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

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

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; oppositely, 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...
We analyse a selection of open-source and actively maintained BDI technologies to inspect their external concurrency models. 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.

### 3 ANALYSIS OF BDI TECHNOLOGIES

We analyse a selection of open-source and actively maintained BDI 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.

### 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 1A1T; 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 20229ZXBMZ).
REFERENCES


