A Framework for Developing Interactive Intelligent Systems in Unity

Andreas Brännström[0000−0001−9379−4281] and Juan Carlos Nieves[0000−0003−4072−8795]

Department of Computing Science
Umeå University
SE-901 87, Umeå, Sweden
{andreasb, jcnieves}@cs.umu.se

Abstract. This paper introduces a lightweight framework for implementing intelligent interactive systems (IIS). In particular, systems that integrate symbolic knowledge bases for reasoning, planning and rational decision-making in interactions with humans. This is done by integrating Web Ontology Language (OWL)-based reasoning and Answer Set Programming (ASP)-based planning software. In order to provide interactive user components, the framework is encompassed in a widely used game development tool, Unity. The proposed framework, UnityIIS, is the first approach for integrating OWL together with ASP in Unity. Its central functionalities for knowledge representation and knowledge revision is presented together with an example application created in the framework. A chatbot agent, embodied in Augmented Reality, is designed, following a Belief, Desire, Intention (BDI) agent model. The set of tools that the framework provides can be applied for developing IIS research prototypes as well as being an asset in teaching practices.

Keywords: Software frameworks · Symbolic reasoning · Agent modeling · Multi-agent systems · Game development.

1 Introduction

Along with advances in artificial intelligence (AI) methods and related technologies, the emergence of Internet of Things (IoT), and the widespread use of wearable smart devices with advanced sensor technology, there is an increased interest in designing and developing Intelligent and Interactive Systems (IIS) [24]. The IIS research field studies interactions between humans and intelligent systems, to develop intelligent digital artifacts that can interact with humans and their environment (increasingly commonly in Mixed-Reality environments, using Virtual Reality (VR) or Augmented Reality (AR) technology) to personalize and support human activities. However, designing and developing intelligent interactive systems are challenging due to that systems usually require composite multi-modal [25] and multi-agent system (MAS) [8] architectures that integrates several types of intelligent technologies, together with interactive components

directed toward the human users. A multi-modal and multi-agent architecture is required in order to model complex human domains that often need to be analyzed in different ways, in terms of, e.g., context reasoning, planning, and rational decision-making, interwoven with a variety of learning components (e.g., as input models or weight revisions). Such diverse operations may require multiple reasoning and planning technologies that are executed in a sequence or in parallel. For instance, context reasoning may lead to goal recognition, which in turn guides a planning procedure.

A variety of tools for developing agent-based systems are available (e.g., JaCaMo [4], JADE [3], Tweety [23]). However, integrating agent technologies with human interaction components (e.g., 2D, 3D or Mixed-Reality interfaces) can pose a technical overhead. Research communities and teaching practices need practical tools and frameworks that support implementing IIS research prototypes, providing functionalities for multi-modal reasoning architectures [17].

This paper introduces a framework for implementing intelligent interactive systems that integrates symbolic knowledge bases for reasoning, planning and rational decision-making in interactions with humans. While there is a variety of technologies and tools available, the framework is centered on three main technologies: Web Ontology Language (OWL) [14], Answer Set Programming (ASP) [12], and the Unity Game Engine [20].

OWL, originally developed for the Semantic Web to ensure a common understanding of information on the Web, has been applied in different areas for representing and sharing knowledge. In the setting of IIS, ontologies are useful for providing personalized human interactions, e.g., by reasoning with knowledge from different sources and modalities, to create cooperative agent technologies across Web-sources, people, (IoT) devices, and physical-virtual spaces.

Answer Set Programming (ASP) is a well-known declarative programming paradigm which has its roots in logic programming and non-monotonic reasoning to solve difficult, typically NP-hard, search problems, useful for different knowledge representation and reasoning tasks. In the setting of IIS and in the proposed framework, we particularly highlight ASP-based planning and action reasoning [12].

The proposed framework integrates OWL and ASP with a widely used game development tool, the Unity Game Engine, that previously has been used for multi-agent applications [2, 19]. Unity provides a wide range of functionalities to develop interactive games and applications, for 2D, 3D and Mixed-Reality interfaces, which can be deployed on a variety of (mobile) devices. We introduce 'Unity Interactive Intelligent Systems' (UnityIIS) framework which enables operations such as create, query, update and delete on OWL-ontologies and ASP-programs, which support implementation of multi-modal reasoning architectures with functionalities such as knowledge revision, goal deliberation, action reasoning and planning. The framework has been applied to develop various prototypes [1, 5] and is accessible online for academic and research purposes[1].

