Concurrency Model of BDI Programming Frameworks: Why Should We Control It?

Extended Abstract

Martina Baiardi
University of Bologna
Cesena, Italy
m.baiardi@unibo.it

Samuele Burattini
University of Bologna
Cesena, Italy
samuele.burattini@unibo.it

Giovanni Ciatto
University of Bologna
Cesena, Italy
giovanni.ciatto@unibo.it

Danilo Pianini
University of Bologna
Cesena, Italy
danilo.pianini@unibo.it

Andrea Omicini
University of Bologna
Cesena, Italy
andrea.omicini@unibo.it

Alessandro Ricci
University of Bologna
Cesena, Italy
a.ricci@unibo.it

ABSTRACT

We provide a taxonomy of concurrency models for BDI frameworks, elicited by analysing state-of-the-art technologies, and aimed at helping both BDI designers and developers in making informed decisions. Comparison among BDI technologies w.r.t. concurrency models reveals heterogeneous support, and low customisability.

KEYWORDS

Agent-Oriented Programming; Concurrency; BDI Agents; Threads

ACM Reference Format:
Martina Baiardi, Samuele Burattini, Giovanni Ciatto, Danilo Pianini, Andrea Omicini, and Alessandro Ricci. 2024. Concurrency Model of BDI Programming Frameworks: Why Should We Control It?: Extended Abstract. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 3 pages.

1 INTRODUCTION

By construction, Belief-Desire-Intention (BDI) agents are able to carry on multiple intentions at any given time [13], and many research and software-development efforts have been devoted to the definition of BDI architectures and programming languages giving precise semantics to the *concurrent* execution of such intentions [3].

As computational entities, agents are autonomous if they encapsulate their own *control flow* [9]. Control-flow encapsulation is commonly referred to as *computational* autonomy [10], and it is considered a necessary pre-requisite for autonomy in software agents. On mainstream programming platforms, autonomy is achieved by mapping each agent onto ad-hoc control-flow-related primitives, such as threads, processes, or event loops; providing different trade-offs in terms of efficiency, determinism, and reproducibility of the Multi-Agent Systems (MASs) built on top of them.

Adopting the right concurrency model is essential, as it deeply impacts many aspects of the agent programming framework and the dynamics of all MASs leveraging it. In particular, the concurrency model affects whether, and to what extent, multiple agents can run at the same time, impacting performance and efficiency of MASs; opposedly, parallelism as well as the determinism of the overall MAS dynamics, is a strict requirement in applications requiring reproducibility, such as multi-agent based simulation [2].

Dealing with concurrency is commonly acknowledged as error-prone and challenging. Thus, mainstream programming platforms provide dedicated libraries and language constructs shielding developers from the intricacies of concurrency. Similarly, Agent-Oriented Programming (AOP) tools and frameworks come with their own concurrency model, often hidden under the hood.

Although hiding concurrency details is helpful to reduce the learning curve, experienced developers unaware of the nuances of the framework they are relying upon may have reduced control over the execution of their MASs and the trade-offs that come with it. This is particularly true for BDI agent technologies, where the semantics of intention scheduling can be realised in many different ways. Arguably, BDI technologies should rather let MASs developers choose or configure the concurrency model of their systems, in order to tune the execution of the MAS to the specific needs of their application and execution environment.

In this study, we provide a taxonomy of the concurrency models available for BDI agent technologies, and classify several widely used frameworks accordingly. The current literature on BDI agents and concurrency (e.g., [5, 6, 14–16]) focuses on agents' *internal* concurrency—roughly, how control loops interleave *intentions*. Conversely, we focus on *external* concurrency, i.e., the way multiple agents are mapped onto the underlying (threads, processes, event loops, executors) concurrency abstractions. Finally, we elaborate on the importance of customisable MASs execution, recommending framework designers to promote a neat separation of the MAS specification from its actual runtime concurrency model.

2 CONCURRENCY MODELS FOR BDI SYSTEMS

Most modern programming languages support concurrency through one or more of the following abstractions: *(i) threads*, the basic units of concurrency [7], i.e., the executors of sequential programs; *(ii) processes*, i.e., containers of threads sharing memory; *(iii) event loops*, i.e., individual threads carrying out sequential programs

This work is licensed under a Creative Commons Attribution International 4.0 License.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

(tasks) enqueued by users; and *(iv) executors*, i.e., event loops with a possibly configurable unbound thread count.

As introduced, *external* concurrency models map MASs concepts onto these abstractions; concretely, they differ in the way the control loop of each agent is mapped onto them. Different model provide different granularity:

One-Agent-One-Thread (1A1T) – each agent is mapped onto a single thread, which is responsible for executing its entire control loop. The control over the of MAS execution is abysmal: essentially, developers are delegating control to the Operating System (OS). Determinism is compromised as well, as the OS scheduler may interleave the execution of different agents arbitrarily. The amount of threads in the MAS is unbound, which may lead to relevant overhead when the number of active agents (threads) is far greater than the amount of hardware cores/processors.

