Integrating Procedural Ontologies in VEsNA: Study, Requirements, and Preliminary Design

Diana Ivan[1,*,†], Angelo Ferrando[2,*,†], Andrea Gatti[1,*,†], Giovanna Guerrini[1,*,†] and Viviana Mascardi[1,*,†]

[1]University of Genova, Italy
[2]University of Modena-Reggio Emilia, Italy

Abstract

In this paper we overview the literature on procedural ontologies and on the integration of ontologies in cognitive and declarative agent frameworks. Based on what we learn from this study, we make a proposal for representing procedural knowledge as OWL ontologies and we outline an extension of the VEsNA framework to integrate them and take advantage of their features.

Keywords

Procedural ontologies, BDI agents, VEsNA

1. Introduction

Quoting the widely accepted definition by Jennings, Sycara, and Wooldridge [1], an intelligent agent is a computer system that is *situated* within an environment, is capable of engaging in *social* interactions, and is autonomous, reactive, and proactive. The strong definition of agents addresses the need of being conceptualized with *mentalistic attributes* such as beliefs, desires, goals, and intentions, leading to the concept of cognitive agents. The most well known architecture for cognitive agents is the Belief-Desire-Intention (BDI) one [2].

In our vision of the next generation cognitive agents, *agent's sociality involves humans* thanks to natural language interaction via chatbots, and *agents are situated in a virtual reality*, to account for the needs raised by the metaverse [3].

This vision is represented by Figure 1, where cognition, natural language interaction, and virtual reality are in full equilibrium.

To ground our vision, driven by current technological and social challenges, we designed and developed VEsNA [4], a general-purpose and agent-based framework for managing *V*irtual *E*nvironments via *N*atural language *A*gents.

VEsNA is freely available to the research community from https://github.com/driacats/VEsNA and, as shown in Figure 1, it integrates

WOA 2024: 25th Workshop "From Objects to Agents", July 8-10, 2024, Forte di Bard (AO), Italy
*Corresponding authors.
†The authors contributed equally.
dianaivan10@yahoo.it (D. Ivan); angelo.ferrando@unimore.it (A. Ferrando); andrea.gatti@edu.unige.it (A. Gatti); giovanna.guerrini@unige.it (G. Guerrini); viviana.mascardi@unige.it (V. Mascardi)
0009-0000-6046-9781 (D. Ivan); 0000-0002-8711-4670 (A. Ferrando); 0009-0003-0992-4058 (A. Gatti); 0000-0001-9125-9867 (G. Guerrini); 0000-0002-2261-9926 (V. Mascardi)

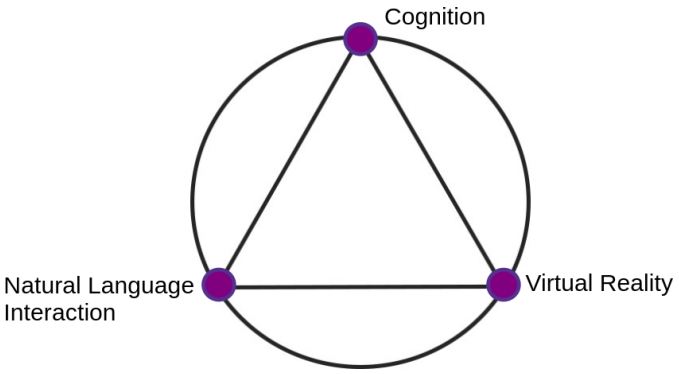

Figure 1: Our vision of social and situated intelligent agents in the metaverse era.

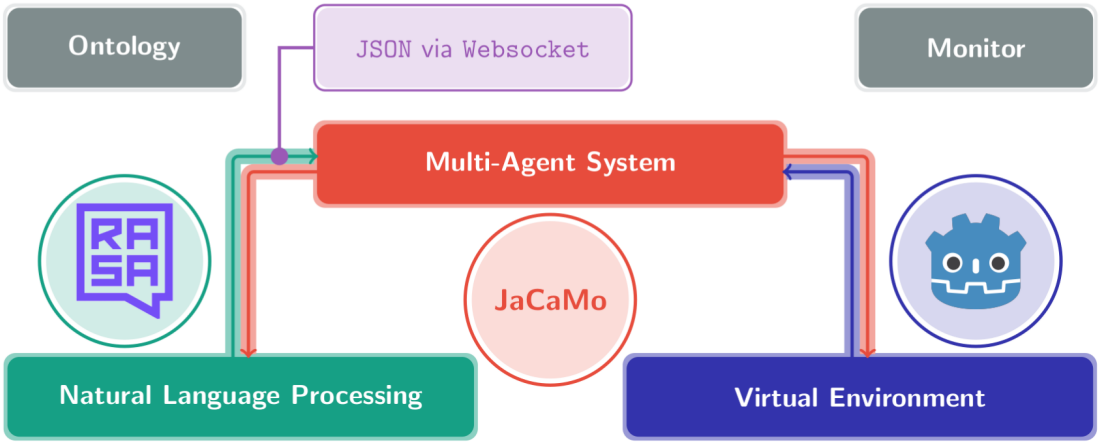

Figure 2: The VEsNA framework implementing our vision.