[1] https://git.io/JMpC6

The rest of this paper is organized as follows. In Section 2, we summarize previous frameworks for implementing agent and MAS-based interactive intelligent systems. In Section 3, we introduce the UnityIIS framework and its main components. Section 4 presents an example application, a chatbot agent, based on the framework. Section 5 and 6 concludes the paper, discussing potentials, limitations and directions for future work.

2 Related Work

There is a large number of tools available for implementing intelligent systems, utilizing agent and multi-agent system (MAS) technologies [10] (e.g., JaCaMo [4], JADE [3], JACK Intelligent Agents [7], MadKIT [13], and Tweety [23]). However, these tools are often large and overwhelming for small-scale applications and projects. Furthermore, these tools often require a high technical overhead when developing systems that involve human interactions, i.e., where (graphical) user interface components must be integrated with the agent-components. This becomes an issue in, e.g., teaching practices, when engaging students in development of interactive intelligent systems, or when developing small-scale research prototypes. A way to approach this is to integrate different agent reasoning components with an interactive game development environment, such as Unity. Within the Unity platform, a variety of frameworks and libraries have been introduced to aid Unity game developers with agent-based functionalities. These frameworks have mostly centered on Machine Learning (ML)-based approaches. For instance, the Machine Learning Agents (ML-Agents) toolkit [15] was introduced for Unity. The ML-Agents toolkit simplifies integration of reinforcement learning, imitation learning agents as well as scripted agents into games developed in Unity.

When it comes to the area of rational agents and logic programming, there are a set of interesting tools available. EmbASP [9] is a framework for integrating logic programming in external systems for generic applications. Similar to the present paper, EmbASP is created to help developers to integrate complex reasoning tasks in applications through logic-based solvers. EmbASP is a prominent tool for integrating logic programming in Unity. However, the software is currently limited to desktop and Android devices, leaving out a variety of mobile, VR and AR devices. Furthermore, it lacks support for OWL ontologies.

A widely used tool for integrating logic programming into external systems is Tweety [23]. Tweety is a Java library that provides a general interface layer for doing research and working with different logical formalisms and knowledge representation approaches. Similar to UnityIIS, through Tweety, different logic based approaches can be integrated in external systems, such as Unity. While Tweety has a rich selection of functionalities, the Java-based approach requires the programmer to develop the integration to Unity, which can constitute a large technical overhead, especially for smaller research prototypes.

In order to decrease this overhead for research prototypes in Unity, we need lightweight frameworks that in a plug-and-play manner integrates symbolic rea-

soning approaches, such as Answer Set Programming and OWL ontology reasoning into Unity. This is where we see the main benefits of the introduced framework, combining relevant tools for developing interactive intelligent systems that can be deployed on a large span of devices. In addition, due to its lightweight approach, the introduced framework can be integrated with previous frameworks, such as Tweety and ML-Agents, for a broader functionality.

3 The UnityIIS Framework

The UnityIIS framework is divided into modules to integrate varying agent-modeling and knowledge representation tools. The main modules are 1) the 'OWL Unity Package' to enable ontology reasoning, and 2) the 'ASP Unity Package' to enable ASP-based action reasoning and planning. The modules can be applied together or individually depending on the needs of the specific application. A set of helper functions is provided for creating, querying and updating OWL-ontologies and ASP-programs. Through the helper functions, procedural updates can be made to an agent's knowledge base. Knowledge revisions are interwoven in the Unity game engine's main update loop (or by using parallel threads). This allows multi-agent architectures with agents that act and interact in parallel. In this way, dynamic logic-based functionality is enabled for building autonomous rational agents in a graphical development environment.

Conceptually, the architecture takes multi-modal inputs from the user, the game environment, and potential sensors. The inputs (processed through suitable input/sensor models) together with current environment variables is collected in a game state data structure. The game state is converter to OWL facts and sent in an OWL ontology file to the ontology reasoner. The ontology inference is processed and converted to ASP facts, sent in an ASP program file to the ASP solver. Answer sets are collected, filtered, and provided as input to application specific operations (see Figure 1).

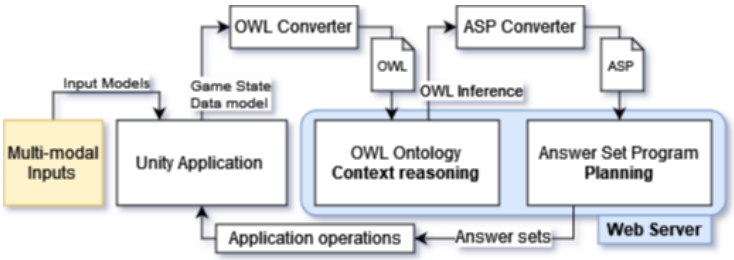


Fig. 1: UnityIIS Conceptual Architecture.

