Towards a Framework for Multi-Bot Collaboration

Alberto Mimbrero, José A. Parejo, Pablo Fernández, Miguel Romero-Arjona, Sergio Segura
SCORE Lab, I3US Institute, Universidad de Sevilla, Seville, Spain
mimbrero@us.es

***Abstract*—Software bots are rapidly proliferating, assisting developers with tasks such as coding, testing, modeling, and documentation. However, most research has focused on the use of isolated bots, overlooking a critical aspect to unlock their full potential: multi-bot collaboration. In this work-in-progress paper, we introduce the first prototype of Botica, a framework leveraging asynchronous APIs, messaging, and containerization to facilitate the development, deployment, and interaction of bots. A pilot study using Botica for REST API testing illustrates its effectiveness and the potential advantages of bot collaboration.**

***Index Terms*—software bots, multi-bot collaboration, orchestration, containerization, software engineering**

I. Introduction

The rise of software development bots—automated tools designed to assist developers with routine and repetitive tasks [1]—has significantly impacted software engineering practices. Popular bots such as GitHub Copilot [2] for code completion, Dependabot [3] for dependency management, and First Timers [4] for supporting new contributors to open-source projects exemplify how these tools streamline development workflows, supporting productivity. Many of these bots provide natural language interfaces, often referred to as *chatbots*, which integrate seamlessly into popular collaboration platforms such as Slack, MS Teams, and Discord. At the time of writing, the "Developer Tools" category in the Slack app repository lists over 500 apps (chatbots) supporting tasks such as end-to-end testing [5], code analysis [6] and vulnerability scanning [7].

Despite substantial advancements in software bot development, progress has primarily focused on isolated bots that address specific software engineering tasks, e.g., [8]–[10]. While this approach promotes cohesion, maintainability, and modularity, it constrains the utility of bots in tasks that require insights and data from different activities. Hence, the need for systems that facilitate the coordination and collaboration of bots is recognized as a challenge within the software engineering community [11]. For instance, in testing scenarios, bots responsible for test case generation, execution, and reporting would ideally communicate and collaborate to achieve optimal results (see running example). This demand for collaboration has driven the emergence of agentic systems, such as AutoGen [12] or CrewAI [13], which facilitate the integration of AI-enabled agents working together to tackle complex, high-level tasks, often leveraging large language models. However, these systems primarily target artificial intelligence (AI) applications and are not specifically designed to support the integration of general-purpose software bots.

In this work-in-progress paper, we introduce the first prototype of Botica, a framework designed to enable and optimize multi-bot collaboration. Leveraging asynchronous APIs, a messaging layer, and containerization, Botica allows for seamless interactions among bots, facilitating the development and deployment of a collaborative bot environment. We present the result of a pilot study using Botica in the context of REST API testing, showing the potential benefits of bot collaboration.

The remainder of the paper is structured as follows: in Section II we present a running example in the context of RESTful API testing. Next, Section III details Botica 's architecture. Section IV examines the pilot study evaluation and findings. Section V discusses related work, and Section VI explores the broader implications of enabling multi-bot collaboration in software engineering.

II. Running Example

Figure 1 shows a scenario of the use of bots for the automated testing of REST APIs, partially inspired by the work of Martin-Lopez et al. [14]. In their work, the authors launched a total of 228 bot instances that automatically generated and executed test cases in 13 industrial APIs for 15 days non-stop, resulting in over one million test cases. As a result, more than 250 bugs were identified, 65 of them confirmed, in the APIs of Amadeus, Foursquare, Yelp, and YouTube. Bots were implemented using the RESTest framework [15] as well as ad-hoc Python and shell scripts. The bots launched included, among others, the following types of bots:

- *Generator bots*. They are responsible for generating test cases for a given API operation, combining different test data (e.g., data dictionaries) and test case generation techniques (e.g., constraint-based testing).
- *Executor bots*. They run the test cases provided by the generator bots against the specified APIs at pre-configured time intervals.
- *Reporter bots*. They generate test reports and coverage reports based on the execution results.

Figure 1 also shows two other types of bots not used in [14], but that would make sense in this scenario, namely:

- *Oracle generator bots*. These bots, inspired by the work of Alonso et al. [16], are responsible for inferring invariants (i.e., potential test oracles) by analyzing the API specification, previous API requests, and their corresponding responses.
- *Chat bot*. This is in charge of keeping developers in the loop, for example, by confirming whether the observed

invariants are actual test oracles or not. Confirmed oracles could then be added as new assertions to future test cases.

