Towards a Visual Programming Framework for BDI Agents

Helena Rentschler[1], Gabriel G. Neres[1], Rafael C. Cardoso[2], André Pinz Borges[1], Gleifer Vaz Alves[1]

Federal University of Technology - Paraná (UTFPR)
Ponta Grossa - PR - Brazil

Department of Computing Science
University of Aberdeen
Aberdeen – UK

{helenarentschler,neres}@alunos.utfpr.edu.br

{apborges,gleifer}@utfpr.edu.br

rafael.cardoso@abdn.ac.uk

Abstract. *Visual programming frameworks can lower the barrier to entry for agent-based development by allowing users to design agent behaviours through graphical abstractions rather than code. However, it remains unclear which agent architectures and visual programming paradigms have been most commonly explored in this area. To address this gap, this paper presents a systematic mapping study of visual programming frameworks for agent-based development, categorising relevant studies by agent architecture and visual programming paradigm. The results show a predominance of reactive agents and flow-based models, with most frameworks targeting computer science laypeople and robotics applications. Based on these findings, the paper also presents a prototype visual programming framework for Belief-Desire-Intention agents.*

1. Introduction

Users unfamiliar with programming who are interested in learning Multi-Agent Systems (MAS) may find the Integrated Development Environments (IDEs) commonly used by Python programmers discouraging, especially because of their initial complexity. This difficulty is further increased by the intrinsic complexity of working with a sequential programming language such as Python. Another barrier lies in the characteristics of the MAS modeling paradigm itself, including the different forms of agency it supports, such as reactive, Belief-Desire-Intention (BDI), and goal-oriented agents.

Reactive agents are systems that operate by directly mapping environmental stimuli into actions, without relying on complex internal representations such as beliefs, goals, or plans [Wooldridge 2009]. BDI agents explicitly represent beliefs, desires (goals), and intentions (plans) to guide decision-making [Bratman et al. 1988]. Goal-oriented agents focus primarily on specifying and achieving goals, often abstracting the internal reasoning process [Cohen and Levesque 1990]. Hybrid architectures combine elements of two or more of these approaches.

The BDI paradigm, in particular, requires a high level of abstraction, especially from learners with little or no background in computing. These learners must understand

not only programming concepts but also agent-oriented concepts such as beliefs, goals, intentions, plans, and decision-making cycles. This creates a significant barrier to using intelligent agent tools. To make agent programming more accessible to this audience, a visual programming paradigm is needed, allowing users to model agent behaviour through graphical elements rather than conventional source code.

Visual programming paradigms can be divided into four main types: block-based, flow-based, state machine, and node graph. The latter three may be regarded as subcategories of the diagram-based paradigm. Block-based languages allow users to drag and drop blocks that represent commands; these blocks snap together, preventing many syntax errors [Kuhail et al. 2021]. Flow-based programming models computation as a sequence of operations connected by data or control flow, making it well-suited to representing processes, pipelines, and event-driven systems. State machine paradigms describe system behavior in terms of states and transitions triggered by events or conditions, and are commonly used in control systems and reactive agent modeling [Harel 1987]. In node graph representations, nodes are connected by edges, with each node representing a component or operation and each edge representing a relationship or dependency.

Bringing together visual programming and BDI agents raises the need to understand which tools already exist for coding multi-agent systems through visual paradigms. To address this, we conducted a systematic mapping to identify and classify the available tools. The results reveal that only a few tools adopt visual programming for this purpose, indicating a gap in the research area.

The remainder of this paper is organized as follows: Section 2 details the systematic mapping process and its results. Section 3 introduces a prototype of our proposed concept using the Multi-Agent System Framework for Python (MASPY) [Mellado et al. 2026], a Python-based framework for developing MAS. Finally, Section 4 presents our conclusions and future work.