(i) a chatbot-like interface to support the sociality of agents; thanks to the chatbot supporting natural language interaction, sociality is not limited to agent-agent conversations, but it is extended to agent-human ones;

(ii) a framework for building the dynamic virtual environment; virtual reality is very suitable for supporting agents' situatedness: therein, agents are expected to perceive the reality and to act over it;

(iii) a framework for implementing cognitive, declarative agents able to reason about knowledge and to provide human-readable explanations; the cognitive framework supports the agents' autonomy, reactivity, proactivity.

In the current release of VEsNA, the chatbot frontend is implemented in Rasa [5, 6], which is an open source project, and the virtual environment is implemented in Godot [7], again fully open source. The framework for supporting cognitive agents is JaCaMo [8]. For the implementation of BDI agents, JaCaMo exploits the Jason interpreter for the AgentSpeak declarative language

[9].

The presence of ontologies in VEsNA was considered since the very beginning as a tool to provide a unique specification of the conversation domain, and generate skeletons for both the Jason agents in the backend and the chatbot frontend from it [10].

Currently, we are considering the possibility to use ontologies also to describe agents' behaviours. We believe that procedural knowledge expressed as Jason plans and procedural knowledge specified via ontologies should co-exist, each having its own peculiar features and strengths. For example, Jason plans are of course natively executable, while ontologies are not, but ontologies may model procedures in a more flexible, human-readable, portable and explainable way than Jason plans. Also, ontologies may undergo formal reasoning [11, 12, 13] and verification [14]; although this is true also for BDI agents [15, 16, 17], reasoning about the features of ontologies is more consolidated and has a broader scope. Hence, ontologies may model abstract processes and their decomposition into sub-processes, and may support reasoning about such processes, while Jason plans may implement the execution of atomic actions.

The contribution of this paper is twofold. First, we design of a novel procedural knowledge structure, taking the existing literature on procedural ontologies and behavior trees into account. By leveraging ontologies, this structure serves as an implementation-agnostic foundation for representing agents' planning and decision-making processes and it empowers agents to make informed choices and adapt to changing circumstances. The procedural knowledge structure is fully independent of any framework that might take advantage of it, albeit our final goal it to exploit it inside VEsNA.

Second, we make our vision more concrete and implementation-driven by eliciting the requirements for integrating procedural ontologies in VEsNA, and we discuss the services that a VEsNA agent should implement to exploit them.

The paper is organized in the following way. Section 2 overviews the works related with this research. Section 3 describes the design of the procedural ontologies we aim at integrating into VEsNA, while Section 4 discusses the requirements for the integration and outlines a high level design of the services that should be implemented to perform it. Finally, Section 5 concludes.

2. Related Work

Behavior trees are closely connected with what we aim at representing using standard ontological languages and are a valuable source of inspiration for our proposal. Also, they are widely employed in a domain that is extremely relevant for the VEsNA project, namely that of game engines and mixed reality. Finally, they have been used to represent BDI-like plans in a few recent studies. For this reason we devote Section 2.1 to introduce and discuss them. In Section 2.2 we introduce the Process Specification Language and the Planning Domain Definition Language, that include some design choices relevant for our work, while in Section 2.3 we overview the literature on integrating ontologies into frameworks for declarative and BDI agents.

2.1. Behavior Trees

In order to introduce behavior trees, we quote Colledanchise and Ögren [18]:

> *A behavior tree is a way to structure the switching between different tasks in an autonomous agent, such as a robot or a virtual entity in a computer game. Behavior trees were developed in the computer game industry, as a tool to increase modularity in the control structures of Non-Player Characters [19, 20]. Formally speaking, a behavior trees is a tree where the internal nodes are called control flow nodes and leaf nodes are called execution nodes. Control flow nodes have at least one child and, in the classical formulation, there exist four categories of control flow nodes (sequence, fallback, parallel, and decorator) that are always internal nodes and two categories of execution nodes (action and condition) that are always leaves.*

The intuition behind the four kinds of control flow nodes is that a sequence node succeeds if all of its children do; a fallback node succeeds if one of the children does; a parallel node succeeds if m out of its n children do, where m is a user-defined threshold; a decorator node applies a custom function to the return value of its unique child. As far as leaves are concerned, an action node executes an action while a condition node checks the truth value of a proposition.

