Revista Eletrônica de Iniciação Científica em Computação, 2026, 24:1
ISSN: 3085-8461 • doi: 10.5753/reic.2026.7183

Research Paper

Facilitating the adoption of the JaCaMo Framework in developing Multi-Agent Systems through of a Visual Studio Code extension

Mustafa de Almeida Neto [Federal Center for Technological Education Celso Suckow da Fonseca (Cefet/RJ) |*mustafa.neto@aluno.cefet-rj.br*]
Bruno Freitas [Federal Center for Technological Education Celso Suckow da Fonseca (Cefet/RJ) |*bruno.freitas@cefet-rj.br*]
Nilson Lazarin [Federal Center for Technological Education Celso Suckow da Fonseca (Cefet/RJ) |*nilson.lazarin@cefet-rj.br*]
Diego Castro [Federal Center for Technological Education Celso Suckow da Fonseca (Cefet/RJ) |*diego.castro@cefet-rj.br*]

Av. Governador Roberto Silveira, 1.900 - Prado – Nova Friburgo/RJ - CEP: 28635-000, Brazil.

Abstract. An agent is an autonomous software entity with the ability to pursue its own goals or respond to its environment. A group of agents interacting with themselves is called a Multiagent System (MAS). A MAS, in turn, is defined as having dimensions which are the agents themselves, the environment on which they interact, and the set of norms they must obey. Each dimension has its own tools for its development, among which the most used are those provided by the JaCaMo framework. Consequently, the development of MAS presents challenges to beginner programmers, as it is necessary to master the tools for these dimensions by using their command-line interfaces, and the higher learning curve on the languages themselves. Despite numerous solutions to ease MAS's development with JaCaMo, most are deprecated or lack common Integrated Development Environment (IDE) features. Given this scenario, this work presents an Extension for the VSCode IDE that integrates multiple features, including project startup code, MAS execution, and syntax highlighting, thereby reducing technical barriers and improving MAS development for programmers. As a result, we present a downloads metric for its underlying infrastructure, which shows a significant increase after its release, indicating interest in extension and ultimately filling a gap in the lack of proper tools for JaCaMo development.

Keywords: MAOP, JaCaMo Framework, Multi-Agent Systems, Visual Studio Code, Belief-Desire-Intention

Received: 14 December 2025 • **Accepted:** 13 March 2026 • **Published:** 10 April 2026

1 Introduction

Multiagent Systems (MAS) are a group of autonomous software entities called agents which, in turn, are capable of sensing the environment, adapt to the current context, and interact with other agents to deliberate about which actions to take Bordini *et al*. [2020]. A multiagent programming paradigm is a MAS engineering model that posits three dimensions: an *agent* dimension, an *environment* dimension, and an *organization* dimension. Based on the Multi-Agent Oriented Programming (MAOP) model, Boissier *et al*. [2013, 2019] proposed the framework JaCaMo (Boissier *et al*. [2016]), which provides a comprehensive infrastructure for developing each of these dimensions: **Ja**son for the agent dimension (Bordini and Hübner [2006]), **Ca**rtago for environment dimension (Ricci *et al*. [2007]), and **Mo**ise (Hübner *et al*. [2007]) for organization dimension.

However, despite JaCaMo offering all the tools for developing a MAS, its adoption for beginner programmers is not easy. To create a project, it is necessary to use command line tools and edit many configuration files, where each file has its own distinct syntax - such as *.jcm*, *.asl*, *.java*, and *.xml* - all organized according to MAOP dimensions.

In the context of making the MAOP learning curve easier for beginners, two main approaches stand out. The first is the *JaCaMo-Web* by Amaral and Hübner [2020], which proposes an approach based on interactive programming that allows the creation, inspection, modification, and destruction of agents, artifacts, and organizations in real time, without requiring a system reboot. These functionalities reduce development time and facilitate incremental prototyping by offering a web interface through a RESTful API. The second is the *ChonIDE* from Souza De Jesus *et al*. [2023], an IDE targeted for the development of Embedded MAS, which integrates functionalities from both from the reasoning layer (agent) as well as the firmware (microcontrollers). It aims to centralize, in a single environment, all the necessary tools for developing an Embedded MAS, including remote access to a physical device.

Despite its contributions, *JaCaMo-Web* does not replace a complete development environment, as it relies on tools such as *Gradle*, *Docker*, and *Node.js*, which in turn require project configuration using the command line. Besides, it focuses on dynamic instance manipulation at runtime, without supporting a project's life cycle. Similarly, while offering an important contribution to the development of distributed and embedded MAS, ChonIDE has significant limitations. First, it allows only the development of the agent dimension. It requires the Linux operating system, limiting its adoption in different educational scenarios such as laboratories running Windows. Many common modern IDE usability features are lacking, such as syntax highlighting, autocomplete, multi-platform compatibility, multiple files open in different tabs, and an intuitive UI Siqueira *et al*. [2024]. Given these limitations, it is evident the necessity of tools to abstract the technical complexity of managing a MAOP project with JaCaMo, allowing a more accessible and integrated development environment, aimed towards easy prototyping and learning.