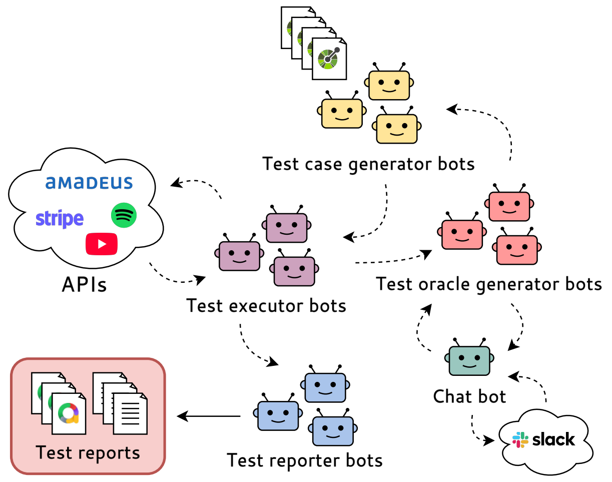

Fig. 1. Running example.

This example scenario illustrates several key challenges. First, synchronizing bots—often developed in different programming languages—is critical to maintaining an efficient workflow across test case generation, execution, reporting, and oracle detection. For instance, in [14], the authors chose to integrate test case generation and execution within a single bot type (referred to as a test bot) because separating and coordinating these tasks proved difficult. Another challenge lies in bot management, which includes tasks such as starting and stopping bots at the correct times, ensuring safe shutdowns without abruptly terminating processes, checking bots statuses, and properly cleaning up resources—processes that were partially manual or could not be done in [14]. Additionally, redundant tasks present a significant issue; isolated bots, unaware of their operating environment, may repeatedly perform the same task without adapting their behaviour. In [14], for instance, bots continued generating and executing test cases despite encountering API quota limit errors or API availability issues.

Martin-Lopez et al. [14] partially mitigated these limitations by using ad-hoc scripts, which manually dealt with background processes to manage the deployment of bots. However, while effective, their approach was difficult to generalize and presented severe scalability limitations, as well as requiring significant manual work. These challenges motivate our work.

III. Botica

In this section, we present Botica, a framework for multi-bot collaboration. Below, we present an overview of its architecture and design.

A. Architecture

Figure 2 shows the architecture of Botica. Each bot instance is deployed inside a container and shares the environment (e.g., network) with the other bots, as well as the internal message broker. There may be multiple instances of each bot type. The message broker facilitates communication between bots and connects them to the Botica *Director*, the central management program for the environment. This program runs on the host machine and is responsible for deploying the message broker and bots inside the containerization software, as well as configuring them within the environment. It also manages bots through a special set of internal control messages that enable orchestration of the environment and constitute the Botica protocol, such as checking bot statuses or requesting shutdowns. For example, a bot can register a *shutdown hook*, so it can postpone a shutdown from the Director if it is still running a task or writing into the disk.

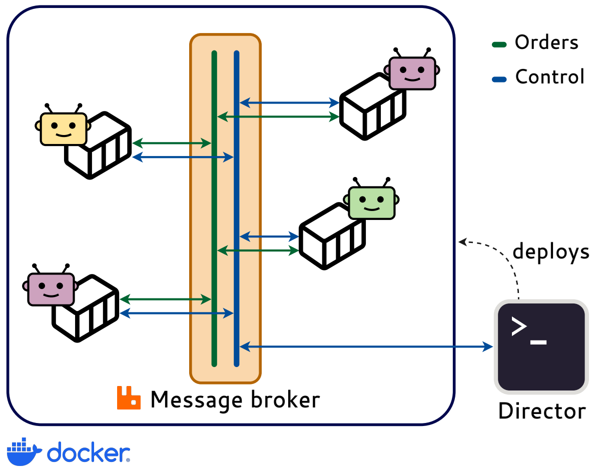

Fig. 2. Botica architecture.

B. Components

The tool is composed of the following main components:

Internal message broker. It facilitates communication both among the bots and between the bots and the Director. It is deployed and managed internally by the Director as a container within the same network as the bots. Bots can subscribe to orders published by other bots, which involves three main concepts: *key*, *order* and *message*—all specified when subscribing and publishing. The *key* is a broker-level property that determines which bots will receive the order and how they receive it. Each bot type subscribes to individual keys and selects a delivery strategy for each: either *distributed* (each new order is delivered to a single available instance of the bot type, enabling load balancing) or broadcast (new orders are delivered to every instance of the bot type). The *order* is a bot-level property used to distinguish between different actions. Finally, the *message* is the actual data sent between bots. This three-level architecture supports the creation of interaction protocols between bots, enabling the implementation of various collaboration and coordination patterns.