Behavior trees clearly share some features with AgentSpeak(L) plans, and indeed a few researchers propose to represent BDI goal-based processes as behavior trees. For example, GORITE [21] is a BDI agent framework where the representation of procedural knowledge is similar to behavior trees but, unlike behavior trees, process models can be used to represent both individual and team behaviors. GORITE allows individual agents and teams of agents to reason about the goals that they are pursuing or intend to pursue, going beyond the power of behavior trees.

Misteli et al. [22] exploit interactive behavior trees [23] to enable user interaction with the system where smart objects are associated with BDI agents. The notion of BDI adopted in [22] is oversimplified w.r.t. what we mean by BDI in the agent-oriented software engineering community, but it allows real and virtual smart objects to act autonomously and make decisions based on their emotions and the state of the system.

Other recent works mention BDI agents together with behavior trees [24, 25], but the connection with Rao and Georgeff's architecture is weak and mainly used as a metaphor. Although not mentioning the BDI architecture at all, the work by Safronov at al. on [26] on belief behavior trees seems relevant for our work, since they allow to automatically create a policy that controls a robot in partially observable environments, which is what we expect to happen to VEsNA agents situated in a virtual world.

2.2. Procedural Ontologies and Automated Planning

The Process Specification Language (PSL) [27, 28] has been designed to facilitate the accurate and comprehensive exchange of process information among various manufacturing systems. It aims to cover a wide range of areas such as scheduling, process modeling, process planning, production planning, simulation and project management. As reported in [27],

> *The primary component of PSL is an ontology designed to represent the primitive concepts that, according to PSL, are adequate for describing basic manufacturing, engineering, and business processes.*

The PLS ontology basic concepts include

> *the 1-place predicates 'activity', 'activity-occurrence', 'object', and 'timepoint' for the four primary kinds of entity in the basic PSL ontology, the function symbols 'beginof' and 'endof' that return the timepoints at which an activity begins and ends, respectively, and the 2-place predicates 'is-occurring-at', 'occurrence-of', 'exists-at', 'before', and 'participates-in', which express important relations between various elements of the ontology.*

PDDL, which stands for Planning Domain Definition Language [29], is a formal language used in the field of artificial intelligence and automated planning to describe planning problems and domain models. PDDL is designed to express the different aspects of a planning problem, including the initial state, the actions that can be performed, the goal state, and any constraints or requirements that must be satisfied. It provides a structured and declarative way to represent the various elements involved in planning, making it easier for planners and planning algorithms to understand and reason about the problem. As far as PDDL actions are concerned, they have parameters, preconditions and effects that can be also conditional (when-effects). Multi-agent extensions of PDDL and of its successive developments exist [30, 31, 32].

2.3. Integration of Ontologies in BDI Frameworks

The connections between declarative agent technologies and ontologies is very well known and studied [33], also as far as BDI frameworks concerned [34, 35, 36, 37].

Because of its similarity with our final goals, we go into the details of Hypermedea [38], a comprehensive framework built upon JaCaMo, designed specifically for web and web of things environments. It comprises four core components that enable agents to operate effectively within these environments. The first component is a Linked Data module, responsible for discovering and extracting RDF triples that represent statements about the web environment. The second component is an ontology module that leverages the discovered statements to infer implicit information. The third component is a planner, which performs reasoning and analysis to assess the potential consequences of an agent's actions within the environment. Lastly, there is a protocol binding component that facilitates the translation of high-level actions into low-level protocol-specific network operations. To evaluate the performance of Hypermedea, the authors focused on two crucial components: Linked Data navigation and the planning module, both of which involve computationally intensive algorithms. The framework demonstrated its efficiency and effectiveness in navigating Linked Data resources and conducting planning tasks. By combining these components and providing a comprehensive framework for web (of things) agents, Hypermedea offers a powerful solution for developing intelligent agents that can seamlessly operate in web-based environments, making use of available data, reasoning capabilities, and communication protocols. The framework opens up opportunities for the development of sophisticated agent-based systems capable of leveraging the web and web of things resources to achieve complex goals and tasks.

Related with the topic dealt with in this section, the work by Sbodio, Martin, and Moulin [39] introduces an extended agent-based simulation framework that incorporates a comprehensive model of beliefs and enables ask-reply communication using the SPARQL language. The

authors demonstrate how the SPARQL query language can be effectively used for describing and discovering operations of web services. They propose utilizing it to express preconditions, postconditions, and agent goals, enabling the evaluation of SPARQL queries as the foundation for service discovery. This involves checking the truth of preconditions, constructing postconditions based on service execution, and determining if executing a service with those results satisfies an agent's goal. Additionally, the paper highlights the potential of leveraging SPARQL features to optimize the discovery algorithm and create relaxed forms of preconditions and goals for use in discovery.

3. Designing Procedural Ontologies

The analysis of the related work presented in section 2, the study of the PLS ontology, and the identification of real-world application domains that may contain procedural knowledge, all support our decision to represent procedural knowledge as ontologies. Ontologies allow us to represent knowledge in an unambiguous and organized manner, extracting key concepts to formalize a flexible yet simple and representative structure and their development is supported by standard representation and query languages, and by open-source graphical development editors. In our preliminary experiments to formalize the structure of the procedural ontology we used the OWL 2 language standard [40] and the graphical editor of Protégé [41].