All-Agents-One-Thread (AA1T) – the whole MAS is executed on a single thread that internally schedules all agents' execution in a custom way, following some (usually cooperative) scheduling policy. This model enables fully *deterministic* execution of MASs, as parallelism is absent. Hence, it is desirable when reproducibility is a concern, such as in testing or reproducible simulations, but it is unsuitable for performance-critical scenarios, when hardware capable of parallel computation is available.

All-Agents-One-Event-Loop (AA1EL) – the whole MAS is executed on a single event loop, which internally schedules all agents' execution with a first-in-first-out queue of tasks, guaranteeing fairness by design. AA1EL is equivalent (also in terms of determinism and performance) to an AA1T strategy with fair scheduling (e.g., round-robin).

All-Agents-One-Executor (AA1E) – each agent's activity is enqueued as task on a shared *executor*. However, tasks are executed concurrently (possibly, in parallel). AA1E is conceptually equivalent to 1A1T, yet technologically preferable as, by controlling the executor's thread count, provides finer control on the degree of parallelism. Two specialisations of this model are possible, depending on whether the alive thread count changes with time: fixed thread pools and variable thread pools.

Further models can be obtained by (possibly hierarchical) combinations of the aforementioned ones, obtaining diverse flexibility/-controllability trade-offs. For instance, considering that event loops, executors, and threads are hosted into processes, we can think of:

One-Agent-One-Process (1A1P) – each agent is a process using threads, executors, or event loops for internal concurrency.

3 ANALYSIS OF BDI TECHNOLOGIES

We analyse a selection of open-source and actively maintained BDI programming technologies to inspect their external concurrency model(s). We focus on Jason [3], Astra [4], JaKtA [1], Phidias [8], Spade-BDI [11], Jadex [12]. In our analysis, for each BDI technology, we combine two approaches: we first run a benchmark to reveal how many threads are involved in a MAS execution; then, we inspect the source code and documentation to understand which concurrency abstractions are used, and to what extent they are customisable.

Table 1 summarises the results of our analysis including the 1A1P model, which is the basis for agents not sharing memory, hence, potentially distributable.

Table 1: BDI technologies and concurrency models. Meaning of symbols: "✓" – supported out of the box; "∗" – supports customizations; "∼" – we were unable to conclusively confirm or rule out support.

Model → Tech. ↓	1A1T	AA1T	AA1EL	AA1E fixed	AA1E variable	1A1P
Jason [3]	✓	✓∗	✓	✓	✓∗	✓∗
Astra [4]	✓∗	✓∗	✓	✓	✓	∼
JaKtA [1]	✓	✓	✓	✓	✓	✓∗
Phidias [8]	✓	×	×	×	×	✓
Spade-BDI [11]	×	×	✓	×	×	✓
Jadex [12]	×	✓	×	×	✓	×

4 DISCUSSION AND CONCLUSION

The concurrency model is a paramount dimension to consider when designing or using a (BDI) MAS technology. Generally, choice is desirable, as different applications and execution environments may benefit from different concurrency models.

From an application development perspective, the concurrency models impact primarily reproducibility and performance. Reproducibility requires determinism (especially when testing), supported by AA1T; sheer performance is usually better with parallel models like 1A1T or, preferably, AA1E. Some scenarios may be better tackled through custom concurrency models, hence, we recommend BDI technology designers to provide dedicated APIs.

We argue that flexibility in the choice of concurrency models is a central feature for BDI technologies Thus, we recommend considering them early in BDI framework design: adopting a specific concurrency model early on may complicate or prevent changing it later. When support for multiple (customisable) concurrency models is not feasible, early analysis can still prove beneficial. For instance, despite being conceptually akin, AA1E is preferable over 1A1T, as the former supports controlling the overall thread count.

Careful design of the BDI framework architecture is essential to ensure separation between the MAS specification and its actual runtime concurrency model: the former should be written once, and the latter should be selected as late as possible (ideally, at application launch). Flexibility enables: *(i)* controlling reproducibility for debugging or simulation, *(ii)* maximising performance in production, *(iii)* comparing and selecting the best model for the scenario at hand.

Summarising, external concurrency of BDI agents is paramount in MAS engineering, Yet, we believe further investigation is needed to provide a general concurrency blueprint for BDI technologies.

ACKNOWLEDGMENTS

This work has been partially supported by: *(i)* "WOOD4.0", Emilia-Romagna project, art. 6 L.R. N. 14/2014, call 2022; *(ii)* "FAIR", (PNRR, M4C2, Investimento 1.3, Spoke 8, P.E. PE00000013); *(iii)* "2023 PhD scholarship co-funded by NextGenerationEU and AUSL Romagna" (PNRR M4C2, Investimento 3.3, D.M. 352/2022); and *(iv)* "ENGINES", (Italian MUR PRIN 2022, grant 20229ZXBZM).