2. Systematic Mapping

This study aims to identify research gaps in visual programming frameworks for developing autonomous agents. The Kitchenham and Charters [Kitchenham and Charters 2007] methodology for performing Systematic Literature Reviews in Software Engineering was applied with the support of the Parsifal tool[1].

2.1. Research Questions

The research questions were defined as follows:

- Q1 Which visual programming frameworks for agent-oriented systems have been reported in the literature, and what is their prevalence?
- Q2 How do these frameworks classify in terms of visual paradigm and agent architecture?
- Q3 What is the end-user profile and target application domain for these tools?
- Q4 How does user interaction occur, and what are the main programming difficulties reported?
- Q5 What is the degree of freedom for customising and extending agent behaviours?

[1] `https://parsif.al` (Accessed: 20 June 2026)

The research questions were motivated by the development of a visual programming environment for MASPY. Thus, the mapping examines existing approaches, interaction mechanisms, target users, limitations, and extension strategies to support evidence-based design decisions for visual BDI agent programming.

2.2. Search

The systematic review began by defining a set of keywords and synonyms, which were then used to construct the following search string:

("Agent-Based Programming" OR "Agent-Oriented Programming" OR "Intelligent Agents" OR "BDI Agents" OR "Belief-Desire-Intention Agents" OR "Multi-Agent Systems" OR "multi-agent") AND ("Block-based Programming" OR "Box-and-Wire Programming" OR "Flow-based Programming" OR "Form-Based Programming" OR "State Machine Programming" OR "visual programming" OR "Low-code" OR "No-Code" OR "Visual Programming Languages" OR "end-user programming" OR "model-driven engineering")

The first left operand of the AND statement targets the study population, focusing on the core domain of MAS. The second targets the intervention, including visual programming paradigms and accessible development approaches. After defining the search strings, they were applied in three different sources: ACM Digital Library, IEEE Xplore, and Scopus. Both ACM and IEEE constrain their papers to the Computer Science and Engineering fields of study, respectively, serving as a pre-filter aligned with the mapping's proposal. As for Scopus, it encompasses a broader set of papers, including those that may be retrieved by less technology-specific keywords.

While executing the search string, the search was restricted to studies published between 2021 and 2026 using the publication-year filters available in the sources.

Figure 1 describes the methods used to retrieve and filter papers throughout the search phase: first, the application of the search string, then the exclusion and inclusion criteria, and finally, the reporting completeness and data-extractability assessment. The diagram is adapted from the PRISMA methodology[2].

A reporting-completeness and data-extractability assessment was applied because the classification required information about the visual paradigm, agent-oriented concepts, intended users, application domain, and interaction mechanisms. The assessment verified whether the paper reported sufficient information to support reliable data extraction and classification.

For this purpose, 6 adjunct quality criteria questions present in table 1, each limited to three possible answers: "yes", "partially" and "no". The scores attributed to each of them being, respectively, 1, 0.5 and 0. Thus, the maximum score a paper could receive was 6, and the inclusion threshold was fixed at 3.5, the minimum possible score above half, considering the increments of 0.5. This threshold ensured that an included study demonstrated at least partial evidence across a majority of the quality dimensions required for data extraction.

Figure 1 shows that 312 papers did not satisfy the acceptance criteria and three

[2] `https://www.prisma-statement.org/` (Accessed: 20 June 2026)