This work proposes a Visual Studio Code (VS Code)

extension to simplify the creation and execution of JaCaMo projects. It is an extended version of the original work by the authors presented at WESAAC 2025 (Neto *et al*. [2025]), including a more detailed explanation of the extension's inner workings and results taken by analyzing download metrics for its infrastructure. The VS Code extension offers an intuitive UI, eliminating the need to interact with the command line for basic operations such as file and directory manipulation and project execution. This way, the user experience improves, and the learning curve to use the JaCaMo framework is lower, especially for beginners and in educational scenarios.

This work is organized as follows. Section 2 presents the necessary background concepts needed, along with related works regarding IDE support for JaCaMo. Section 3 presents the proposed extension for the development of the JaCaMo application within VSCode. Section 4 details the implementation of the extension. Section 5 presents the results obtained. Section 6 presents the conclusion.

2 Background

The development of MAS comprises a series of fundamental steps, including requirement analysis, interaction modelling, agent coding, and testing and maintenance. The distributed and autonomous nature of such systems poses another challenge, as it requires not only MAS knowledge, such as perception, reasoning, and communication between agents, but also knowledge of many tools needed to program these dimensions. Recent studies, from Souza De Jesus *et al*. [2023]; Siqueira *et al*. [2024]; Lima *et al*. [2025], highlight that the lack of proper development tools hinders productivity, making it challenging for new developers to create MAS solutions.

In this context, an *Integrated Development Environment* (IDE) plays a fundamental role by consolidating many essential project management functionalities within a single environment, including code editing, execution, and debugging. This integration results in greater automation of tasks, improving productivity. Nowadays, one of the most widely used IDEs is Visual Studio Code (VSCode), known for its flexibility in supporting the addition of *extensions*. The extensions enable the addition of even more functionality in the IDE, such as automated code completion, syntax highlighting, automated code execution, and support for additional programming languages.

The different approach adopted by VSCode is the *extension-in-IDE* paradigm, in which each extension functions as a small, incorporated application running on top of the core of the IDE. This model enables the integration of specific functionality in a lightweight manner by leveraging the IDE's visual elements. According to Liu *et al*. [2025], this enables the adoption of the tool for different domains and workflows, without the need to adopt new tools for different domains

Recent work has successfully explored this paradigm to create a variety of new extensions. Almeida Neto and Lazarin [2024] integrated ChatGPT to VSCode to help the detection of code vulnerabilities. Vahlbrock *et al*. [2022] created *VSCode Migrate*, a tool for semi-automatic migration with projects with low testing coverage, creating customized interfaces and editing oriented by external scripts. Durelli *et al*. [2022] proposed *Divinator*, an extension for automated code summary in natural language by using deep learning. These examples demonstrate that the *extension-in-IDE* paradigm has been utilized to introduce new, advanced functionalities while maintaining usability and developer familiarity with the IDE — features explored in this work to support the JaCaMo framework.

Table 1. VSCode Extensions for JaCaMo available at the VSCode Store

Extension by	Functional coverage
Krausburg [2020]	Syntax highlighting and snippets JaCaMo (`.asl` and `.jcm`) files
Tkampik [2018]	Syntax highlighting for `.asl` e `.mas2j` files
Tedeschi [2021]	Syntax highlighting for `.asl` files
U473t8 [2019]	Syntax highlighting for `.jcm` files

Table 1 summarizes the current available extensions in the VSCode Store for the JaCaMo framework. As the table shows, only 4 extensions have been found. All these tools offer only basic syntax highlighting for the JaCaMo ecosystem, such as *.asl*, *.mas2j* and *.jcm*. They don't offer more advanced functionality, such as automated project creation, interactive execution, or integration of all MAOP dimensions (agent, environment, and organization). Besides, the majority of these extensions are outdated, with their last commits from 2018 to 2021, reinforcing the lack of solutions to develop MASs using JaCaMo within VSCode. In this scenario, this work aims to fill this gap by creating a modern and functional extension, aligned with the best practices for their creation according to the *extension-in-IDE* paradigm.

3 Proposal

The traditional workflow for the development of applications using the JaCaMo framework requires that the developer manually install Java and execute a series of commands using Gradle from the Command line Interface (CLI). Currently, some tools help simplify this process, such as the JaCaMo-CLI, but they also introduce new challenges: the user must install the tool by cloning the project from GitHub and compiling it by running the command *./gradlew createBin* and move the resulting binary to the folder */usr/bin*. Even worse, the compilation script is not compatible with Windows, leaving these users unable to use JaCaMo.

