Towards Application of Programming Language for Communication Flows Specification in Multi-agent Systems on Real-World Use Cases

Tomislav Peharda, Bogdan Okreša Đurić, Igor Tomičić
Artificial Intelligence Laboratory
University of Zagreb
Faculty of Organization and Informatics
Pavlinska 2, 42000 Varaždin, Croatia
{tomislav.peharda, dokresa, igor.tomicic}@foi.unizg.hr

Abstract. *In this work-in-progress article we focus on the application of programming language for communication flows specification in multi-agent systems on real-world use cases. Agents orchestration in multi-agent systems architecture may be very troublesome, mainly due to agents being independent units that may be implemented differently. The proposed programming language for communication flows specification attempts to overcome this challenge by providing explicit communication flows definitions, that agents' communications component relies on, which enables enhanced orchestration capabilities. Use cases that are covered in this paper are in the domains of streaming audio, video, and sensors data, and provide a couple of examples of how the proposed programming language may be used.*

Keywords. multiagent systems, communication flows specification, agent orchestration

1 Introduction

It is often the case within computer science that there is a type of task that is being executed repeatedly to either deliver a resource or simply do a job that shall impact the requesting party (directly or indirectly). A software term that is used to describe such a program is agent. The default definition of an agent (Russell and Norvig, 2022, p. 54) sets it as anything that is located in an environment, and can perceive it, and act upon it. One of the main characteristics of an agent, though, is its autonomy in performing a task, which means it acts as a standalone unit that is adequately designed to achieve the goal without human directions or intervention.

If a task is too complex for a single agent to complete it, the task is broken down into multiple sub-tasks, in which case each sub-task is being executed by a separate agent. In this scenario, agents complete their sub-tasks and communicate the results to other agents. This way, all agents collaborate to achieve the common goal. A cluster of interconnected agents is called a multiagent system (MAS).

Since agents are autonomous units, from a technical standpoint, each agent might be designed and implemented differently. When speaking of agent coordination and orchestration in MAS architecture, this may bring up several challenges, one of which is inconsistency in communication flows (Schatten, Tomičić, et al., 2020). For example, agent A is designed and implemented to communicate with agent B, however, agent B is not designed to communicate with agent A. In this case, agent orchestration will not be successful, as agents won't participate in the communication and will not be able to achieve the higher goal.

The herein proposed solution to this problem is a new programming language based on process calculus (Parrow, 2001, p. 481) for communication flows specification. The language shall implement concepts and mechanisms that enhance agents orchestration capabilities in MAS architecture.

The goal of this paper is to provide examples of how programming language may be utilized in the use cases of data streaming. Significance is also put on what possibilities would an engineer be provided with by using the language, and how it compares to the traditional approaches to agent design and implementation.

The rest of the paper is structured as follows. Section 2 provides information on related work, basic concepts necessary herein, and similar research. Section 3 describes the programming language proposed as a result of this research, followed by Sec. 4 where the programming language use is exemplified in a real-world use case. Finally, Sec. 5 provides a short discussion, conclusion, and some ideas that provide opportunities for future work.

2 Related Work

Regarding message transport, Corkill (Corkill, 1988) suggests a broadcasting concept in which each agent communicates with all agents in a MAS. In this case, the expectation is that each agent shall implement a filtering mechanism on the receiving side to discard messages that are not of its interest. Although flexible, this approach does not consider performance implications so much.

Similarly, Goldman, and Ziberstein in (Goldman and Zilberstein, 2003) build on top of the previously described idea with the addition of taking into account that external agents are not part of the observed MAS, and likely do not have a filtering mechanism in place. To address this, the authors propose that there is a middleware agent in the MAS to which all agents send their messages to, and the middleware agent is responsible for routing the received messages to the appropriate parties.

Berna-Koes, Nourbakhsh, and Sycara in their research (Berna-Koes et al., 2004) focus on transporting non-textual messages such as video, audio, or other media types, which are not adequately supported by most MAS frameworks as they primarily rely on textual messages. Their research proposes the introduction of backchannels, enabling agents to exchange messages of these media types.