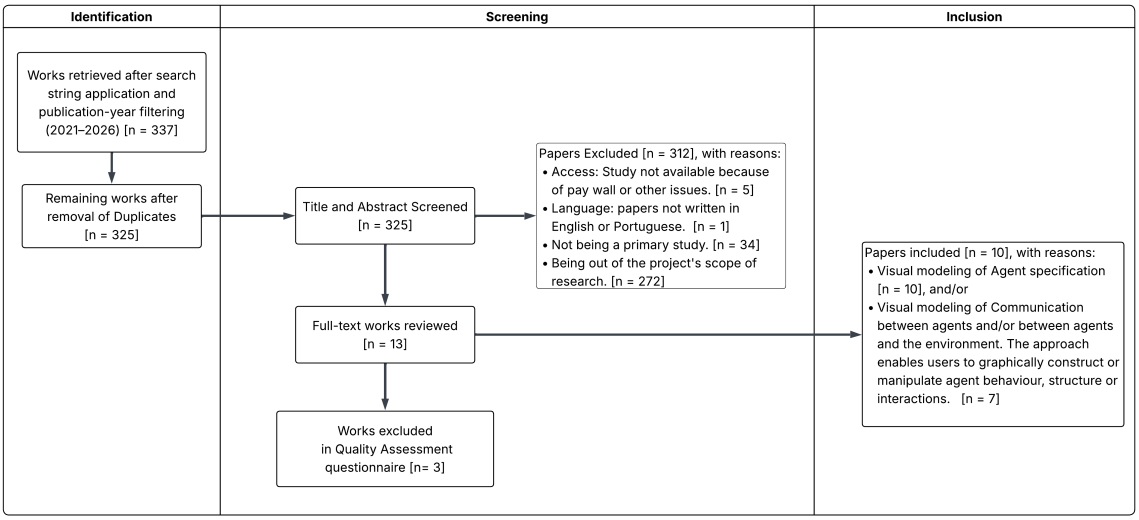


Fig. 1. Diagram describing the methods used during the search phase.

Tab. 1. Reporting completeness and data-extractability assessment.

Id	Quality criterion
Q1	Is the software or model properly identified and described in the article?
Q2	Does the software or model explicitly implement or support components of agent-oriented programming (e.g., beliefs, plans, goals, or messages)?
Q3	Does the article describe the end-user profile?
Q4	Is the application domain well described in the article?
Q5	Are the limitations of the presented software or model reported?
Q6	Is the chosen visual programming paradigm explicitly explained in the article?

additional papers did not pass the reporting completeness and data-extractability assessment, receiving scores of 2.0, 2.5, and 3.0. This resulted in a final set of 10 papers for the systematic mapping. These papers are shown in Table 2, along with their unweighted quality scores.

2.3. Data Analysis

To understand which visual programming paradigms have been used in agent-oriented programming between 2021 and 2026, this study addresses the research questions outlined in the previous section, based on the 10 papers selected through the systematic review. Table 3 presents the classification across four categories: framework type, framework name, visual paradigm, and application domain.

Each framework was classified by the presence of a conceptual model, a conceptual model with software implementation, or solely the software implementation. Papers 1, 3, and 7 did not intend to be regarded as visual programming models but were interpreted as such due to the utility of the tools presented.

Tab. 2. Final set of papers and their score.

Id	Paper Title	Score
1	A New Modeling Framework for Cyber-Physical and Human Systems [Poursoltan et al. 2022]	4.5
2	Debugging in the Domain-Specific Modeling Languages for multi-agent systems [Tezel and Kardas 2025]	5.5
3	Domain-Specific Modelling Languages for Participatory Agent-Based Modelling in Healthcare [Godfrey 2021]	4.5
4	Doodlebot: An Educational Robot for Creativity and AI Literacy [Williams et al. 2024]	4.0
5	Goal-Oriented End-User Programming of Robots [Porfirio et al. 2024]	5.5
6	Interactive Constructionist Scaffolds for Agent-based Modeling and Programming in NetLogo [Chen et al. 2024]	5.0
7	Modeling Service Choreographies and Collaborative Tasks for Autonomous Mixed-Fleet Systems [Wiesmayr et al. 2024]	4.5
8	SavviDriver: model-based framework for game-based testing of autonomous vehicles in diverse multi-agent traffic scenarios [Chan et al. 2026]	4.5
9	The Pocketworld Playground: Engaging Online, Out-of-School Learners with Agent-based Programming [Chen et al. 2023]	5.5
10	Visual Programming of a Human-Machine Interface for a Multi-Robot Support System [Sartori and Schlette 2021]	4.0