Inspired by behavior trees, where activities may be broken down into sub-activities (indeed, the control flow nodes model a anonymous activity consisting of the execution of all, or one, or a given number of sub-activities specified in lower levels of the tree) and by PDDL, where actions are characterized by preconditions and effects, we designed our Procedural Ontology around the `Activity` concept, which represents the main procedure or protocol. This activity is composed of multiple `SubActivity` that can be viewed as sub-actions or sub-tasks. This hierarchical breakdown allows for complex activities to be decomposed into smaller and more manageable components.

Each sub-activity within the ontology could be associated with one or more `Precondition`, which represent the requirements that must be met before the execution of the sub-activity itself. Preconditions specify the necessary conditions or constraints that need to be fulfilled for a successful execution.

Additionally, the ontology incorporates the concept of `Successor`. A successor represents an activity or sub-activity that follows another one in the procedural workflow. It denotes the next step or task that should be executed after the ending of the preceding sub-activity to complete the main activity.

Sub-activities may be optional, and have many different preconditions and many successors, hence generating a behavior graph, rather than a behavior tree. However, cycles are not allowed. Activities have a `Priority` corresponding to their level in the tree induced by the activity graph, as if the graph was visited following a breadth-first strategy. The priority of a direct sub-activity of a with priority p is $p + 1$. `Priority` is needed to make explicit which sub-activities are children of what.

Atomic sub-activities (namely, actions) are just identified via their name in the activity graph; complex sub-activities that require further breakdown (namely, complex procedures or

behaviors) are described by an `Activity` concept with the same name as the sub-activity.

By incorporating these elements into the ontology, we aim to provide a structured representation of procedural workflows, facilitating a clear understanding of the dependencies and order of execution within a procedure or protocol.

Figure 3 presents the graph representation of the ontology. Within the yellow rectangle we highlighted the corresponding OWL classes of the previously discussed concepts.

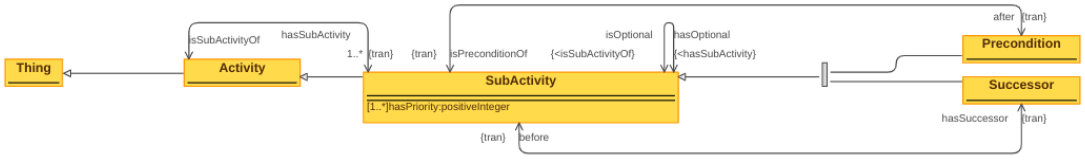

Figure 3: Representation of the Procedural Ontology.

An example of instantiation of the Procedural Ontology is illustrated in Figure 3, on a domain we are all familiar with. In order to `cookPasta` we need to `boilWater`, `addSalt` and `addPasta`. While we `waitCooking`, we should `prepareSeasoning`. Differently from PLS, we do not support the notion of duration of an event, which would instead be extremely useful is this domain, to avoid overcooked pasta! Despite the lack of notion of time and duration, we can express the need to prepare seasoning while the pasta is boiling, by setting `addPasta` as a precondition of both `waitCooking` and `prepareSeasoning`, and by setting `prepareSeasoning` and `drainPasta` as necessary preconditions for `seasonPasta`. One sub-activity with no successor represents the last action or procedure of the given activity. As shown in Figure 3, `prepareSeasoning` is not an atomic action, and it is hence described by an activity.

4. Integrating Procedural Ontologies into VEsNA

In this section we discuss the requirements that a VEsNA extension should feature to properly and fruitfully integrate procedural ontologies.

- **Persistence of Semantic Information**: VEsNA agents must be able to access the semantic information contained in ontologies.
  Typically, ontologies rely on web-published resources but this can lead to potential service disruptions and unavailability of the agent functionalities. We identify the need of a persistent storage in order to enhance availability, reliability and consistency of the agent's access to ontological data, contributing to a more robust and uninterrupted functioning.
- **Processing and Reasoning**: the agent should have the capability to process ontologies and employ reasoning to derive new knowledge.
  Moreover, it must be able to distinguish its own standard plan-driven reasoning from the reasoning on ontological knowledge, in order to handle situations where erroneous inferences or logical inconsistencies due to incorporation of ontologies may occur.