Instead, we propose unifying all these functionalities into a single extension, leveraging the IDE's infrastructure to improve beginners' learning outcomes and ease experimentation and adoption of JaCaMo. The extension aims to integrate all the necessary JaCaMo tools into the IDE's interface, interacting directly with jacamo-cli, which runs Gradle to compile the project using Java, as shown in Figure 1.

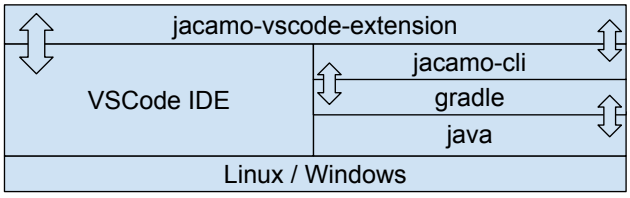


Figure 1. Interaction of the extension with other components

The main functionalities proposed are as follows:

I) **Creation of new projects**, automating the generation of file structure needed for a project with the MAOP

paradigm;

II) **Simplifying the execution of MAS**, offering a graphical UI to compile, start, and stop MAS from the IDE's UI;

III) **Support multidimensional coding**, offering specific functionalities for editing each type of file of JaCaMo — the languages Jason, CArtAgO, and

IV) **Ensure multiplatform compatibility**, making the extension work both in Linux and Windows. Table 2 presents a summary of all these requisites.

Table 2. Main functionalities of the extension

Functionality	Implementation
Creation of new projects	Button on the interface that execute `jacamo-cli` to create a project
Simplifying the execution of MAS,	Buttons on the interface show the outputs of jacamo-cli on the integrated terminal
Support multidimensional coding	*Syntax highlighting* for files `.asl` (Jason), `.java` (CArtAgO) and `.jcm` (Moise+).
Ensure multiplatform compatibility	Abstraction of Operating System commands, supporting Windows and Linux.

4 Development

This section outlines the adopted methodology for constructing the extension. The first step was defining the requisites. This was done through informal interviews with users of the JaCaMo framework and members of research groups about agents, aiming to understand the most desired functionalities. From these interviews, the most important feature requested was to make available an *Activity Bar* with commands from CLI of JaCaMo, highlighting the existing difficulties users face with CLI. Other requested features included syntax highlighting for the framework's languages and publishing the extension on the Marketplace to make the installation and update processes easier.

The development followed the directives for creating VSCode extensions (Microsoft [2026]). Node.js was chosen as the language for its implementation, taking advantage of its ecosystem for web development to help the integration with VSCode API, which is responsible for the interaction with the editor, allowing the creation of commands, custom configuration, and integration with the UI. For packaging, publishing, and testing, it was necessary to use the VSCode Extension CLI (VSCE), which enabled rapid execution of the tool from the IDE. Finally, the project was developed following the best practices for code versioning, using its issue tracking tool from Github.

Besides the technologies mentioned earlier, the extension used specific dependencies for testing, code analysis, and VSCode compatibility. For the analysis, the type libraries from TypeScript, such as *@types/vscode*, *@types/mocha*, and *@types/node*, were used to standardize the code. Another tool used was *eslint*, which helped to enforce good code practices, such as indentation, line breaking, and grouping of imports. Finally, the libraries *@vscode/test-cli* e *@vscode/test-electro* were used to run automated tests for the application.

4.1 Technology-enablers for distribution of JaCaMo-CLI for Windows and Linux

As mentioned earlier, jacamo-cli must be compiled manually. For Linux distributions based on Debian, the repository is downloaded and compiled according to the project's instructions, then packaged with the *Advanced Packaging Tool* (APT) from Debian. This packaging also resolves the Java dependencies needed by jacamo-cli (*openjdk-21-jdk* or *temurin-21-jdk*) so that they are automatically installed. This package is publicly available from the CHON research group repository[1].

However, there isn't support of jacamo-cli for Windows systems. To overcome this limitation, the jacamo-cli project was first downloaded and opened in IntelliJ, an IDE for Java applications, and then exported as an encapsulated Java program (*.jar*). This jar is then converted to an executable Windows file using Launch4j. Afterwards, an installer was created to include all the necessary Java dependencies to avoid double installations of it, using Inno Settup, a free installer creation tool for Windows programs. The added Java dependencies here were JDK 21 from the Eclipse Temurin project, a free, open-source reference implementation of Java. The generated installer is also available from the CHON group repository[2].

4.2 Extension development

Figure 2 shows the execution steps of the extension, from the command activation by the user, followed by its execution by jacamo-cli, until its final output on VSCode terminal.