Q1 – Which visual programming frameworks for agent-oriented systems have been reported in the literature, and what is their prevalence? Table 3 does not indicate a meaningful prevalence of any particular visual programming framework across the selected studies. Although papers 6 and 9 both use Pocketworld and Playground, these papers share the same authors, suggesting that this recurrence reflects the continuation of a specific research line rather than broader adoption of these frameworks within the field. Overall, the results suggest that the use of visual programming frameworks in agent-oriented programming remains diverse, with no single framework emerging as dominant.

Q2 – How do these frameworks classify in terms of visual paradigm and agent architecture? The predominant visual paradigms identified were Flow-Based, Block-Based, State Machine, and Node Graph, as presented in Table 3. Other paradigms were employed primarily to represent specific agent-related features. For example, Paper 1 uses a class diagram to illustrate the compositional relationships among different agents. Paper 2 incorporates dependency structures within its flow-based representations, while Paper 7 proposes a set of UML-based models. Additionally, all visual software tools analyzed employed forms to configure initial parameters, define settings, and assign default values.

Regarding paper 1, it identified the use of Discrete Event System Specification (DEVS) for agent description, with an interest in displaying input and output ports rep-

Tab. 3. Categorisation of papers in terms of visual programming paradigms and application domain. The Id refers to the paper Id in Table 2.

Id	Framework Type	Framework Name	Visual Paradigm	Application Domain
1	Conceptual Model	—	State Machine Others	Multi-Agent Systems General Modeling
2	Software Conceptual Model	MASDebugFW; SAML	Flow-Based Node Graph	Multi-Agent Systems General Modeling
3	Conceptual Model	—	Flow-Based Node Graph	Multi-Agent Systems General Modeling
4	Software	DoodleBot	Block-Based	Education Robotic Human-Robot Interaction
5	Software Conceptual Model	Polaris	State Machine	Robotics Human-Robot Interaction Requirements Engineering
6	Software	Pocketworld Playground	Block-Based	Education Learning Environments
7	Conceptual Model	—	Flow-Based State Machine Node Graph	Industrial Logistics
8	Software Conceptual Model	SavviDriver	Others	Multi-Agent Systems General Modeling
9	Software	Pocketworld Playground	Block-Based	Education Learning Environments Multi-Agent Systems
10	Software	—	Flow-Based	Human-Robot Interaction Industrial

resenting messages coming in and out of the agent. This formalism, focusing primarily on the Phase Transition Diagram (PTD), was, in this mapping, classified as a generalisation of *State Machines*. This agent paradigm is hybrid, combining reactive, event-driven behaviour with plan-like trajectories in the PTD.

The paper that fulfils all the requirements of a *BDI architecture* is number 2. The system structure is modelled under distinct perspectives, with the diagrams representing the sequences and dependencies between the elements: the MAS diagram, which depicts the structural relationships between agents and capabilities and the capability diagram, classified as a dependency diagram that shows how beliefs and events trigger plans; and the flow-based visualisation to explicitly show the actions and the environment functions (operations) called by them.

In paper number 3, the action-card language is described, based on the card system of public hospitals. Each card describes a situation and a protocol, meaning the actions are based on the outcome of a previous card. That composes a flow-like visual paradigm. This paper was classified as reactive, since behaviour is mainly driven by message triggers and condition-action execution.

The interface described in paper 4, classified as goal-oriented, uses a block-based language to teach AI concepts to students through interaction with physical or virtual robots. Collaboratively, users describe the agents' behavior by selecting blocks that represent specific goals and choosing planning algorithms to compute optimal trajectories.

Paper 5 falls within the field of Human-Robot Interaction (HRI), as it focuses on interactions between humans and robotic agents. In the goal-oriented Polaris software, the environment is depicted as a map on which the robot will execute its actions. A group of goals is represented as a state in the goal automaton, called a checkpoint. The user can insert a predicate to define the goal. After configuring the goal automaton in the drawing area, a flow diagram is generated that outlines the necessary steps to fulfill each goal.