ASP and OWL reasoners are accessed remotely through a Web API (Application Programming Interface). Due to the Web API approach, the developer

does not have to be concerned about system integrations nor system dependencies and can instead focus on application specific development. The Web API solution further allows the system to be deployed on any (mobile) device with internet access. With minor adjustments to the framework, the reasoners can instead be accessed locally on the device. This adaption can be important for some applications where data privacy and external transmissions must be considered. In the following subsections, we present the main components of the framework.

3.1 OWL Ontology Module

The 'OWL Unity Package' integrates an ontology reasoner, BaseVISor [18], to the Unity platform through a Web API. BaseVISor is a forward-chaining inference engine optimized for the processing of RDF triples. BaseVISor supports reasoning services such as realization, classification, satisfiability, conjunctive query answering, entailment, and consistency.

UnityIIS's Web API accesses BaseVISor by running a remote command line execution on a BaseVISor standalone application. In a HTTP call from Unity, rules and/or facts are appended in a BaseVISor (BVR) file. The BVR file is an XML file written in BaseVISor markup language. The BVR file includes references to OWL ontology files. Furthermore, in the BVR file, additional facts can be defined in terms of RDF-triples (⟨subject, predicate, object⟩), and rules (⟨head, body⟩) for complementing and processing ontology inferences.

The most central functions of the 'OWL Unity Package' are presented below, which allows creating, updating and deleting inference rules, queries, facts and BaseVISor inference response filters before connecting to the ontology reasoner:

- **UpdateOWLFile()** - Updates an OWL ontology file. This allows adding, creating, updating and deleting facts or inference rules in the ontology file before connecting to the ontology reasoner.
- **UpdateBVRFile()** - Updates RDF-formatted content in a BVR file. This allows adding, creating, updating and deleting facts, rules, queries and response formats before connecting to the ontology reasoner.
- **OWLConverter()** - Converts input from the Unity application to OWL-format. This allows converting user and environment input into facts, rules, queries, etc.
- **RDFConverter()** - Converts input from the Unity application to BaseVISor RDF-format. This allows converting user and environment input into facts, rules, queries, etc.
- **QueryOntology()** - Starts a request to the ontology reasoner API. The most recent BVR file and OWL file contents are sent in the request. The ontology inference is returned and collected to be further processed in Unity.

3.2 Answer Set Programming Module

The 'ASP Unity Package' integrates an Answer Set Programming (ASP) solver and grounder, Clingo [11], to the Unity platform through a Web API by running

a remote command line execution on a Clingo standalone application. UnityIIS provides a set of helper functions to allow querying ASP knowledge bases, updating facts and inference rules, and retrieving solutions in terms of answer sets. The most central functions are presented below:

- **UpdateASProgram()** - Updates an ASP program file. This allows adding, creating, updating and deleting facts, constraints, action specifications, goal specifications, etc., before connecting to the solver.
- **ASPConverter()** - Converts input from the Unity application to ASP-format. This allows converting user input, environment input, and prior ontology inferences, into ASP-formatted facts, goals, rules, queries, etc.
- **QueryASProgram()** - Connects to a Web API with the ASP solver, Clingo. The most recent ASP program file content is sent in the request. The ASP answer sets are returned and collected to be further processed in Unity.