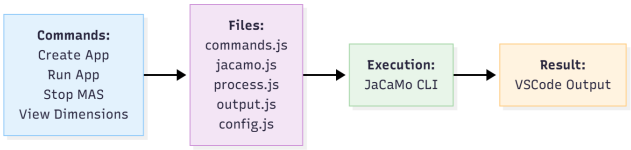


Figure 2. General extension execution flow: commands, files, execution, and output

The project structure was created using the VSCE (Visual Studio Code Extension), which provides the basic files needed to create VSCode extensions. Next, the *package.json* file is configured to register commands, permissions, and shortcuts that define the extension's behaviour in the editor's environment.

To support this workflow, the extension's architecture is organized into distinct modules to ensure a clear separation of concerns. Commands triggered via the Activity Bar are routed through the main entry point (*extension.js*) to the Command Handler (*commands.js*), which delegates logic to specialized services. The App Manager (*jacamo.js*) centralizes domain logic, including environment validation, *.jcm* file discovery, and lifecycle management (e.g., detecting running instances). Actual execution is managed by the Process Runner (*process.js*), which wraps system calls to the JaCaMo CLI, ensuring all operations use the correct working directory for reproducibility. Feedback is handled by the Output Manager (*output.js*), which parses and formats standard output streams with visual cues, while configuration and path resolution are abstracted by *config.js*. This layered approach

[1] `https://jacamo-cli.chon.group`
[2] `https://jacamo-win.chon.group`

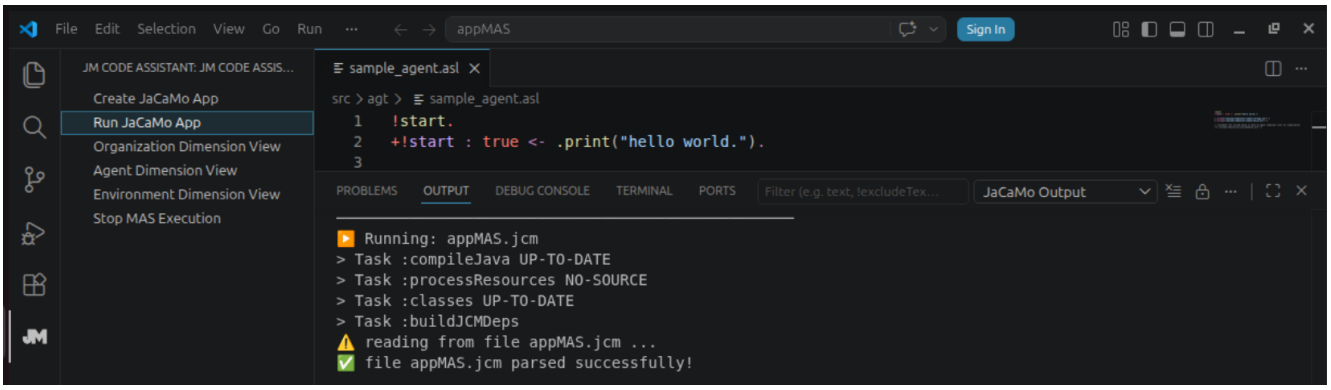


Figure 3. Activity bar on VSCode

keeps the command entry points lightweight while isolating system execution details.

To simplify the life cycle of a JaCaMo project, the following main functionalities were implemented. Also, to provide a deeper understanding of the internal interactions between the extension components and the underlying infrastructure, the detailed execution flows for the three primary commands are presented below.

Create JaCaMo App: A command to create an initial structure of a project using JaCaMo, asking the user only the name of the application.

Figure 4 shows the initial file structure for a new project. Agents using the Jason language are defined in the *src/agent* folder. Organization, which is the set of goals that each agent in the MAS must achieve, is defined in an *org/org.xml* file. The main JaCaMo project file is *exampleProject.jcm*. This file contains the MAS declaration, which comprises *agents* and *workspaces*. A workspace, in turn, contains a set of *artifacts* on which an agent can perceive the environment as beliefs in his mind and a set of actions that can be taken on them. Initially, the project has only the agent bob, defined in *sample_agent.asl*. This agent acts on the artifact *c1* defined in the *.jcm* file. The artifact is then defined on the *Counter.java*, which is, initially, is a simple counter.

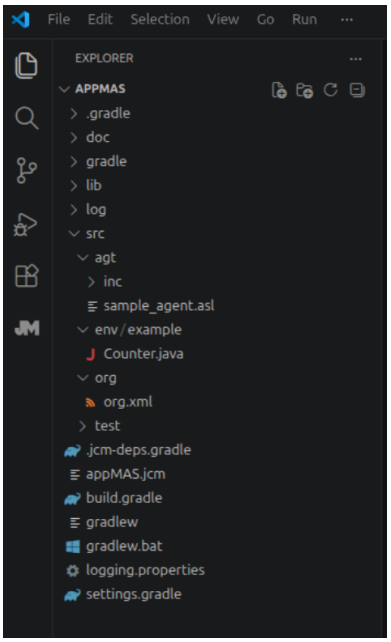


Figure 4. Initial project file structure from "Create JaCaMo App" command

Figure 5a demonstrates the flow of the "Create JaCaMo App" command. When the user presses the interface button, the Command Handler prepares the commands to be run by the App Manager. The manager then invokes the jacamo-cli to scaffold the project structure and returns a success confirmation to the IDE output.