The authors of papers 6 and 9 present the Pocketworld Playground (POP), a block-based environment built on NetLogo (an agent-based simulation framework). POP applies block programming in a reactive paradigm, allowing the construction of complex behaviours from decentralised and probabilistic "micro-behaviours".

Paper number 7 describes a reactive, event-driven model for coordinating mixed-fleet systems that reflects the characteristics of the logistics domain. It describes several models, mainly the interaction model, which represents the flow and required sequence of events that activate services, and the process model, where processes are defined as choreographies of multiple services connected through events.

The work presented in 8 differs from others by presenting a tree structure for goals, based on the KAOS (Keep All Objectives Satisfied) model, to specify driving styles in autonomous vehicles. In the tree, sub-objectives branch out from the top-level objective as a statement formed by AND and OR functions. If the statement is true, then the top-level objective is fulfilled.

Paper 10 proposed a system that uses a "skill block" to control the actions and reactions of industrial collaborative robots. The key difference in this solution is that it treats robots and the human operator as functional blocks in a logical chain. This allows factory workers to program assembly sequences simply by chaining success and error blocks, in a system also classified as flow-based.

Overall, 4 papers used flowchart diagrams, 3 used block-based language, 3 used state machines, and 3 used node graphs. This indicates little to no prevalence of a specific visual programming paradigm over the others. Flow-based representations appear across all agent categories, indicating greater versatility, while block-based paradigms are mainly associated with reactive agents, as they emphasize the composition of behaviors through predefined blocks, often focusing on direct action execution rather than explicit goal modeling or hybrid reasoning. State machine representations are predominantly associated with reactive and goal-oriented architectures, since they naturally model state transitions triggered by events or conditions and can also represent goal progression through states. Node graph paradigms are mainly associated with BDI-like and hybrid architectures, as their structure supports the explicit representation of dependencies, reasoning flows, and relationships between internal components such as beliefs, plans, and actions.

Q3 – What is the end-user profile and target application domain for these tools? The frameworks analysed were predominantly designed for laypeople in Computer Science, including children, teenagers, and individuals interested in agent-oriented programming for personal use. Regarding the application domain, papers 3 and 7 specifically target professionals in the health and logistics domains who require agent models. Moreover, three of the ten articles fall within the robotics domain. The remaining frameworks, mainly the ones containing conceptual models, targeted CS students, professionals with programming knowledge, or had no specified end-user profile. The target application domain for each paper is shown in Table 3.

Q4 – How does user interaction occur, and what are the main programming difficulties reported? The user interaction mode and the core challenge for each study are described in Table 4. A difficulty reported across most frameworks is the developer's required abstraction skills. Furthermore, block-based languages may face limitations when modelling complex agent environments, as their abstractions are often designed to simplify programming for children, teenagers, and lay users. While this makes them accessible, it can also limit the flexibility needed to represent dynamic environments and more advanced agent behaviours.

Tab. 4. Summary of interaction modes and core challenges.

Id	Interaction Mode	Core Challenge
1	Conceptual meta-modelling	Modelling human unpredictability
2	Modelling through diagrams	Complex model-to-code navigation
3	Conceptual modelling with diagrams	Technical-clinical communication gaps
4	Block-based with drag-and-drop input and output in animations or physical robots	Physical setup and interface clutter
5	Designing goal automata via forms	Shift from action-based to predicate-based logic
6	Interaction through help dialogues and system building with blocks	Users overlooking hidden help tools
7	Designing event-driven service diagrams (conceptual model)	Visualizing dynamic and partially unknown processes
8	Modelling the goal tree	Lack of systematic reconfiguration tools
9	Interaction through help dialogues and system building with blocks	Limited support for advanced features
10	Building sequences with drag-and-drop predefined skill blocks	Manual data input required for digital twins

Q5 – What is the degree of freedom for customising and extending agent behaviours? The degree of freedom of customising agent behaviour is measured using BDI agents as a baseline of comparison. Agents classified as reactive provide better support for describing the sequence of actions and focus on events and their corresponding triggered actions, possessing no teleological behaviour or reasoning cycles whatsoever.

