Building Multiagent Systems: An Ontology-Driven Approach with SPADE

DSC Europe 2024

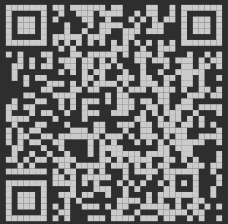
Bogdan Okreša Đurić

University of Zagreb Faculty of Organization and Informatics Artificial Intelligence Laboratory

19 November 2024

Contents

This document and the accompanying material are available online.



Introduction

Introduction

Assistant Professor at the University of Zagreb Faculty of Organization and Informatics, and a member of the Artificial Intelligence Laboratory at UNIZG FOI.

Scientific interests mainly in:

- multiagent systems,
- semantic modelling,
- gamification,
- artificial intelligence,
- computer games.

Introduction

One of the teachers of the following courses in Croatian or English:

- Multiagent Systems,
- Database Theory,
- Declarative Programming,
- Introduction to Artificial Intelligence,
- Introduction to Computer Games,
- Internet Security,
- Computer Game Development Platforms.

Introduction

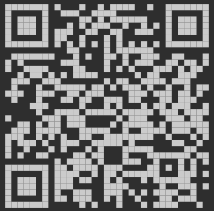
Engaged in international activities and promoting international relations:

- Erasmus student at Karl-Franzens University of Graz (AT),
- Erasmus intern at Jožef Stefan Institute in Ljubljana (SI),
- Erasmus+ intern at Elettra Sincrotrone in Trieste (IT),
- 3-month research stay at Universitat Politècnica de València in Valencia (ES),
- ITEC student at Centre for Development of Advanced Computing in NOIDA (IN),
- 16-month research visit at Universitat Politècnica de València in Valencia (ES),

Introduction

Publications

Project activity



BSc and MA Mentees

Agents and Multiagent Systems

Agents and Multiagent Systems

'[...] we require systems that can decide for themselves what they need to do in order to achieve the objectives that we delegate to them. Such computer systems are known as agents. Agents that must operate robustly in rapidly changing, unpredictable, or open environments, where there is a significant possibility that actions can fail, are known as intelligent agents, or sometimes autonomous agents.' [1]

Agents and Multiagent Systems

What is an Agent?

Agents and Multiagent Systems: What is an Agent?

‘An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.’ [2, p. 54]

Agents and Multiagent Systems: What is an Agent?

‘An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.’ [2, p. 54]

Agents and Multiagent Systems: What is an Agent?

‘An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.’ [2, p. 54]

More specifically,

‘An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to achieve its delegated objectives.’ [1]

Agent

Figure 1: Visual definition of an artificial agent, based on [2, p. 55]

Figure 1: Visual definition of an artificial agent, based on [2, p. 55]

Figure 1: Visual definition of an artificial agent, based on [2, p. 55]

Agents and Multiagent Systems

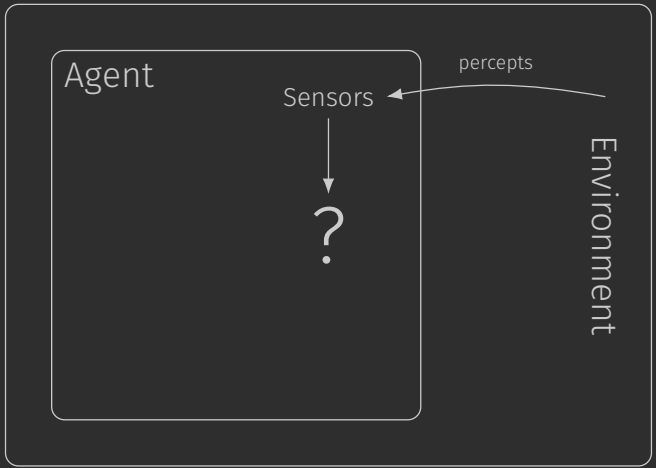


Figure 1: Visual definition of an artificial agent, based on [2, p. 55]

