145–189.

[9] Flanagan, D., Matsumoto, Y. (2008). *The Ruby Programming Language.* O'Reilly Media.

[10] [Online] http://www.python.org/

[11] [Online] http://groovy.codehaus.org/