On the other hand, service orchestration providers are also relevant to the subject of this research. Some of these providers are Kubernetes, and Docker Swarm. Basically, what these providers enable is infrastructure for the coordination of a larger number of services at once. However, the infrastructure support these providers enable is on a high level and refers to enabling services to share resources, storage, network, and enable communication (Powell, 2023). Essentially, it is about characteristics that all services share, regardless of their purpose or type. Since communication (protocol, language, etc.) differs from one service type to another, it is considered a low-level component, thus is something that these providers do not take into account.

In the research presented in (Schatten, Tomičić, et al., 2020), the authors describe a conceptual model for the orchestration platform for hybrid artificial intelligence methods that is based on a MAS architecture. The idea is that the platform enables orchestration of various game engines via application programming interfaces (APIs). Essentially, APIs are used for external users to start up game engines that they wish to use, while these game engines are being treated as agents.

Holonic systems (Rodriguez et al., 2011) in MAS architecture refer to an entity that may be observed as a standalone autonomous agent or as a piece of a larger system. As a standalone agent, holon has built-in business logic by which it acts independently and correspondingly communicates with other external agents. In the context of a larger system, it is collaborating with other internal peer agents to achieve a common goal. Such hierarchy enables the construction of a more complex organizational structure which has proved to be helpful when a complex task needs to be broken down into multiple sub-tasks.

Process calculus (Parrow, 2001) stands for a family of approaches for modeling concurrent systems with a focus on describing interaction, communication, and synchronization between agents. The main features that all approaches within process calculus share are message-passing between the independent processes that represent interactions, the use of a small number of primitives, operators to describe processes, and specific algebraic rules for process operators. Reduction rules are an essential aspect of process calculus as they help achieve the explicit description of communication that takes into account parallel composition, sequentialization, and transformation of some input into an output. Process calculus provides a formal backdrop for modeling complex distributed systems and shall be used as a theoretical foundation for the research at hand. π-calculus, which is part of the Process calculus family is used as a formal backdrop for the construction of the programming language.

3 Programming Language for Communication Flows Specification

To solve the problem of agents orchestration in MAS, the proposal is to use a programming language for communication flows specification that adds support for consistent flows specification across all agents. The high-level idea behind the programming language is that by creating communication flows specification, which essentially describes what pairs of agents engage in communication, each agent is provided with a list of communication flows it needs to implement.

The programming language, which is still being developed, is based on process calculus. Process calculus is used as a foundation for this programming language, as it embodies characteristics for communication specification, which takes into account parallelism, sequentization, and input-to-output transformation, which fit very well with the communication flows specification.

With the programming language comes a declarative engine, which is capable of processing communication flows specification. Once communication flows specification is parsed, each agent is provided with a list of communication flows it shall implement, which ensures proper agent orchestration capabilities. Ultimately, this solves the problem of communication flows inconsistency between agents, where for example, agent A might be designed to communicate with agent B, but agent B is not designed to communicate with agent A. By adding safeguard in the agent codebase, it can be easy to identify if an agent is not implementing some of the communication flows, which may especially come handy if communication flows specification gets updated in the future.

Another valuable feature of the communication flows specification is that it can serve as a source of truth for describing all communication flows in an observed MAS. Without communication flows specification, to get an idea of all communication flows in MAS, one should look into every agent codebase individually to determine its communication flows.

Three aspects of communication that the programming language is set to enhance are (a) specifying pairs of agents that communicate, (b) communication protocol, and (c) expected input and output. These aspects provide great grounds for building more advanced communication channels. Specifying pairs of agents enables a description of one-way communication between two pairs of agents. Each agent might have multiple communication flows defined where in some cases it might be in the role of a receiving party, while in others it may be in the role of the sender party. By using different symbols for indication of communication between agents, it is possible to specify what type of protocols shall be used for the communication. Currently, transmission control protocol (TCP) and user datagram protocol (UDP) protocols are supported. Lastly, defining expected input can be useful to announce that a particular agent is only capable of processing messages received in a particular format, which is helpful from the perspective of data integrity and validation. Similarly, defining expected output makes sure that other agents are acknowledged in what format should the incoming message be.