Paper 1, described as hybrid, provides visual tools for events, triggers, and plan-like descriptions in the phase transition diagram, yet it does not explicitly support beliefs as a prerequisite for plan execution, lifecycles, or goals.

Regarding goal-oriented frameworks, the focus is on structures that define the steps necessary for an agent to fulfil a goal, in sequential form (papers 4 and 5) or tree form (paper 8). Paper 5 presents tools for defining an agent's goals through forms, while paper 4 provides an even broader option of using a block-based interface for defining an agent's attributes.

Lastly, paper 2's flow-based visual representation contains all the tools necessary to define an agent within the BDI paradigm.

2.4. Mapping Remarks

The systematic mapping highlights several relevant trends in the current literature. Most visual programming tools focus on reactive and goal-oriented agents, while paper 2 [Tezel and Kardas 2025] presents the only selected work that fully supports a BDI architecture. This indicates a scarcity of tools, models, and visual structures for representing BDI-specific concepts, particularly agent communication and the definition of beliefs and goals through visual blocks. The mapping also shows that these tools primarily target children, lay users, and professionals from diverse fields who need accessible ways to model problems without conventional programming environments.

3. Visual MASPY Prototype

Based on the systematic mapping, we developed a reduced-scope visual programming environment for MASPY [Mellado et al. 2026]. The prototype supports the definition of agents, initial beliefs and goals, plan events and contexts, messages, variables, and simple control flow. Its value model is limited to simple values and does not yet support data structures such as lists or tuples.

The interface combines block-based construction with a flow-oriented representation of plan execution. Figure 2 shows the *Build System* screen, which integrates system configuration and initial set-up. The example defines the agents `Sender` and `Receiver`, connected through the `Default` channel, and associates the goal `send_info(''Hello'')` with the sender and the belief `Receiver` with the receiver.

Each plan is divided into *Plan Attributes*, which declares its triggering event and context, and *Plan Code*, which defines its operations. Figures 3(a) and 3(b) show the sender and receiver plans.

The purple block represents the event that activates a plan, which can wrap a triangular type of event block and either a Goal or a Belief block. For example, gaining `Goal(''send_info'', Any)` is translated into `@pl(gain, Goal(''send_info'', Any))`, with the received value bound to `msg`. The blue message block represents agent communication; its destination, channel, and content generate `self.send(''Recv'', achieve, Goal(''receive_info'', msg))`. In the receiver plan, the context block adds `Belief(''Receiver'')` as a condition for execution.