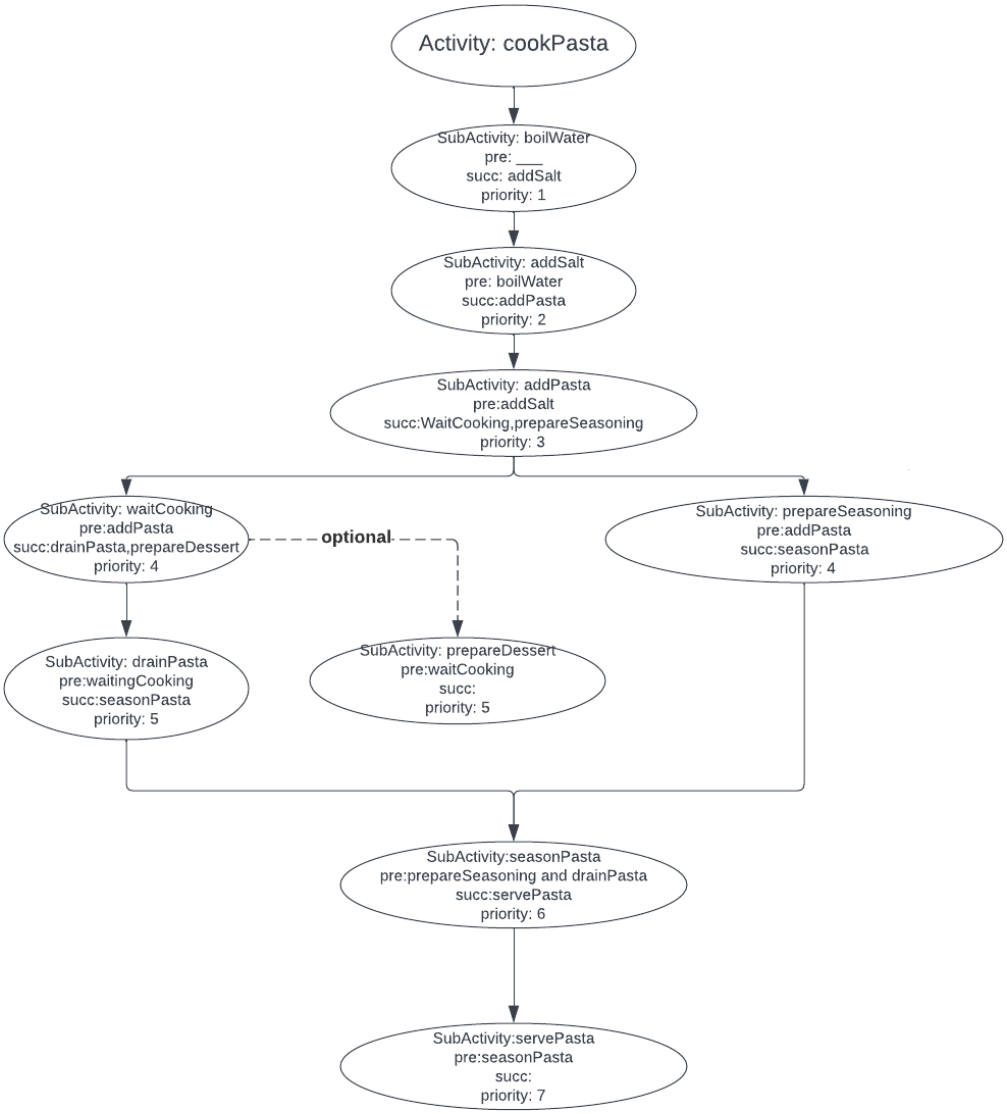

Figure 4: cookPasta activity.

- **Information Removal**: the agent must be able to forget incorrect, outdated, or unnecessary information coming from ontologies.
- **Ontological-to-Beliefs Concepts Association**: since a VEsNA agent is expected to interact with humans via the Rasa interface, the agent must be capable of mapping relevant words extracted from users' sentences to known ontological concepts if it recognizes a relation with the knowledge it has acquired or learned previously.
- **Procedural Knowledge in Means-End Reasoning**: the agent should actively utilize the procedural knowledge learned from ontologies in the process of means-end reasoning. The process involves deciding how to achieve an intention by utilizing the available means. In this context, the agent can collaborate with a human user to achieve their goals

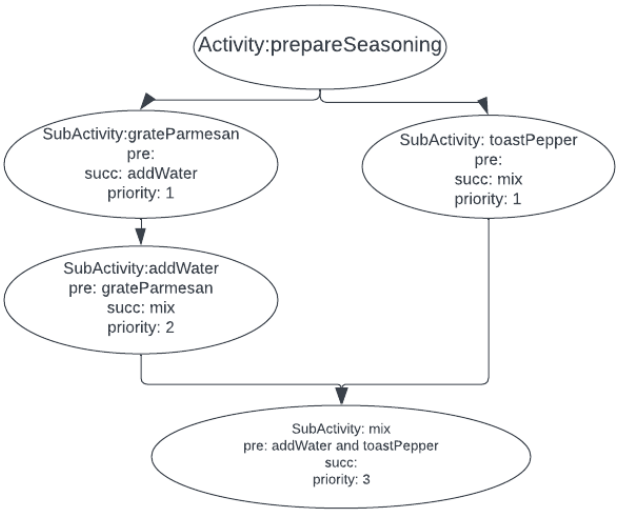

Figure 5: prepareSeasoning activity.

by suggesting a tailored procedure or plan.