Run JaCaMo App: the user is capable of running *.jcm* file directly from the VSCode environment, without the need to use external tools. The extension can also detect multiple running instances of a JaCaMo application, preventing potential conflicts.

The "Run JaCaMo App" workflow is illustrated in Figure 5b. This process encompasses configuration validation, the automatic discovery of *.jcm* files within the workspace, and an optional conflict resolution mechanism to handle already active instances. It concludes by spawning the JaCaMo process, with live streaming of standard output and errors directly to the integrated VS Code output panel.

Stop MAS Execution: Identifies and ends any MAS currently running using jacamo-cli commands

Figure 5c depicts the "Stop MAS Execution" routine. This sequence validates the configuration, retrieves a list of active MAS instances for user selection, and invokes the CLI stop command. The operation's result is then relayed back to the user via the output channel and an informational message.

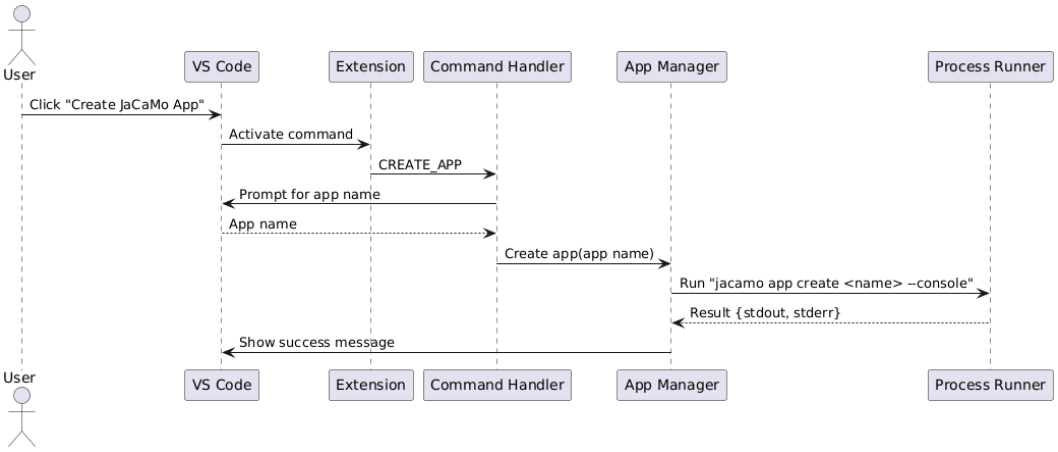
Output Panel: All the logs are exhibited on a single, integrated panel on VSCode. The data is formatted with icons and visual hints, easing the readability and debugging.

Web interface access: the extension has shortcuts to open web interfaces to inspect the organization (Figure 6), agent (Figure 7), and environment dimensions of a MAS (Figure 8). These views are easily accessible by pressing their respective buttons shown on Figure 3.

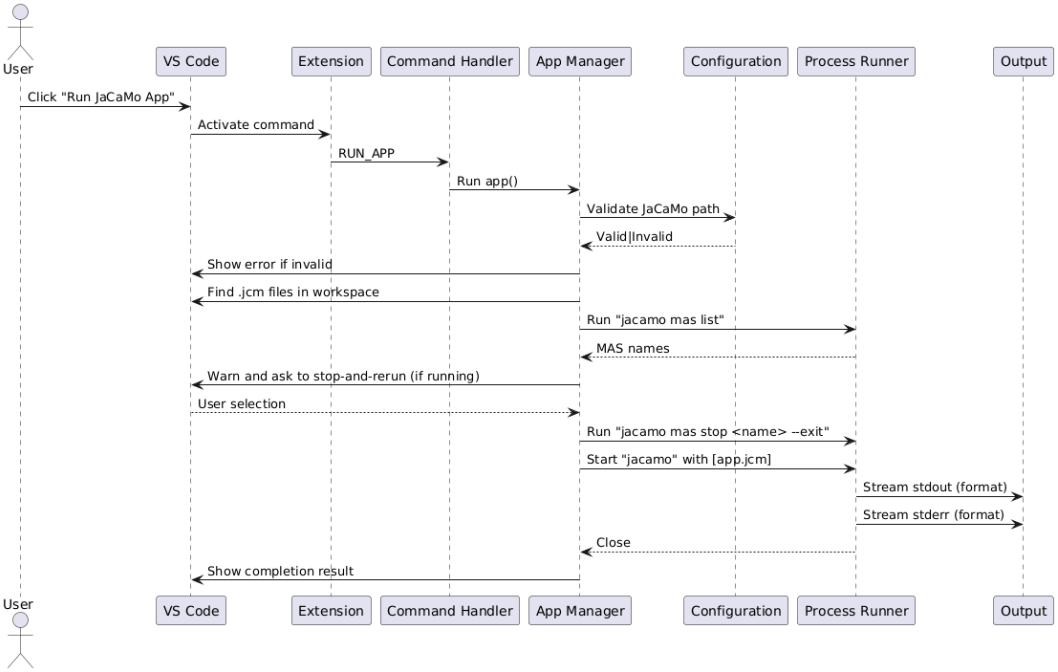
To improve usability, the extension adds an *Activity Bar* to VSCode's side panel, where all the functionalities above are implemented as buttons, as shown in Figure 3. This, in turn, completely removes the need to use any CLI command.