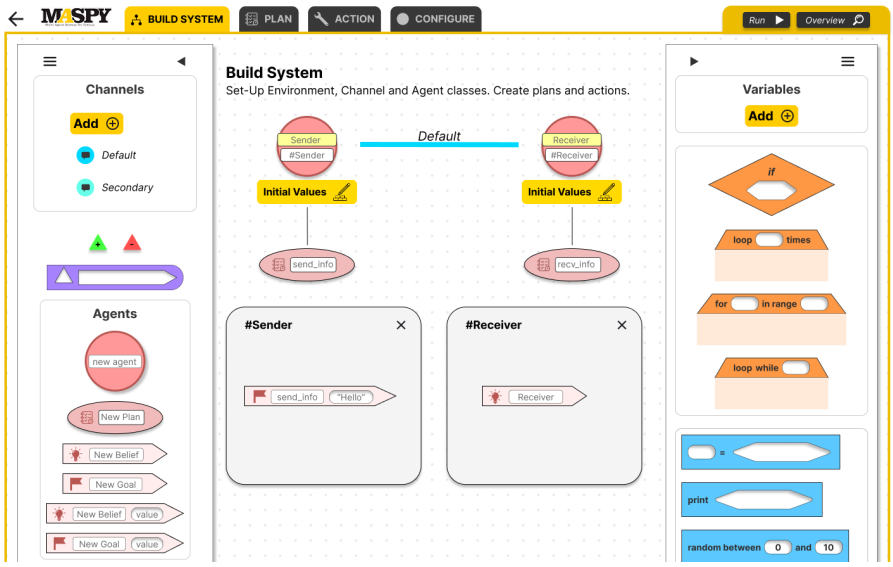
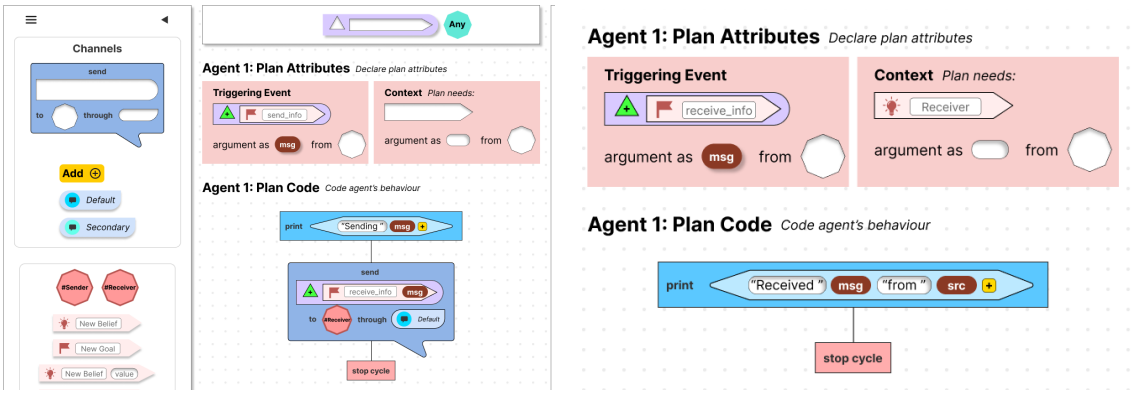


Fig. 2. Unified system construction and initial set-up screen of the prototype.



(a) **(b)**

Fig. 3. Visual plan definitions: (a) the `Sender` plan and (b) the `Receiver` plan.

Figure 4 shows the intended output MASPY program. Agent cards and initial values become class instantiations and constructor arguments, plan attributes become decorator arguments, and operational blocks become ordered method statements. This direct mapping keeps the generated code inspectable while hiding Python syntax from users.

4. Conclusion

In this paper, we have presented a systematic mapping towards the use of visual tools for agent programming. We have found no meaningful prevalence of any single visual paradigm, and identified a gap in the literature: current visual tools focus predominantly on reactive and goal-oriented agents, leaving BDI agents under represented, especially regarding visual structures for expressing beliefs, goals, and message exchange between agents.

As a result, we have presented a visual programming interface prototype for the MASPY framework, combining flow-based and block-based paradigms into an educational tool to facilitate the development of BDI programs and the explanation of BDI

```python
from maspy import *

class Sender(Agent):
    @pl(gain, Goal("send_info", Any))
    def send_info(self, src, msg):
        self.print(f"Sending {msg} to Receiver")
        self.send("Recv",achieve,Goal("receive_info",msg))
        self.stop_cycle()

class Receiver(Agent):
    @pl(gain, Goal("receive_info", Any), Belief("Receiver"))
    def recv_info(self, src, msg):
        self.print(f"Received: {msg} from {src}")
        self.stop_cycle()

if __name__ == "__main__":
    Sender(goals=Goal("send_info", "Hello"))
    Receiver("Recv",beliefs=Belief("Receiver"))
    Admin().start_system()
```

Fig. 4. MASPY code corresponding to the visual definitions.

concepts for laypeople in Computer Science.