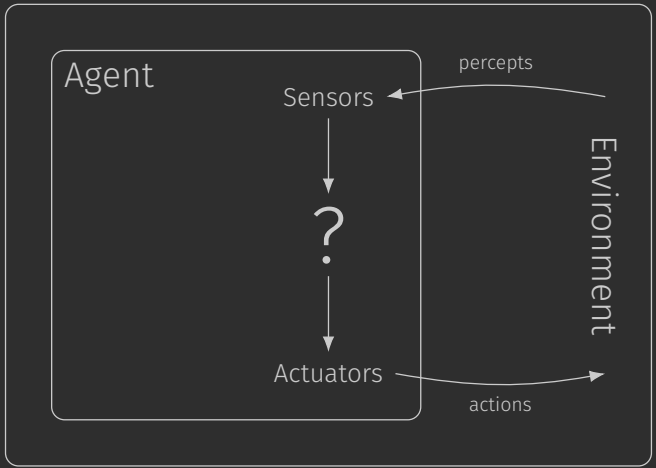


Figure 1: Visual definition of an artificial agent, based on [2, p. 55]

Agents and Multiagent Systems

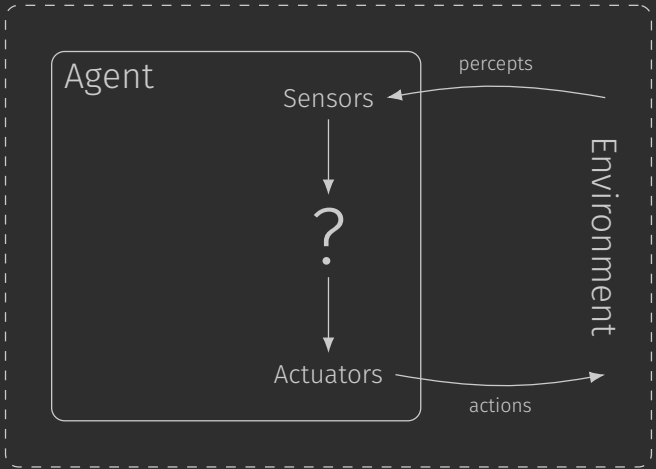


Figure 1: Visual definition of an artificial agent, based on [2, p. 55]

Agents and Multiagent Systems

Agent $\in$ Multiagent System

Agents and Multiagent Systems: Agent $\in$ Multiagent System

A multiagent system (MAS) is a system wherein '[...] an agent must make decisions in environments that contain multiple actors.' [2]

Agents and Multiagent Systems: Agent $\in$ Multiagent System

A MAS is a system wherein '[...] an agent must make decisions in environments that contain multiple actors.' [2]

In more detail, a MAS is a system wherein [3]:

- each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint;
- there is no system global control;
- data are decentralized;
- computation is asynchronous.

Agents and Multiagent Systems: Agent $\in$ Multiagent System

A MAS is a system wherein '[...] an agent must make decisions in environments that contain multiple actors.' [2]

In more detail, a MAS is a system wherein [3]:

- each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint;
- there is no system global control;
- data are decentralized;
- computation is asynchronous.

Agents can be cooperative or competitive. Often modelled as organisations.

Agents and Multiagent Systems

Implementing Agents

Agents and Multiagent Systems: Implementing Agents

Smart Python Agent Development Environment (SPADE) is a multiagent systems implementation platform written in Python and based on instant messaging using extensible messaging and presence protocol (XMPP).

SPADE is a multiagent systems implementation platform written in Python and based on instant messaging using XMPP.

To be run and to communicate, a SPADE agent must be connected to an XMPP server using its JID comprising its name and the hostname of the server, e.g.:

```
alice @ localhost
```

The XMPP server of choice is Prosody.

- `docker-compose.yml` for running Prosody,
- `prosody.cfg.lua` for configuration that requires no user registration,
- generating or using SSL/TLS certificates.

Conda environment with all the required dependencies.