REFERENCES

[1] Martina Baiardi, Samuele Burattini, Giovanni Ciatto, and Danilo Pianini. 2023. JaKtA: BDI Agent-Oriented Programming in Pure Kotlin. In *Multi-Agent Systems - 20th European Conference, EUMAS 2023, Naples, Italy, September 14-15, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 14282)*, Vadim Malvone and Aniello Murano (Eds.). Springer, 49–65. https://doi.org/10.1007/978-3-031-43264-4_4

[2] Stefania Bandini, Sara Manzoni, and Giuseppe Vizzari. 2009. Agent Based Modeling and Simulation: An Informatics Perspective. *J. Artif. Soc. Soc. Simul.* 12, 4 (2009). http://jasss.soc.surrey.ac.uk/12/4/4.html

[3] Rafael H. Bordini, Jomi F. Hübner, and Michael J. Wooldridge. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason.* John Wiley & Sons, Ltd. XVII, 292 pages. http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470029005.html

[4] Rem W. Collier, Sean Edward Russell, and David Lillis. 2015. Reflecting on Agent Programming with AgentSpeak(L). In *PRIMA 2015: Principles and Practice of Multi-Agent Systems*. Springer International Publishing, 351–366. https://doi.org/10.1007/978-3-319-25524-8_22

[5] Lavindra de Silva. 2020. An Operational Semantics for True Concurrency in BDI Agent Systems. In *AAAI Conference on Artificial Intelligence.* https://api.semanticscholar.org/CorpusID:214310595

[6] Lavindra de Silva, Felipe Meneguzzi, and Brian Logan. 2020. BDI Agent Architectures: A Survey. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, Christian Bessiere (Ed.). ijcai.org, 4914–4921. https://doi.org/10.24963/ijcai.2020/684

[7] E. W. Dijkstra. 1965. Solution of a Problem in Concurrent Programming Control. *Commun. ACM* 8, 9 (sep 1965), 569. https://doi.org/10.1145/365559.365617

[8] Fabio D'Urso, Carmelo Fabio Longo, and Corrado Santoro. 2019. Programming Intelligent IoT Systems with a Python-based Declarative Tool *(CEUR Workshop Proceedings, Vol. 2502)*, Claudio Savaglio, Giancarlo Fortino, Giovanni Ciatto, and Andrea Omicini (Eds.). CEUR-WS.org, 68–81. https://ceur-ws.org/Vol-2502/paper5.pdf

[9] James J. Odell. 2002. Objects and Agents Compared. *Journal of Object Technology* 1, 1 (May–June 2002), 41–53. http://www.jot.fm/issues/issue_2002_05/column4

[10] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. 2008. Artifacts in the A&A Meta-Model for Multi-Agent Systems. *Autonomous Agents and Multi-Agent Systems* 17, 3 (Dec. 2008), 432–456. https://doi.org/10.1007/s10458-008-9053-x Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.

[11] Javier Palanca, Jaime Andres Rincon, Carlos Carrascosa, Vicente Julián, and Andrés Terrasa. 2022. A Flexible Agent Architecture in SPADE *(Lecture Notes in Computer Science, Vol. 13616)*. Springer, 320–331. https://doi.org/10.1007/978-3-031-18192-4_26

[12] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. 2005. Jadex: A BDI reasoning engine. 149–174. https://doi.org/10.1007/0-387-26350-0_6

[13] Anand S. Rao. 1996. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Agents Breaking Away*, Walter Van de Velde and John W. Perram (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 42–55. https://doi.org/10.1007/BFb0031845

[14] Alessandro Ricci and Andrea Santi. 2013. Concurrent Object-Oriented Programming with Agent-Oriented Abstractions: The ALOO Approach. In *2013 Workshop on Programming Based on Actors, Agents, and Decentralized Control (AGERE! 2013)* (Indianapolis, Indiana, USA). ACM, New York, NY, USA, 127–138. https://doi.org/10.1145/2541329.2541333

[15] Alessandro Ricci and Andrea Santi. 2013. From Actors and Concurrent Objects to Agent-Oriented Programming in SIMPAL. In *Concurrent Objects and Beyond – Festschrift in Honor of Akinori Yonezawa*, Gul Agha, Atsushi Igarashi, Naoki Kobayashi, Hidehiko Masuhara, Satoshi Matsuoka, Etsuya Shibayama, and Kenjiro Taura (Eds.). Springer.

[16] Maicon R. Zatelli, Alessandro Ricci, and Jomi F. Hübner. 2015. Evaluating Different Concurrency Configurations for Executing Multi-Agent Systems. In *Engineering Multi-Agent Systems*, Matteo Baldoni, Luciano Baresi, and Mehdi Dastani (Eds.). Springer International Publishing, Cham, 212–230.