Future work will evaluate the prototype with MAS and MASPY specialists performing representative programming tasks. The evaluation will consider task completion, execution time, errors, and usability problems identified through Nielsen's usability heuristics [Nielsen 2024], while user satisfaction will be measured using a CSAT questionnaire [Hayes 2008]. An introductory MASPY activity will also compare task performance before and after using the interface to examine its contribution to learning. The results will guide refinements to the visual notation, interaction mechanisms, and MASPY integration.

Acknowledgments: This study was financed in part by project: 444568/2024-7, CNPq/MCTI/FNDCT Nº 22/2024, *Conhecimento Brasil Programme – Support for Network Projects with Brazilian Researchers Abroad*

References

Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355.

Chan, K. H., Zilberman, S., and Cheng, B. H. C. (2026). Savvidriver: model-based framework for game-based testing of autonomous vehicles in diverse multi-agent traffic scenarios: Savvidriver: model-based framework for game-based testing... *Softw. Syst. Model.*, 25(1):135–161.

Chen, J., Horn, M., and Wilensky, U. (2024). Interactive constructionist scaffolds for agent-based modeling and programming in netlogo. In *Proceedings of FabLearn / Constructionism 2023: Full and Short Research Papers*, FLC '23, New York, NY, USA. Association for Computing Machinery.

Chen, J., Zhao, L., Horn, M., and Wilensky, U. (2023). The pocketworld playground: Engaging online, out-of-school learners with agent-based programming. In *Proceedings of the 22nd Annual ACM Interaction Design and Children Conference*, IDC '23, page 267–277, New York, NY, USA. Association for Computing Machinery.

Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, 42(2–3):213–261.

Godfrey, T. (2021). Domain-specific modelling languages for participatory agent-based modelling in healthcare. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 654–659.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.

Hayes, B. E. (2008). *Measuring customer satisfaction and loyalty: survey design, use, and statistical analysis methods*. ASQ Quality Press.

Kitchenham, B. A. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

Kuhail, M. A., Farooq, S., Hammad, R., and Bahja, M. (2021). Characterizing visual programming approaches for end-user developers: A systematic review. *IEEE Access*, 9:14181–14202.

Mellado, A. L. L., Pinz Borges, A., and Alves, G. V. (2026). Maspy: A python-based framework for developing bdi multi-agent systems. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Computational Social Science: The PAAMS Collection*, pages 216–227, Cham. Springer Nature Switzerland.

Nielsen, J. (2024). 10 usability heuristics for user interface design.

Porfirio, D., Roberts, M., and Hiatt, L. M. (2024). Goal-oriented end-user programming of robots. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '24, page 582–591, New York, NY, USA. Association for Computing Machinery.

Poursoltan, M., Pinède, N., Vallespir, B., and Traore, M. K. (2022). A new modeling framework for cyber-physical and human systems. In *2022 Annual Modeling and Simulation Conference (ANNSIM)*, pages 90–101.

Sartori, A. and Schlette, C. (2021). Visual programming of a human-machine interface for a multi-robot support system. pages 387–392.

Tezel, B. T. and Kardas, G. (2025). Debugging in the domain-specific modeling languages for multi-agent systems. *J. Comput. Lang.*, 83:101325.

Wiesmayr, B., Zoitl, A., and Hästbacka, D. (2024). Modeling service choreographies and collaborative tasks for autonomous mixed-fleet systems. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion '24, page 234–244, New York, NY, USA. Association for Computing Machinery.

Williams, R., Ali, S., Alcantara, R., Burghleh, T., Alghowinem, S., and Breazeal, C. (2024). Doodlebot: An educational robot for creativity and ai literacy. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '24, page 772–780, New York, NY, USA. Association for Computing Machinery.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. Wiley.