Agents and Multiagent Systems: Implementing Agents

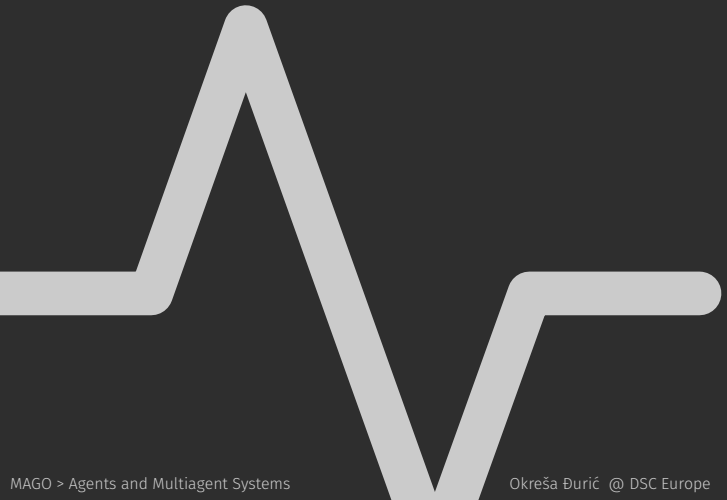
```python
import spade


class DummyAgent(spade.agent.Agent):
    async def setup(self):
        print("Hello, world! I'm agent {}".format(str(self.jid)))


async def main():
    dummy = DummyAgent("alice@localhost", "password")
    await dummy.start()


if __name__ == "__main__":
    spade.run(main())
```

Listing 1: Simple agent



Activity

`agent simple.py`

Agents and Multiagent Systems: Implementing Agents

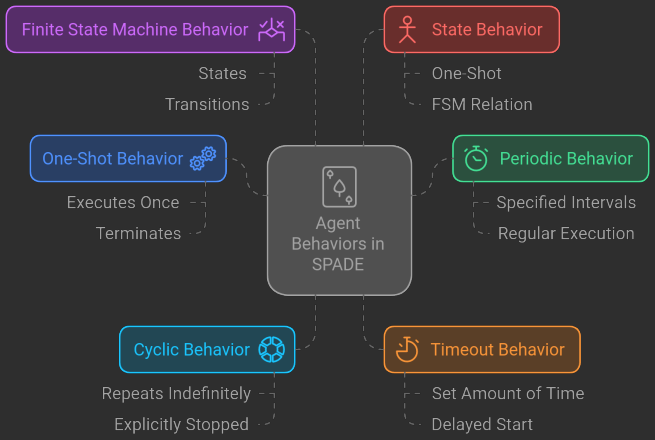


Figure 2: Different types of agent behaviours in SPADE

Agents and Multiagent Systems: Implementing Agents

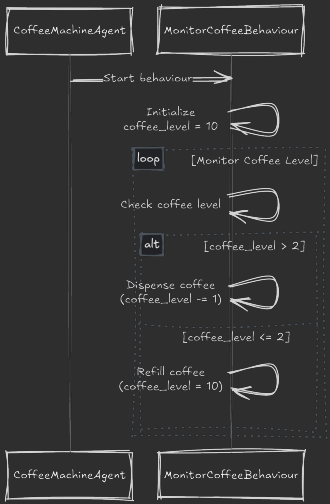
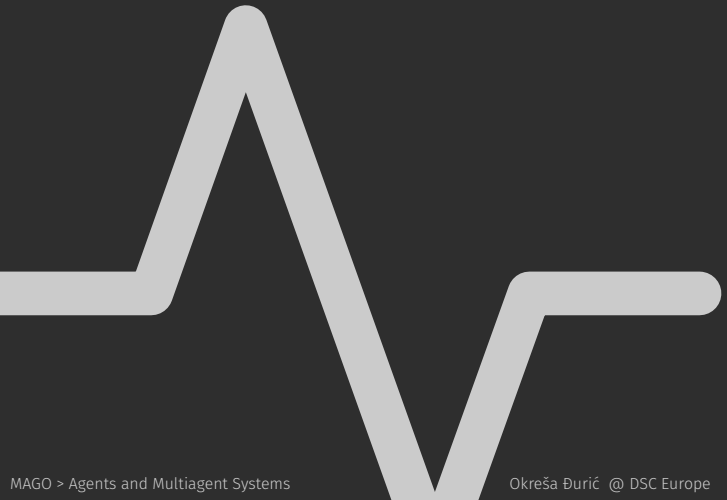