3.3 Web API to OWL/ASP Reasoners

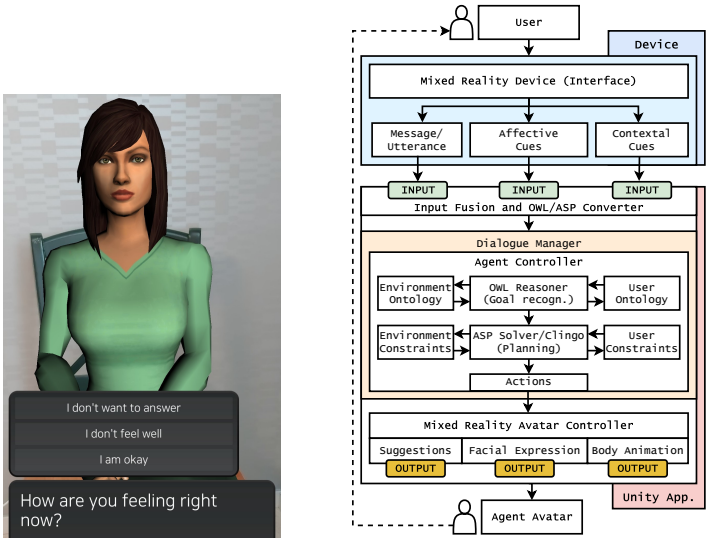
The UnityIIS Web API is accessed through HTTP requests from the UnityIIS application, delivering updated BVR, OWL and ASP files to the server. The server API, written in PHP, receives the HTTP requests and executes server side command line operations to initiate the BaseVISor and Clingo batch programs. Before executing the batch programs, received BVR, OWL and ASP files are temporarily stored on the server to be appended in the reasoning processes. BaseVISor and Clingo batch programs are accessed in separate calls in order to enable individual executions and separate processing of results. This allows, for instance, ontology inferences (e.g., context recognition) to be appended to the ASP program input for further processing (e.g., planning).

4 Example: BDI Chatbot in Augmented Reality

Using the components of the UnityIIS framework, we develop an Augmented Reality (AR) chatbot [5], following a BDI (Belief, Desire, Intention) agent model [22] (see Fig 2-a). Belief is the agent's internal knowledge of its environment, which is updated during the interaction by getting new observations. Desires are goals that the agent aims to fulfill, which are updated during the interaction by reflecting upon new beliefs. Intentions are what goals the agent has chosen to achieve, selected in a deliberation process and used for generating a plan.

The belief of the agent is represented in an OWL ontology, for context reasoning and goal recognition, and in an ASP program, for planning and action reasoning. The UnityIIS framework enables belief revisions (OWL/ASP file updates) at application run time by interweaving the agent's control loop with the Unity game loop. The Unity game loop does cyclic update iterations on a given frequency. During each frame, external inputs are processed, game status is updated, and graphics are redrawn.

Through a chatbot style interaction, user input is collected and converted to OWL and ASP input strings. The OWL input strings are sent to the OWL



(a) AR Chatbot (b) System Architecture

Fig. 2: Example: BDI Chatbot in Augmented Reality.

ontology reasoner to deliberate about contextual goal definitions. The ontology inference is converted to ASP and appended to the ASP input string, which is sent to the ASP solver. The ASP solver generates a plan for interaction, and decides on suitable actions (see Fig 2-b).

5 Discussion

This paper has presented a development framework, UnityIIS, composed of a set of tools to develop agent-based and MAS-based interactive intelligent systems. The underlying Unity platform allows agent modeling components to be integrated with graphical and user interaction components in a modular way. The components of the framework can further be integrated with other packages for Unity for interweaving other AI and agent-based components, such as machine learning (e.g., the ML-Agents toolkit [15]).

UnityIIS is the first development framework that integrates OWL ontology reasoning together with ASP in Unity. UnityIIS is inline with the idea of multi-modal reasoning agents architectures [17]. The framework is aimed to reduce technical overhead when developing interactive intelligent systems, making it practical for implementing research prototypes.

A limitation of the framework regards the general problem when ontologies become larger, which can result in exponentially increased computational complexities for certain operations [16]. This can become an issue for applications with large ontologies that require quick response times. To account for this,

the framework can with minimal changes access reasoning tools through a local command line interface instead of the Web API solution. Furthermore, the lightweight open-ended architecture can be improved, with more strict framework structures (e.g., in line with particular agent meta-models [6, 21, 22]). This would streamline and accelerate the development of certain applications.

6 Conclusion and Future Work

In this paper, we introduce the UnityIIS framework, developed for implementing interactive intelligent systems, powered by OWL ontologies and ASP programs, in Unity. The framework delivers accessible tools for knowledge representation, agent and MAS-based interactive systems. The framework aims to decrease technical overhead (often the case with current, more extensive, frameworks) when developing IIS systems, suitable for developing research prototypes as well as being utilized by teaching practices in the topic of IIS. In future work, we aim to integrate additional logic-based reasoning and knowledge representation tools, as well as adding features to develop different agent models, such as the BDI agent model and different MAS models. UnityIIS will further be extended with functionalities toward dialogue systems.

Acknowledgements This work was partially funded by the Knut and Alice Wallenberg Foundation.