Types of agents that are supported by the programming language are: basic agent, channel agent, and holon agent. The basic agent may only send and receive messages from the channel agent and the holon agent. The channel agent serves as a proxy, with capabilities to filter and transform input to output. The holon agent is conceptually designed to serve as a gateway to agents outside the observed MAS. The holon agent is anticipated for communication with agents outside the observed MAS.

4 Application of Programming Language to Real-world Use Cases

In this section, we provide a description of how to the application of the programming language could be used to specify communication flows in a MAS.

4.1 Communicating Sensors

The use case is as follows: there are three basic agents, agent B, C, and D, which are sensors capturing temperature value every 5 minutes. These three sensors are located within the same area, however, within 5 kilometers radius. The idea is that by looking at the average value of the captured temperatures from these 3 agents, we can gain insight into the temperature of the observed area.

The three agents are solely responsible for capturing temperature. Data they collect they send to the agent A which is a channel. In the programming language, each basic agent interface utilizes the reduction rule to specify the sender and the receiver in the format sender → receiver. If the basic agent specification, keyword self is used for implication of the currently observed agent. If the keyword self is used as a sender in the communication flow specification, that implies that the currently observed agent shall send a message to the receiving counter-party. As previously said, within the programming language, channels are used to optionally transform and proxy data to the interested parties. Each of the sensors agents send a message in the JSON format `{"temperature": <numeric value ↪ >, "location": <name of location>}`. That being said, the input of the channel is also defined to take an input in this exact format. This is defined by the reduction rule in the format input → output which indicates the transformation. If a message of a different format is received, it will be dropped. Whenever the channel receives a message in the adequate format, it parses the message and extracts out only the value, which also indicates it drops the location attribute. The channel sends out the transformed message.

In this use case, there is a basic agent D which is listening to any messages emitted by the channel A. Agent D is responsible for creating an average value of temperature values it receives in the window of 5 minutes and sharing the reckoned value to the audience. For example, this could be achieved by sending an email, posting a message on a messaging application, or similar.

The snippet provided below is a description of the communication flows specification in the described multi-agent systems architecture-based use case in the proposed programming language. Figure 1 provides a visual representation of the communication flows in the described MAS.

```
channel A:
   json({"temperature": ?temp, "location
      ↪ ": ?loc}) --> ?temp

agent B:
   self -> A

agent C:
   self -> A

agent D:
   self -> A

agent E:
   A -> self
```

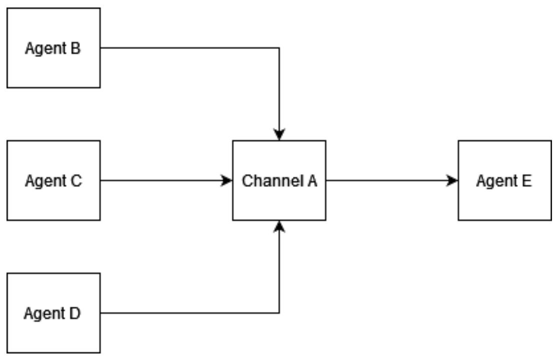

Figure 1: Communications flows

4.2 Streaming Game Platform

The second use case presented here uses a streamed holographic 3D game engine described in more detail in (Schatten, Okreša Ðurić, et al., 2022). The game engine described therein makes it possible for players to stream games over a 3D holographic gaming console. The game engine, as well as the implementation of the gaming console, is implemented using a MAS, since the required level of synchronisation and complexity fits tightly into the observable natural application domains of systems comprising agents.

The above referenced streamed holographic 3D game engine's prototype architecture, shown in Fig. 2, features at least three types of agents: (a) a game agent, (b) a game streaming agent, (c) a videoconferencing agent. The game agent is running the game, and making sure the camera transformer is used correctly, so the game is shown as intended on the holographic 3D gaming console. The game streaming agent makes sure that the game is streamed towards the player, i.e. that the streaming channel is established and maintained towards the player, and that the player is provided with the satisfying gaming experience. The videoconferencing agent implements a group videoconferencing option that provides the player with the possibility of sharing a link towards the public or a select number of individuals, who could watch the stream, join in a group discussion on the player's performance, and watch the player's camera stream alongside the game stream.