Figure 3: Visual representation of a simple coffee machine agent



Activity

`agent coffee.py`

Agents and Multiagent Systems: Implementing Agents

```
7   class CoffeeMachineAgent(Agent):
8       class MonitorCoffeeBehaviour(CyclicBehaviour):
9           async def dispense_coffee(self):
10              await sleep(1)
11              self.coffee_level -= 1
12
13          async def refill_coffee(self):
14              self.coffee_level = 10
15              print("Coffee refilled to 10 cups.")
16              self.kill()
17
18          async def on_start(self):
19              print("Starting coffee monitoring . . .")
20              self.coffee_level = 10
21
22          async def run(self):
23              print(f"Current coffee level: {self.coffee_level} cups")
24
25              if self.coffee_level <= 2:
26                  print("Low coffee level detected! Refilling coffee . . .")
27                  await self.refill_coffee()
28              else:
29                  print("Sufficient coffee level. Dispensing a cup . . .")
30                  await self.dispense_coffee()
```

Listing 2: The coffee machine agent with a single behaviour

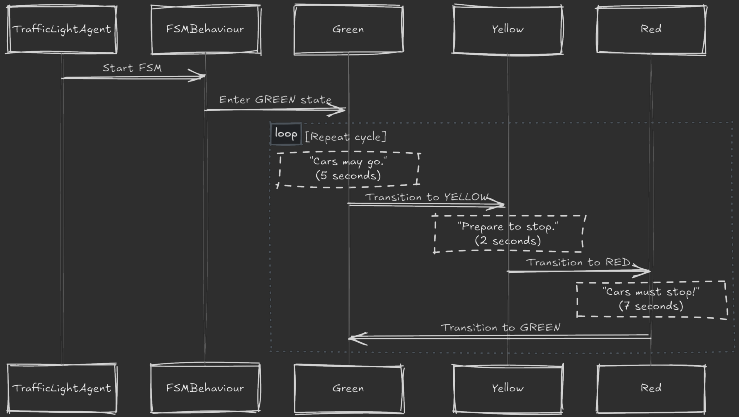
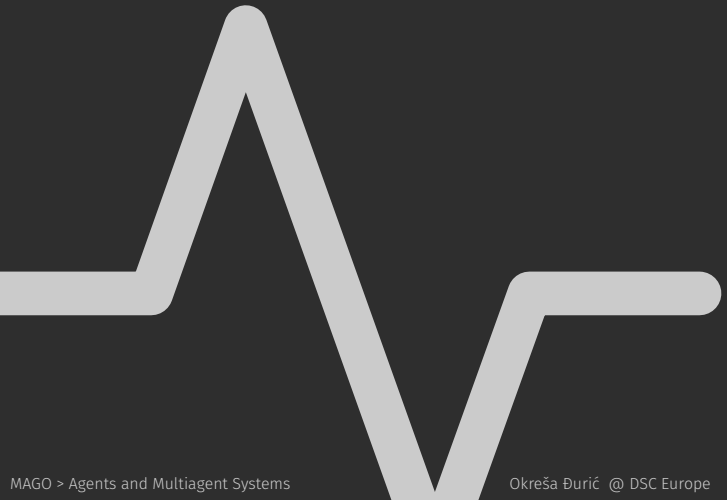


Figure 4: Visual representation of a traffic light agent with a finite state machine behaviour