During our analysis on the agent's needs, we have identified three levels of persistence that can be implemented in Jason:

- **Mental State**: the agent should be able to recall and retain its acquired knowledge and beliefs, up to the point of interruption.
  Similar to how humans retain their learned knowledge after waking up from sleep, this persistence of the mental state allows for continuity of the agent's cognitive processes. In order to accomplish this, we may declare a belief base file in our agent project configuration (referred to as `<agent_name>.bb`).
- **Ontological Knowledge**: in order to store the learned procedural ontologies, we may need a semantic database; we are considering Virtuoso [42] - in particular its Docker image [43] - for storing ontological knowledge ("ontological belief base").
- **Generated Plan**: during the implementation phase, we intend to utilize Jason pre-processing directives to make the plans that the agent can generate on the fly from the procedural knowledge contained in the procedural ontology persistent. This approach establishes an agent cache used to save the generated plans, ensuring fault tolerance and enhancing the agent's performance and responsiveness.

To accomplish the previously summarized features that the agent must exhibit, we have devised the following new Jason internal actions as necessary for handling ontologies:

- **Memorize**, allowing the agent to interface with ontologies published on the web and store them in the agent's ontological belief base.
- **Inspect**, allowing the agent to associate a given a term or word uttered by the human user and recognized by Rasa with an ontological concept, if a similarity with the concepts inside its ontological belief base is found.

- **Forget**, to remove ontologies from the agent's belief base. This capability is useful when the agent determines that certain ontologies are no longer reliable or relevant to its current objectives.
- **Infer**, to retrieve ontologies from the ontological belief base, apply reasoning techniques, and save the inferred ontologies back into the storage. This process allows the agent to enrich the original information with new deductions, grounding the results of the reasoning and making them persistent.
- **Reason**, that represents the most substantial and challenging functionality of the agent. This action enables the agent to generate SPARQL queries automatically by mapping the beliefs expressed in agent speak code.
  This internal action plays a key role in the system, as it allows the agent to generate plans based on data retrieved from the ontological belief base.

Note that Jason internal actions are executed within the agent itself, unlike normal actions that are executed outside the agent's mind and have the ability to change the environment. They offer an AgentSpeak code interface and are implemented using Java in the back end. Accessing ontologies from inside Java code can be done by making use of specific APIs such as Jena [44] and OWL API [45].

While in our preliminary design we devise the actions above as internal actions for the Jason implementation of VEsNA agents, we are also considering to implement them as functions offered by CArtAgO artefacts [46] inside JaCaMo, possibly taking advantage of the services that Hypermedea [38] already offers.

In none of the cases above, any change to the BDI architecture and interpreter would be required at all: Jason agents should use the **Memorize**, **Inspect**, **Forget**, **Infer**, **Reason** internal actions (resp. functions implemented by artefacts) as they use any other internal action (function).

5. Conclusions and Future Work

In this paper we have presented a study on the integration of procedural ontologies and cognitive agents with the ultimate goal of enhancing intelligent decision-making in VEsNA. This would make VEsNA agents able to properly operate in complex, knowledge-intensive domains, still retaining their capability to understand natural language statements thanks to the Rasa frontend, and sense and act into a virtual world thanks to the Godot backend.

While our work on procedural ontologies is still in its preliminary stages, VEsNA is becoming more and more solid and robust. Enabling VEsNA agents to effectively utilize and reason about semantic procedural data foundation would dramatically extend their capabilities, and might also represent a foundation for better understanding the cognitive agents' planning activity in the metaverse, namely a virtual world where humans may interact using natural language.

Besides, of course, refining the design of procedural ontologies and on the extension of VEsNA agents to cope with them, and implementing the designed extension, there are several routes for further work that could be explored. These include:

1. Integration of Procedural and Descriptive Ontologies: one potential area of improvement is the integration of the procedural ontology with a purely descriptive static ontology.

This integration would provide agents with a more precise understanding of users' actions - or generally, more precise information on the user itself - and allows for more accurate framing of user's intents within the procedural context.

2. Enhanced Precondition Matching: the matching of preconditions can be further refined, particularly in the domain of procedural knowledge. Exploring methods to verify the matching of preconditions more accurately, even when they involve facts that do not directly represent an activity within the procedure, would enhance the agent's decision-making capabilities.
3. Agent Proactiveness: an interesting direction for further work is to make the agent more proactive in gathering information from users. For example, the agent could autonomously request information on the performance of optional activities, leading to a more comprehensive and accurate understanding of the user's intent.
4. Refined Rasa Pipeline: tuning the Rasa pipeline could be explored to improve the recognition of relevant information within user sentences during conversations. Fine-tuning the pipeline would enable the semantic agent to extract and utilize the most pertinent parts of the user's input, leading to more effective and context-aware responses.
5. Integration with LLM Models: evaluating the integration of large language models (LLMs) - such as ChatGPT [47] - could be another avenue of research. By incorporating LLMs into the system, the agent's output could be refined to generate responses that are even closer to natural language, further enhancing the user experience and interaction.