Based on the above description, every single player's experience can be described as being the result of a process realised as a group effort of a combination of multiple agents, i.e. a multiagent system. As such, it can be described using the programming language described earlier in this paper, in terms of communication flows specification. In describing this example in the context of the programming language for communication flows specification, only a single communication flow is considered: a signal is sent from the user interface towards the game streaming agent, forwarded towards the game agent. Furthermore, a specific example of interacting with the game is observed in more detail, in order to keep the example in the constraints appropriate for this paper.

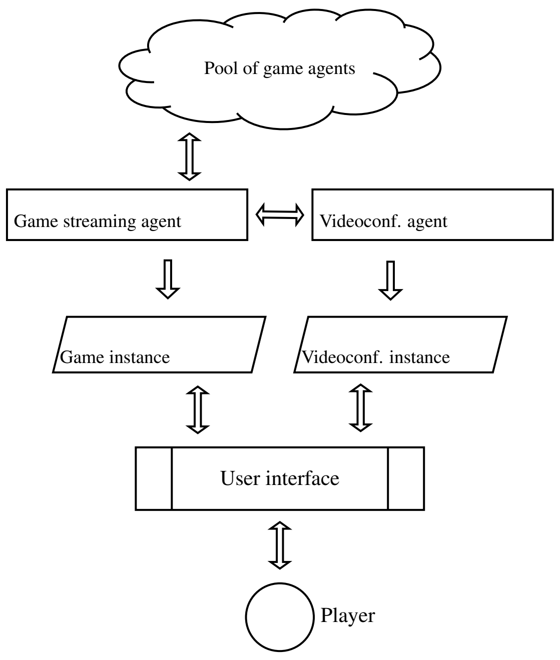

Figure 2: Architecture prototype of the streamed holographic 3D game engine (Schatten, Okreša Ðurić, et al., 2022)

Therefore, three agents can be observed: the user interface (agent UI), the game streaming agent (agent S), and the game agent (agent G). User interface can be modelled as an agent for the purposes of modelling the system using the proposed programming language, even though the implementation may not follow this specification, and the user interface is not implemented as a standalone agent similar to the other two, but as a more abstract form of a reactive agent that is not autonomous.

Agent UI sends player-activated controls from a controller to agent S. The game streaming agent (agent S) takes input from the user interface, and translates it into the correct command that is expected by the game agent. Usually, the game agent is expecting one of the commands in a predefined set of inputs that are used for controlling the player or other in-game elements, written in a configuration file, referenced as `config` in the listing below. Translation is performed using a list of expected input values and the appropriate output values, such as is shown here, where the event from the controller is listed as the key, and the appropriate action is listed as the value of the key-value pair:

```
{
    "controls":
        {
            "UP": "up",
            "DOWN": "down",
            "LEFT": "left",
            "RIGHT": "right",
            "SELECT": "esc",
            "START": "enter",
            "A": "space",
            "B": "enter"
        }
}
```

The communication flow described above can be modelled using two channels (UIS and SG), and three agents (UI, S, G). The first channel is used by agent UI to send user input to agent S (channel UIS), and the second channel is used to send the appropriate command from agent S to agent G (channel SG). Translation of the received user input can be performed either as a part of an agent, or as a part of a channel. Since a channel is implemented as a specific form of an agent, translation action can be given to the channel, as is shown for the purposes of this example. The channel translates the input into the command based on both of the received values, and the applicable configuration file. The listing below can be used to describe the example:

```
channel UIS:
    json({"key": ?key, "player": ?player})
        ↪  --> json({"key": ?key})

channel SG:
    json({"game": ?game, "key": ?key}) -->
        ↪  config[?game][?key]

agent UI:
    self -> UIS

agent S:
    UIS -> self & self -> SG

agent G:
    SG -> self
```

5 Conclusion and Future work

The proposed programming language, and the corresponding declarative engine are taking a holistic approach to solving the problem of agent orchestration and coordination in the multi-agent systems architecture. The programming language's main focus is put on enhancing communication flows specification, which in the context of MAS implies to agent orchestration. Some of the identified flaws of the traditional approaches to the communications flows specification primarily deal with communication flows inconsistency across different agents, which leads to challenges in the orchestration.