Additionally, the extension provides syntax highlighting for files by reusing code from the JaCaMo4Code extension Krausburg [2020], improving code readability.

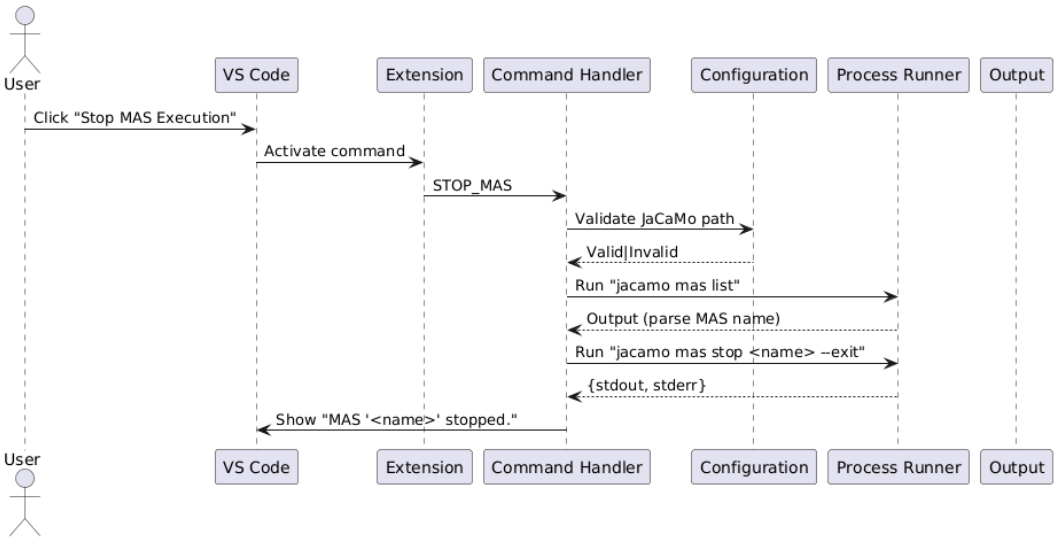
With development complete, the extension was packaged using the command *vsce package*, which generates the *.vsix* files ready for distribution. The extension was published on the official VSCode Marketplace, making it readily available for developers.