Acknowledgments

This work was partially supported by the "ENGINES — ENGineering INtElligent Systems around intelligent agent technologies" project funded by the Italian MUR program "PRIN 2022" under grant number 20229ZXBZM.

References

[1] N. R. Jennings, K. P. Sycara, M. J. Wooldridge, A roadmap of agent research and development, Auton. Agents Multi Agent Syst. 1 (1998) 7–38.

[2] A. S. Rao, M. P. Georgeff, BDI agents: From theory to practice, in: ICMAS, The MIT Press, 1995, pp. 312–319.

[3] H. Wang, H. Ning, Y. Lin, W. Wang, S. Dhelim, F. Farha, J. Ding, M. Daneshmand, A survey on the metaverse: The state-of-the-art, technologies, applications, and challenges, IEEE Internet Things J. 10 (2023) 14671–14688.

[4] A. Gatti, V. Mascardi, VEsNA, a framework for virtual environments via natural language agents and its application to factory automation, Robotics 12 (2023) 46.

[5] T. Bocklisch, J. Faulkner, N. Pawlowski, A. Nichol, Rasa: Open source language understanding and dialogue management, CoRR abs/1712.05181 (2017). URL: http://arxiv.org/abs/1712.05181. arXiv:1712.05181.

[6] Rasa technologies, Rasa web site, 2024. URL: https://rasa.com/, accessed on June 24, 2024.

[7] Godot foundation, Godot web site, 2024. URL: https://godotengine.org/, accessed on June 24, 2024.
[8] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, Multi-agent oriented programming: programming multi-agent systems using JaCaMo, MIT Press, 2020.
[9] R. H. Bordini, J. F. Hübner, M. J. Wooldridge, Programming multi-agent systems in AgentSpeak using Jason, J. Wiley, 2007.
[10] Z. N. Esfahani, D. C. Engelmann, A. Ferrando, M. Margarone, V. Mascardi, Integrating ontologies and cognitive conversational agents in On2Conv, in: EUMAS, volume 14282 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 66–82.
[11] P. F. Patel-Schneider, Partial reasoning in knowledge representation systems based on description logics, in: Description Logics, AAAI Fall Symposium Series, AAAI Press, 1992, pp. 74–75.
[12] G. D. Giacomo, M. Lenzerini, Tbox and abox reasoning in expressive description logics, in: Description Logics, volume WS-96-05 of *AAAI Technical Report*, AAAI Press, 1996, pp. 37–48.
[13] D. Calvanese, Finite model reasoning in description logics, in: Description Logics, volume WS-96-05 of *AAAI Technical Report*, AAAI Press, 1996, pp. 25–36.
[14] D. Calvanese, A. Gianola, A. Mazzullo, M. Montali, SMT safety verification of ontology-based processes, in: AAAI, AAAI Press, 2023, pp. 6271–6279.
[15] J. Sudeikat, L. Braubach, A. Pokahr, W. Lamersdorf, W. Renz, Validation of BDI agents, in: PROMAS, volume 4411 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 185–200.
[16] B. Archibald, M. Calder, M. Sevegnani, M. Xu, Modelling and verifying BDI agents with bigraphs, Sci. Comput. Program. 215 (2022) 102760.
[17] M. Xu, T. Rivoalen, B. Archibald, M. Sevegnani, Can-verify: A verification tool for BDI agents, in: iFM, volume 14300 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 364–373.
[18] M. Colledanchise, P. Ögren, Behavior Trees in Robotics and AI: An Introduction, CRC Artificial Intelligence and Robotics Series, Chapman & Hall, 2018.
[19] M. Mateas, A. Stern, A behavior language for story-based believable agents, IEEE Intell. Syst. 17 (2002) 39–47.
[20] I. Millington, J. Funge, Artificial Intelligence for Games, Second Edition, Morgan Kaufmann, 2009.
[21] L. Cirocco, D. Jarvis, J. Jarvis, R. Rönnquist, GORITE: A BDI realisation of behavior trees, in: ICA, IEEE, 2022, pp. 6–11.
[22] P. Misteli, S. Poulakos, M. Kapadia, R. W. Sumner, Towards emergent play in mixed reality, International SERIES on Information Systems and Management in Creative eMedia (CreMedia) (2018) 51–56.
[23] M. Kapadia, J. Falk, F. Zünd, M. Marti, R. W. Sumner, M. H. Gross, Computer-assisted authoring of interactive narratives, in: I3D, ACM, 2015, pp. 85–92.
[24] J. Hanák, P. Chudỳ, J. Vlk, Collaborative agents for synthetic tactical training, in: 2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC), IEEE, 2023, pp. 1–9.
[25] J. Turi, Planning from operational models for deliberate acting in Robotics, Ph.D. thesis, INSA de Toulouse, 2024.
[26] E. Safronov, M. Colledanchise, L. Natale, Task planning with belief behavior trees, in:

IROS, IEEE, 2020, pp. 6870–6877.
[27] C. Schlenoff, M. Grüninger, F. Tissot, J. Valois, J. Lubell, J. W. Lee, The Process Specification Language (PSL) Overview and Version 1.0 Specification, Technical Report, 2000.
[28] M. Grüninger, C. Menzel, The process specification language (PSL) theory and applications, AI Mag. 24 (2003) 63–74.
[29] AIPS-98 Planning Competition Committee, PDDL–The Planning Domain Definition Language, Technical Report, 1998.
[30] M. Brenner, A multiagent planning language, in: Proc. of ICAPS'03 Workshop on PDDL, volume 3, 2003.
[31] D. L. Kovacs, A multi-agent extension of PDDL3.1, in: ICAPS 2012 Proceedings of the 3rd Workshop on the International Planning Competition (WS-IPC 2012), 2012, pp. 19–37.
[32] D. Borrajo, S. Fernández, Efficient approaches for multi-agent planning, Knowl. Inf. Syst. 58 (2019) 425–479.
[33] V. Mascardi, J. A. Hendler, L. Papaleo, Semantic web and declarative agent languages and technologies: Current and future trends - (position paper), in: DALT, volume 7784 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 197–202.
[34] Á. F. Moreira, R. Vieira, R. H. Bordini, J. F. Hübner, Agent-oriented programming with underlying ontological reasoning, in: DALT, volume 3904 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 155–170.
[35] V. Mascardi, D. Ancona, M. Barbieri, R. H. Bordini, A. Ricci, CooL-AgentSpeak: Endowing AgentSpeak-DL agents with plan exchange and ontology services, Web Intell. Agent Syst. 12 (2014) 83–107.
[36] A. Freitas, A. R. Panisson, L. Hilgert, F. Meneguzzi, R. Vieira, R. H. Bordini, Integrating ontologies with multi-agent systems through cartago artifacts, in: WI-IAT (2), IEEE Computer Society, 2015, pp. 143–150.
[37] D. Vachtsevanou, B. de Lima, A. Ciortea, J. F. Hübner, S. Mayer, J. Lemée, Enabling BDI agents to reason on a dynamic action repertoire in hypermedia environments, in: AAMAS, ACM, 2024, pp. 1856–1864.
[38] V. Charpenay, A. Zimmermann, M. Lefrançois, O. Boissier, Hypermedea: A framework for web (of things) agents, in: Companion Proceedings of the Web Conference 2022, WWW '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 176–179. URL: https://doi.org/10.1145/3487553.3524243. doi:10.1145/3487553.3524243.
[39] M. L. Sbodio, D. Martin, C. Moulin, Discovering semantic web services using SPARQL and intelligent agents, Journal of Web Semantics 8 (2010) 310–328. URL: https://www.sciencedirect.com/science/article/pii/S1570826810000533. doi:https://doi.org/10.1016/j.websem.2010.05.002, semantic Web Challenge 2009 User Interaction in Semantic Web research.
[40] P. Patel-Schneider, M. Krötzsch, P. Hitzler, B. Parsia, S. Rudolph, OWL 2 Web Ontology Language Primer (Second Edition), W3C Recommendation, W3C, 2012. Https://www.w3.org/TR/2012/REC-owl2-primer-20121211/.
[41] Stanford University, Protégé, https://protege.stanford.edu/, 2024. Last Accessed: June 24, 2024.
[42] O. Erling, I. Mikhailov, Virtuoso: Rdf support in a native rdbms, in: Semantic Web Information Management, 2009.

[43] OpenLink Software, Inc., OpenLink Virtuoso Open Source Edition 7.2 Docker Image, 2024. URL: https://hub.docker.com/r/openlink/virtuoso-opensource-7/, last Accessed: June 24, 2024.
[44] The Apache Software Foundation, Apache Jena, 2024. URL: https://jena.apache.org/, last Accessed: June 24, 2024.
[45] owlcs, OWL API, 2024. URL: https://github.com/owlcs/owlapi, last Accessed: June 24, 2024.
[46] A. Ricci, M. Viroli, A. Omicini, CArtAgO : A framework for prototyping artifact-based environments in MAS, in: E4MAS, volume 4389 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 67–86.
[47] Open AI, Introducing ChatGPT, 2022. URL: https://openai.com/blog/chatgpt, accessed on June 24, 2024.