Activity

`agent traffic light.py`

Agents and Multiagent Systems: Implementing Agents

```python
class TrafficLightFSMBehaviour(FSMBehaviour):
    async def on_start(self):
        print(f"Traffic Light FSM starting at initial state {self.current_state}")

    async def on_end(self):
        print(f"Traffic Light FSM finished at state {self.current_state}")
        await self.agent.stop()


class GreenLightState(State):
    async def run(self):
        print("Green Light: Cars may go!")
        await asyncio.sleep(5)
        self.set_next_state(STATE_YELLOW)


class YellowLightState(State):
    async def run(self):
        print("Yellow Light: Prepare to stop!")
        await asyncio.sleep(2)
        self.set_next_state(STATE_RED)


class RedLightState(State):
    async def run(self):
        print("Red Light: Cars must stop!")
        await asyncio.sleep(5)
        self.set_next_state(STATE_GREEN)


class TrafficLightAgent(Agent):
    async def setup(self):
        fsm = TrafficLightFSMBehaviour()
        fsm.add_state(name=STATE_GREEN, state=GreenLightState(), initial=True)
        fsm.add_state(name=STATE_YELLOW, state=YellowLightState())
        fsm.add_state(name=STATE_RED, state=RedLightState())
        fsm.add_transition(source=STATE_GREEN, dest=STATE_YELLOW)
        fsm.add_transition(source=STATE_YELLOW, dest=STATE_RED)
        fsm.add_transition(source=STATE_RED, dest=STATE_GREEN)
        self.add_behaviour(fsm)
```

Listing 3: Traffic light agent with finite state machine behaviour

Agents and Multiagent Systems: Implementing Agents

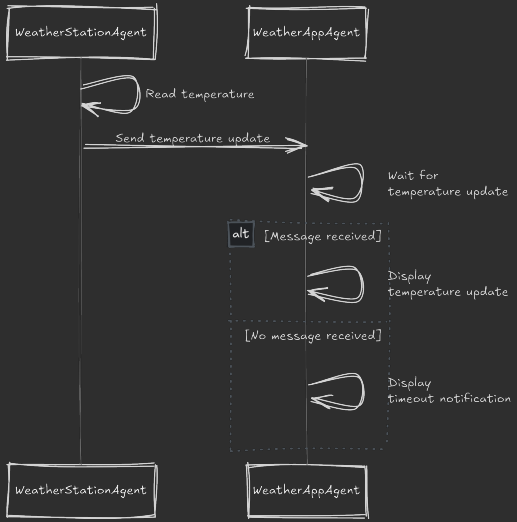
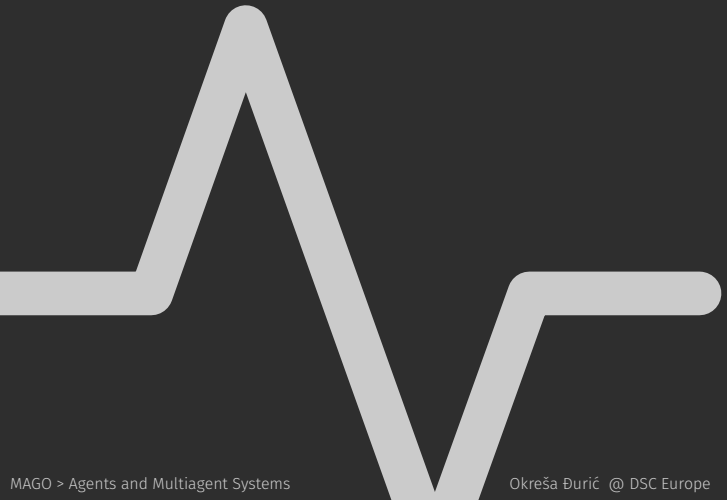


Figure 5: Visual representation of two communicating agents



Activity

`agent weather.py`

Agents and Multiagent Systems: Implementing Agents

```python
class WeatherStationAgent(Agent):
    class SendWeatherUpdateBehaviour(OneShotBehaviour):
        async def run(self):
            temperature = random.uniform(-10, 40)
            msg = Message(to="weather_app@localhost")
            msg.set_metadata("performative", "inform")
            msg.body = f"Current temperature: {temperature:.1f}°C"

            await self.send(msg)
            print(f"Weather update sent: {msg.body}")

            await self.agent.stop()

    async def setup(self):
        behaviour = self.SendWeatherUpdateBehaviour()
        self.add_behaviour(behaviour)
```

```python
class WeatherAppAgent(Agent):
    class ReceiveWeatherUpdateBehaviour(OneShotBehaviour):
        async def run(self):
            print("Waiting for weather update...")
            msg = await self.receive(timeout=10)
            if msg:
                print(f"Weather update received: {msg.body}")
            else:
                print("No weather update received after 10 seconds.")

            await self.agent.stop()

    async def setup(self):
        behaviour = self.ReceiveWeatherUpdateBehaviour()
        template = Template()
        template.set_metadata("performative", "inform")
        self.add_behaviour(behaviour, template)
```

Listing 4: Agent communication example

Ontologies

Ontologies

‘The word ontology means a particular theory of the nature of being or existence. The ontology determines what kinds of things exist, but does not determine their specific properties and interrelationships.’ [2, p. 290]

An ontology is a vocabulary of a chosen domain.