(a) Sequence diagram for the "Create JaCaMo App" command



(b) Sequence diagram for the "Run JaCaMo App" command



(c) Sequence diagram for the "Stop MAS Execution" command

Figure 5. Sequence diagrams of the extension's main functionalities

Figure 6. Artifact view from CarTaGo

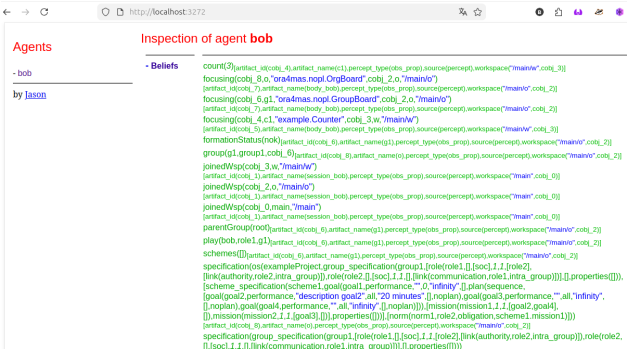


Figure 7. Agent view from Jason

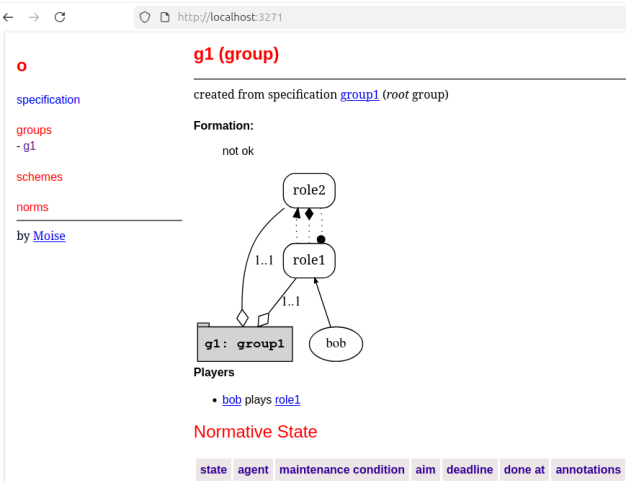


Figure 8. Organization view from Moise

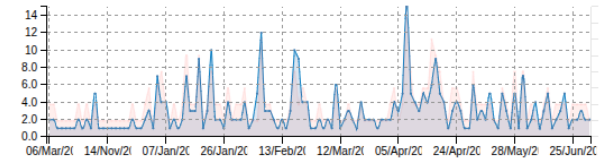
4.3 Reproducibility

The source code, a video demonstration, and instructions for installing and executing the extensions are available online[3].

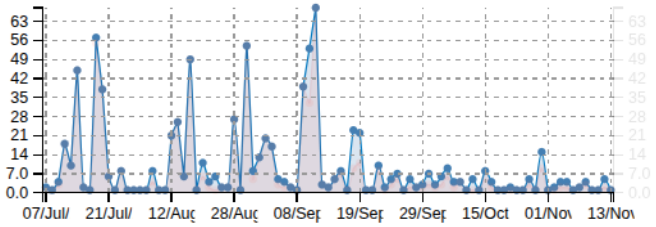
5 Results

This section covers the results obtained. We present the total downloads from the extension and its underlying infrastructure, jacamo-cli, before and after its release as a metric to evaluate the extension's success. We've decided to use this metric as it is readily available to us and, for software, it is an indication of its popularity among a greater range of users.

As previously stated in Section 4.1, the extension uses jacamo-cli as its underlying infrastructure on Linux, and a custom packaged executable on Windows. Since both packages are hosted on the CHON research group's web pages, we can monitor traffic generated by their downloads. This was



(a) Total downloads of jacamo-cli and jacamo-win before 7 July of 2025



(b) Total downloads of jacamo-cli and jacamo-win from 7 July of 2025 to 13 November of 2025

Figure 9. Comparison of total downloads of the extension's underlying infrastructure before and after its release

done by installing a reverse proxy on that server, a software that registers all incoming web requests, then filtering out all the requests except downloads for these files. The original data collected to compile these results are available online.

Figure 9a shows the downloads request frequency of the packages from 6 March 2024, the day jacamo-cli was first released, to 7 July 2025. The jacamo-win was released on 4th January 2025. The total number of download requests during this time period was 383, with a peak of 14 on a single day.

Next, in Figure 9b, we show the download requests starting from 7 July 2025 — the day the extension was released to the agent community for internal testing — to 13th November 2025. We excluded the requests after 13 November because new versions of jacamo-cli and jacamo-win were released, triggering updates on all machines that had it already installed. It is clear that, just from this small sampling period, it has generated interest, even in its initial testing format. The total number of download requests was 853, more than doubling the number of installs shown in Figure 9a over a shorter period, with a peak of 63 requests on a single day.

Since the official release on the VSCode extension store on 8 August 2025 to the time of this article's writing[4], it has been installed 150 times.

6 Conclusion

This article presented an extension for Visual Studio Code to support the creation and development of MAS using the JaCaMo framework. The extension use as its underlying infrastructure the jacamo-cli and jacamo-win, running their necessary commands automatically to execute the most common development operations: create an initial file structure for a new project, and start/stop a MAS's execution. Furthermore, the extension displays debugging information in VSCode's interface, eliminating the need to query logs for each tool in the framework.

We then demonstrate the extension's impact on JaCaMo's developers by analysing the number of downloads of jacamo-cli and jacamo-win before and after its creation, showing that it more than doubled over a shorter period. This suggests that the tool is helping to fill a void of proper tools to help the development of MAS's

[3] `https://papers.chon.group/REIC/jacamoExtension/`

[4] `http://jacamo-vscode.chon.group/`

References

Almeida Neto, M. R. d. and Lazarin, N. M. (2024). Uma extensão para o VSCode que utiliza o ChatGPT como ferramenta de apoio ao desenvolvimento de software seguro. In *XV Computer on the Beach (COTB'24)*, Balneário Camboriú, Brazil. UNIVALI. DOI: 10.14210/cotb.v15.p310-311.

Amaral, C. J. and Hübner, J. F. (2020). Jacamo-Web is on the Fly: An Interactive Multi-Agent System IDE. In *Engineering Multi-Agent Systems*, volume 12058, pages 246–255. Springer International Publishing, Cham. DOI: 10.1007/978-3-030-51417-4_13.