Director. It is responsible for deploying and managing both the message broker and the bots. It first reads the environment configuration file to gather the necessary queues and other messaging configurations from the bots. Then, using the container software API and daemon, it creates and starts the message broker container with the custom configuration for the environment. Once the message broker is ready, a connection is established and the bot containers are created and started. The Botica Director also manages bots through special control messages—referred to as packets—which are treated and implemented differently from standard orders, such as heartbeat checks or shutdown requests. The Botica Director is provided as an interactive CLI program, offering management capabilities for the user, such as information retrieval or shutdown commands.

Bots. Bots automatically establish a connection with the message broker on startup and integrate with the environment, by publishing orders and subscribing to orders from other bots, or scheduling proactive actions—code that runs independently without needing an external trigger, often at regular intervals. They are connected to the Director and listen and reply to control messages, such as the heartbeat checks. By default, a shared volume is provided to enable internal file exchange among bots, though they can also be configured to mount files and directories from the host file system.

C. Tooling

Botica has been developed across three GitHub repositories, including the main program, Botica Director, and two libraries for bot development [17]–[19]. Botica Director, written in Java 11, comprises 3.5K lines of code (LoC). Libraries are available for Java (530 LoC) and Node (+700 LoC). Both libraries facilitate the development of bots that can be easily integrated into the platform, including templates to streamline project setup and build processes.

The entire environment is set up through a configuration file, where the different bots are declared (see sample extract in Listing 1). This file specifies their container configuration (e.g., the container image in line 3, files/directories to mount on the container file system in lines 4-6), the keys and delivery strategies of the orders to subscribe to (lines 30-31), and the bot instances (lines 15-20), among others. There may be multiple instances of each bot type, each with a unique identifier (key in line 15) and specific environment variables (lines 16-17). A template repository is also provided to facilitate the environment creation.

```
1  bots:
2    generator:
3      image: "botica-bot-restest-generator"
4      mount:
5        - source: "./src"
6          target: "/app/src"
7        - ...
8      publish:
9        key: "test_execution"
10       order: "execute_test_cases"
11     lifecycle:
12       type: proactive
13       period: 60
14     instances:
15       generator-omdb-r-cbt-custom:
16         environment:
17           - USER_CONFIG_PATH=...
18       generator-omdb-r-ft:
19         environment:
20           - USER_CONFIG_PATH=...
21       ...
22
23   executor:
24     image: "botica-bot-restest-executor"
25     mount: { ... }
26     publish:
27       key: "test_reporting"
28       order: "generate_test_report"
29     subscribe:
30       - key: "test_execution"
31         strategy: distributed
32     lifecycle:
33       type: reactive
34       order: "execute_test_cases"
35     instances:
36       executor-1: { }
37       ...
38
39   reporter:
40     image: "botica-bot-restest-reporter"
41     mount: { ... }
42     subscribe:
43       - key: "test_reporting"
44         strategy: distributed
45     lifecycle:
46       type: reactive
47       order: "generate_test_report"
48     instances:
49       reporter-1: { }
50       ...
```

Listing 1. Extract of an example environment configuration file.

IV. Preliminary evaluation

As a pilot study, we implemented an instance of the running example using Botica. The final architecture consisted of 21 generator bots, 9 executor bots, and 5 reporter bots, all implemented using RESTest [14] and integrated into Botica through its Java libraries. These bots were configured to autonomously generate and execute test cases on the web APIs of Marvel (character endpoints) [20], Stripe (product endpoints) [21], OMDb (search endpoint) [22], and Rest-countries (all endpoints) [23]—33 API endpoints in total. Over a 24-hour period, test cases were generated at intervals between 100 and 345 seconds. After each generation batch, executors were triggered by notification messages to initiate test execution. Reporter bots subsequently generated results dashboards upon receiving messages from the executors. The implementation took 401 LoC, with 202 lines dedicated to the environment configuration file (see excerpt in Listing 1) and 199 lines written in Java, excluding RESTest-specific code. The messages exchanged through Botica were collected for later analysis. Source code and data are available at [24].