Programming language based on π-calculus is designed to solve this problem by introducing communication flows specification, which shall provide description of communication flows of the entire MAS in a single specification flow. This also gives visibility into all communication flows present in MAS, which would otherwise be impossible, as one would need to look into each agent separately to determine communication flows. The declarative engine, which is essentially in the role of a parser extracts out valuable information for each agent separately, so that each agent might be provided with the list of communication flows it needs to implement.

The above is one of the benefits of the implementation approach using the proposed programming language, as opposed to the traditional approaches to agent design and implementation. Therein, there is a high possibility for a higher degree of code redundancy and duplication, especially with regard to the communication component, as each agent is required to implement it individually. Additionally, there is a possibility for inconsistency in communication flows between agents.

By being provided with per-agent-parsed list of communication flows to implement, each agent is in a good spot to achieve proper orchestration within the MAS it resides in. By implementing communication mechanism for all the required communication flows, all agent communication shall be consistent, as there should be no discrepancies where one agent might be designed to send message to another one, while the other is not designed for receiving the message.

Using the described example in the domain of sensors capturing temperature, we have shown how the utilization of different types of agents supported by the programming language may conceptually lead to proper orchestration, and ease of communication flows specification. The example also touches on the flexibility of agents business logic, as agents may have different purpose, which implies that the programming language in no way attempts to limit the business logic, but only to enhance the communication component. This is further emphasised using the second provided example.

The future work of this research is in the direction of designing and developing an orchestration platform that utilizes the communication flows specification, and starts up agents based on it. The platform intends to take the current work a step further, as with the programming language solely, agents are provided with the communication flows that are needed to be implemented. The idea around the orchestration platform is that whole agent communication would be covered by the platform itself by implementing the communication layer around an agent. This essentially means that with the communication flows specification being in the place, the only needed part for agent to work properly would be to specify what business logic should an agent do.

Acknowledgement

This work has been fully supported by the Croatian Science Foundation under the project number IP-2019-04-5824.

References

Berna-Koes, M., Nourbakhsh, I., & Sycara, K. (2004). Communication efficiency in multi-agent systems. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, *3*, 2129–2134.

Corkill, D. D. (1988). *Design alternatives for parallel and distributed blackboard systems*. University of Massachusetts at Amherst. Computer; Information Science [COINS].

Goldman, C. V., & Zilberstein, S. (2003). Optimizing information exchange in cooperative multi-agent systems. *Proceedings of the second international joint conference on Autonomous agents and multi-agent systems*, 137–144.

Parrow, J. (2001). An Introduction to the π-Calculus. In J. A. Bergstra, A. Ponse, & S. A. Smolka (Eds.), *Handbook of Process Algebra* (1st ed., pp. 479–542). Elsevier.

Powell, R. (2023). *Docker Swarm vs Kubernetes: How to choose a container orchestration tool*. CircleCI. Retrieved June 28, 2023, from https://circleci.com/blog/docker-swarm-vs-kubernetes/

Rodriguez, S., Hilaire, V., Gaud, N., Galland, S., & Koukam, A. (2011). Holonic multi-agent systems. In *Self-organising software* (pp. 251–279). Springer.

Russell, S. J., & Norvig, P. (Eds.). (2022). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson Education Limited.

Schatten, M., Okreša Đurić, B., & Peharda, T. (2022). Towards a Streamed Holographic 3D Game Engine. In N. Vrček, L. Guàrdia, & P. Grd (Eds.), *Proceedings of the Central European Conference on Information and Intelligent Systems* (pp. 17–22). Faculty of Organization and Informatics.

Schatten, M., Tomičić, I., & Okreša Đurić, B. (2020). Orchestration Platforms for Hybrid Artificial Intelligence in Computer Games – A Conceptual Model. In V. Strahonja, W. Steingartner, & V. Kirinić (Eds.), *Central European Conference on Information and Intelligent Systems* (pp. 3–8). Faculty of Organization and Informatics, University of Zagreb.