Boissier, O., Bordini, R. H., Hübner, J. F., and Ricci, A. (2019). Dimensions in programming multi-agent systems. *The Knowledge Engineering Review*, 34:e2. DOI: 10.1017/S026988891800005X.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2013). Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747–761. DOI: 10.1016/j.scico.2011.10.004.

Boissier, O., Hübner, J. F., and Ricci, A. (2016). The JaCaMo Framework. In *Social Coordination Frameworks for Social Technical Systems*, volume 30, pages 125–151. Springer International Publishing, Cham. Series Title: Law, Governance and Technology Series. DOI: 10.1007/978-3-319-33570-4_7.

Bordini, R. H., El Fallah Seghrouchni, A., Hindriks, K., Logan, B., and Ricci, A. (2020). Agent programming in the cognitive era. *Autonomous Agents and Multi-Agent Systems*, 34(2):37. DOI: 10.1007/s10458-020-09453-y.

Bordini, R. H. and Hübner, J. F. (2006). BDI Agent Programming in AgentSpeak Using Jason. In *Computational Logic in Multi-Agent Systems*, volume 3900, pages 143–164. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science. DOI: 10.1007/11750734_9.

Durelli, R. S., Durelli, V. H. S., Bettio, R. W., Dias, D. R. C., and Goldman, A. (2022). Divinator: A Visual Studio Code Extension to Source Code Summarization. In *10th Workshop on Software Visualization, Evolution and Maintenance (VEM 2022)*, pages 1–5, Online. SBC. DOI: 10.5753/vem.2022.226187.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2007). Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3-4):370–395. DOI: 10.1504/IJAOSE.2007.016266.

Krausburg, T. (2020). JaCaMo4Code - Visual Studio Marketplace. URL: `https://marketplace.visualstudio.com/items?itemName=tabajara-krausburg.jacamo4code`.

Lima, G., Siqueira, E., Medeiros, T., Pantoja, C., Lazarin, N., and Viterbo, J. (2025). A Modularized and Reusable Architecture for an Embedded MAS IDE:. In *ICEIS 2025*, pages 1034–1045, Porto, Portugal. SCITEPRESS. DOI: 10.5220/0013439600003929.

Liu, Y., Tantithamthavorn, C., and Li, L. (2025). Protect Your Secrets: Understanding and Measuring Data Exposure in VSCode Extensions. In *IEEE SANER 2025*, pages 551–562, Montreal, QC, Canada. IEEE. DOI: 10.1109/SANER64311.2025.00058.

Microsoft (2026). Extension API. Available at: `https://code.visualstudio.com/api`.

Neto, M. A., Lazarin, N., Freitas, B., and Castro, D. (2025). Proposal of a VSCode extension to facilitate the adoption of JaCaMo in Multi-Agent System development. In *Proceedings of the 19th Workshop-School on Agents, Environments, and Applications*, pages 79–86, Fortaleza, Brazil. SBC. DOI: 10.5753/wesaac.2025.37547.

Ricci, A., Viroli, M., and Omicini, A. (2007). CArtA gO: A Framework for Prototyping Artifact-Based Environments in MAS. In *Environments for Multi-Agent Systems III*, volume 4389, pages 67–86. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science. DOI: 10.1007/978-3-540-71103-2_4.

Siqueira, E., Ramos, G., Raboni, T., Pantoja, C., and Lazarin, N. (2024). Comparative Analysis of an IDE Prototype for Developing Embedded MAS. In *Proceedings of the 18th Workshop-School on Agents, Environments, and Applications*, pages 85–95, Brasília, Brazil. SBC. DOI: 10.5753/wesaac.2024.33458.

Souza De Jesus, V., Lazarin, N. M., Pantoja, C. E., Vaz Alves, G., Lima, G. R. A. D., and Viterbo, J. (2023). An IDE to Support the Development of Embedded Multi-Agent Systems. In *PAAMS 2023*, Cham. Springer. DOI: 10.1007/978-3-031-37616-0_29.

Tedeschi, S. (2021). AgentSpeak - Visual Studio Marketplace. Available at: `https://marketplace.visualstudio.com/items?itemName=Stex93.agentspeak`.

Tkampik (2018). code-mas2j - Visual Studio Marketplace. Available at: `https://marketplace.visualstudio.com/items?itemName=tkampik.code-mas2j`.

U473t8 (2019). VS Code JaCaMo - Visual Studio Marketplace. Available at: `https://marketplace.visualstudio.com/items?itemName=u473t8.code-jcm`.

Vahlbrock, T., Guddat, M., and Vierjahn, T. (2022). VSCode Migrate: Semi-Automatic Migrations for Low Coverage Projects. In *ICSME 2022*, Limassol, Cyprus. IEEE. DOI: 10.1109/ICSME55016.2022.00070.