References

1. Andreas, B., Kampik, T., Nieves, J.C.: Towards human-aware epistemic planning for promoting behavior-change. In: Workshop on Epistemic Planning (EpiP)@ ICAPS, Online, October 26-30, 2020 (2020)
2. Becker-Asano, C., Ruzzoli, F., Hölscher, C., Nebel, B.: A multi-agent system based on unity 4 for virtual perception and wayfinding. Transportation Research Procedia **2**, 452–455 (2014)
3. Bellifemine, F., Bergenti, F., Caire, G., Poggi, A.: Jade—a java agent development framework. In: Multi-agent programming, pp. 125–147. Springer (2005)
4. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with jacamo. Science of Computer Programming **78**(6), 747–761 (2013)
5. Brännström, A.: Interactive rational agent embodied in augmented reality, https://people.cs.umu.se/andreasb/index-assets/MIRAI-2021-poster.pdf
6. Brännström, A., Kampik, T., Ruiz-Dolz, R., Taverner, J.: A formal framework for designing boundedly rational agents. In: 14th International Conference on Agents and Artificial Intelligence (ICAART), Online, February 3-5, 2022. vol. 3, pp. 705–714. SciTePress (2022)
7. Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: Jack intelligent agents-components for intelligent agents in java. AgentLink News Letter **2**(1), 2–5 (1999)
8. Calegari, R., Ciatto, G., Mascardi, V., Omicini, A.: Logic-based technologies for multi-agent systems: a systematic literature review. Autonomous Agents and Multi-Agent Systems **35**(1), 1–67 (2021)

9. Calimeri, F., Fuscà, D., Germano, S., Perri, S., Zangari, J.: Fostering the use of declarative formalisms for real-world applications: The embasp framework. New Generation Computing **37**(1), 29–65 (2019)
10. Cardoso, R.C., Ferrando, A.: A review of agent-based programming for multi-agent systems. Computers **10**(2), 16 (2021)
11. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: A user's guide to gringo, clasp, clingo, and iclingo (2008)
12. Gelfond, M., Kahl, Y.: Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach. Cambridge University Press (2014)
13. Gutknecht, O., Ferber, J.: Madkit: A generic multi-agent platform. In: Proceedings of the fourth international conference on Autonomous agents. pp. 78–79 (2000)
14. Hitzler, P.: A review of the semantic web field. Communications of the ACM **64**(2), 76–83 (2021)
15. Juliani, A., Berges, V.P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., et al.: Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627 (2018)
16. Li, Y., Jianhui, Z., Liu, J., Hou, Y.: Matching large scale ontologies based on filter and verification. Mathematical Problems in Engineering **2020** (2020)
17. Mascardi, V., Weyns, D., Ricci, A., Earle, C.B., Casals, A., Challenger, M., Chopra, A., Ciortea, A., Dennis, L.A., Díaz, Á.F., et al.: Engineering multi-agent systems: State of affairs and the road ahead. ACM SIGSOFT Software Engineering Notes **44**(1), 18–28 (2019)
18. Matheus, C.J., Baclawski, K., Kokar, M.M.: Basevisor: A triples-based inference engine outfitted to process ruleml and r-entailment rules. In: 2006 Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML'06). pp. 67–74. IEEE (2006)
19. Mathieu, P., Picault, S.: The galaxian project: A 3d interaction-based animation engine. In: International Conference on Practical Applications of Agents and Multi-Agent Systems. pp. 312–315. Springer (2013)
20. Okita, A.: Learning C# programming with Unity 3D. AK Peters/CRC Press (2019)
21. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the a&a meta-model for multi-agent systems. Autonomous agents and multi-agent systems **17**(3), 432–456 (2008)
22. Rao, A.S., Georgeff, M.: Bdi agents: from theory to practice. In: Proceedings of the First International Conference on Multiagent Systems. vol. 95, pp. 312–319 (1995)
23. Thimm, M.: Tweety: A comprehensive collection of java libraries for logical aspects of artificial intelligence and knowledge representation. In: Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning (2014)
24. Xhafa, F., Patnaik, S., Tavana, M.: Advances in Intelligent, Interactive Systems and Applications: Proceedings of the 3rd International Conference on Intelligent, Interactive Systems and Applications (IISA2018), vol. 885. Springer (2019)
25. Zhao, R., Wang, K., Divekar, R., Rouhani, R., Su, H., Ji, Q.: An immersive system with multi-modal human-computer interaction. In: 2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018). pp. 517–524. IEEE (2018)