Table I summarizes the key metrics and outcomes of our study. Overall, 558,768 test cases were generated across 9,466 batches and successfully executed, resulting in the exchange of over 18.9K messages among the bots. Note that not all generated test cases were fully executed, as some were still being processed or remained in queue when the bots were stopped. This collaborative approach helped address, either fully or partially, the challenges outlined in Section II. Bot synchronization allowed the creation of a fully automatic workflow, with test generation and execution separated into two different bots. Generated test batches were executed efficiently, leveraging load balancing among executor bots. All bots were automatically started and stopped without incident. To streamline data handling, generator and executor bots exchanged files via the shared volume provided, while output files were stored in a directory mounted from the host OS file system. These preliminary results show the potential of Botica in supporting effective development, deployment, and collaboration of multi-bot applications for software engineering.

TABLE I
EXPERIMENTAL DATA

Metric	**Value**
Number of APIs	4
Number of endpoints	33
Marvel	6
Stripe	5
OMDb	1
Restcountries	21
Number of bots	35
Generators	21
Executors	9
Reporters	5
Number of messages (orders)	18,917
Disk usage	46 GiB
Number of generated tests	558,768
Number of executed tests	555,717
Passed	432,775
Failed	122,589
Broken	353

V. Related work

Current research largely focuses on isolated bots and specific software engineering tasks, such as recommending the most suited developers for branch merges [8], intelligent software refactoring [9], or proposing fixes for static analysis warnings [10]. While human-machine interaction has been extensively studied [25]–[29], the field of bot-to-bot interaction remains largely underexplored [11] and it has only been addressed in specific domains such as the Wikipedia bot ecosystem [30] or the botnets threats in security [31]. Similarly, different tools have been developed to facilitate the development and deployment of individual bots (e.g., [32], [33]), yet they lack mechanisms to support collaborative bot interaction—the goal of Botica.

Frameworks such as Atomic Agents [34], AutoGen [12] or CrewAI [13] facilitate the creation of autonomous multi-agent AI systems. However, these frameworks are dependent on LLM-based agents and are not designed for other purposes. By contrast, Botica is a general-purpose tool for bot deployment, lifecycle management, and coordination. It provides asynchronous messaging capabilities to create interaction protocols, supporting a variety of collaboration and coordination patterns, including load balancing and information broadcasting.

VI. Conclusions and future work

This paper presents Botica, a work-in-progress framework for multi-bot collaboration in software development environments. Botica leverages containerization and asynchronous communications to facilitate interaction and coordination among bots. The framework allows users to create a range of customizable bots, providing a flexible infrastructure that has proven useful in automating REST API testing tasks.

In future work, we aim to expand Botica with a broader set of features. Planned enhancements include advanced management capabilities, such as dynamically scaling the number of bot instances based on workload demands. We are also developing a Complex Event Processing (CEP) module that will enable bots to adjust their behaviours based on patterns detected in message flows. For instance, this could allow generator bots to pause for a specified time upon detecting quota limits or API availability issues, or trigger warnings via a chatbot if an unusual number of test failures is observed. Finally, we intend to conduct large-scale evaluations across various application scenarios, incorporating diverse bot types (including AI-driven bots) and collaboration patterns.

Acknowledgments

This work is a result of grant PID2021-126227NB-C22, funded by MCIN/AEI /10.13039/501100011033/ERDF/EU; and grant TED2021-131023B-C21, funded by MCIN/AEI/10.13039/501100011033 and by European Union “NextGenerationEU/PRTR”.

References

[1] L. Erlenhov, F. G. d. O. Neto, and P. Leitner, “An empirical study of bots in software development: characteristics and challenges from a practitioner’s perspective,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 445–455. [Online]. Available: https://doi.org/10.1145/3368089.3409680

[2] “GitHub Copilot,” https://github.com/features/copilot, accessed November 2024.

[3] “GitHub Dependabot,” https://github.com/dependabot, accessed November 2024.

[4] “First Timers,” https://github.com/first-timers, accessed November 2024.

[5] “Cypress,” https://slack.com/marketplace/A010PD913CG, accessed November 2024.

[6] “CodeFactor,” https://slack.com/marketplace/A2U8M78RH, accessed November 2024.

[7] “Intruder,” https://slack.com/marketplace/AB2HWP5KL, accessed November 2024.

[8] C. Costa, J. Figueiredo, L. Murta, and A. Sarma, “Tipmerge: recommending experts for integrating changes across branches,” in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 523–534. [Online]. Available: https://doi.org/10.1145/2950290.2950339

[9] V. Alizadeh, M. A. Ouali, M. Kessentini, and M. Chater, “RefBot: Intelligent Software Refactoring Bot,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 823–834.

[10] D. Şerban, B. Golsteijn, R. Holdorp, and A. Serebrenik, “SAW-BOT: Proposing Fixes for Static Analysis Warnings with GitHub Suggestions,” in *Proceedings - 2021 IEEE/ACM 3rd International Workshop on Bots in Software Engineering, BotSE 2021*. United States: IEEE Computer Society, Jun. 2021, pp. 26–30.

[11] J. Cabot, “The Present and Future of Bots in Software Engineering,” https://livablesoftware.com/present-future-bots-software-engineering/, accessed November 2024.

[12] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations," in *Conference on Language Modeling*, 2024. [Online]. Available: https://aka.ms/autogen-pdf

[13] "CrewAI," https://www.crewai.com, accessed November 2024.

[14] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "Online testing of restful apis: Promises and challenges," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 408–420.

[15] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "RESTest: automated black-box testing of RESTful web APIs," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 682–685. [Online]. Available: https://doi.org/10.1145/3460319.3469082

[16] J. C. Alonso, S. Segura, and A. Ruiz-Cortés, "Agora: Automated generation of test oracles for rest apis," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 1018–1030. [Online]. Available: https://doi.org/10.1145/3597926.3598114

[17] "Botica repository on GitHub," https://github.com/isa-group/botica, accessed November 2024.

[18] "Botica Java library repository on GitHub," https://github.com/isa-group/botica-lib-java, accessed November 2024.

[19] "Botica Node library repository on GitHub," https://github.com/isa-group/botica-lib-node, accessed November 2024.

[20] "Marvel API," https://developer.marvel.com/docs, accessed November 2024.

[21] "Stripe Products API," https://docs.stripe.com/api/products, accessed November 2024.

[22] "OMDb API," https://www.omdbapi.com/, accessed November 2024.

[23] "REST Countries API," https://restcountries.com/, accessed November 2024.

[24] "Supplementary material," https://github.com/isa-group/botica-infrastructure-restest/tree/botse-paper-evaluation, accessed November 2024.

[25] J. Zamora, "I'm sorry, dave, i'm afraid i can't do that: Chatbot perception and expectations," in *Proceedings of the 5th International Conference on Human Agent Interaction*, ser. HAI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 253–260. [Online]. Available: https://doi.org/10.1145/3125739.3125766

[26] D. Liu, M. J. Smith, and K. Veeramachaneni, "Understanding user-bot interactions for small-scale automation in open-source development," in *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–8. [Online]. Available: https://doi.org/10.1145/3334480.3382998

[27] A. P. Chaves and M. A. Gerosa, "How should my chatbot interact? a survey on social characteristics in human–chatbot interaction design," *International Journal of Human–Computer Interaction*, vol. 37, no. 8, pp. 729–758, 2021. [Online]. Available: https://doi.org/10.1080/10447318.2020.1841438

[28] A. P. Chaves, J. Egbert, T. Hocking, E. Doerry, and M. A. Gerosa, "Chatbots language design: The influence of language variation on user experience with tourist assistant chatbots," *ACM Trans. Comput.-Hum. Interact.*, vol. 29, no. 2, Jan. 2022. [Online]. Available: https://doi.org/10.1145/3487193

[29] L. A. Tran, B. Hensen, R. Klamma, and S. Chantaraskul, "Privacy and security in mixed reality learning environments by input and user/bot interaction protection," in *Proceedings of the 2022 4th Asia Pacific Information Technology Conference*, ser. APIT '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 63–71. [Online]. Available: https://doi.org/10.1145/3512353.3512363

[30] R. S. Geiger and A. Halfaker, "Operationalizing conflict and cooperation between automated software agents in wikipedia: A replication and expansion of 'even good bots fight'," *Proc. ACM Hum.-Comput. Interact.*, vol. 1, no. CSCW, Dec. 2017. [Online]. Available: https://doi.org/10.1145/3134684

[31] J. M. F. M. Augusto Santos, Michele Nogueira, "A stochastic adaptive model to explore mobile botnet dynamics," *IEEE Communications Society*, 2016. [Online]. Available: https://doi.org/10.1109/lcomm.2016.2637367

[32] G. Daniel, J. Cabot, L. Deruelle, and M. Derras, "Xatkit: A Multimodal Low-Code Chatbot Development Framework," *IEEE Access*, vol. 8, pp. 15 332–15 346, 2020.

[33] A. Ait, J. L. C. Izquierdo, and J. Cabot, "A tool for the definition and deployment of platform-independent bots on open source projects," *Proceedings of the 16th ACM SIGPLAN International Conference on Software Language Engineering*, 2023.

[34] "Atomic Agents," https://gjmcn.github.io/atomic-agents, accessed November 2024.