An ontology is a ‘formal specification of a shared conceptualisation’ [4], [5].

An ontology is a knowledge model.

Ontologies

An ontology is often described as a set of triples:

(subject , predicate , object)

where:

- subject and object are usually concepts, their individuals, or literals,
- predicate is an object or a data property.

An example:

(a dog , can play with , a ball)

Ontologies

Common concepts related to ontologies:

Resource description framework (RDF) is a standard for representing data on the web using triples. It helps link and describe information so computers can understand relationships between different concepts.

Web ontology language (OWL) is a language for creating complex models of knowledge (i.e. ontologies) by defining relationships, classes, and rules. It extends RDF to enable reasoning, such as inferring new facts from given data.

Ontologies

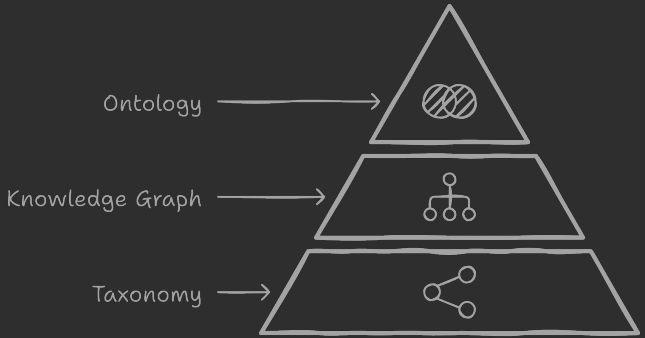


Figure 6: Hierarchical visualisation of information structures taxonomy, knowledge graph, and ontology

Ontologies

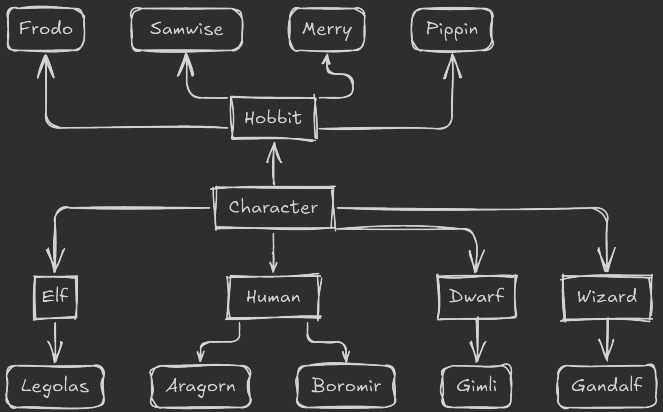


Figure 7: Taxonomy of the selected concepts of the Lord of the Rings novel by J. R. R. Tolkien [6]

Ontologies

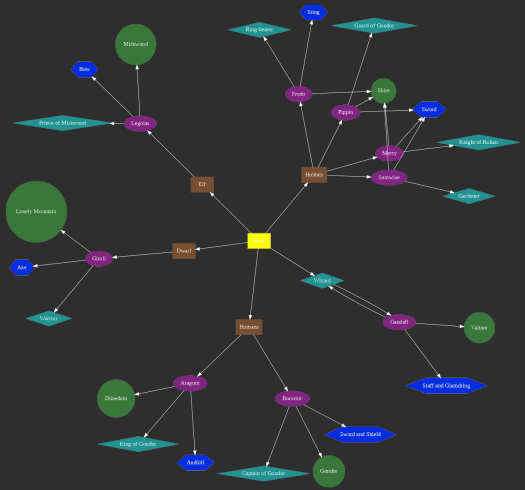


Figure 8: Knowledge graph of the selected concepts of the Lord of the Rings novel by J. R. R. Tolkien

Ontologies

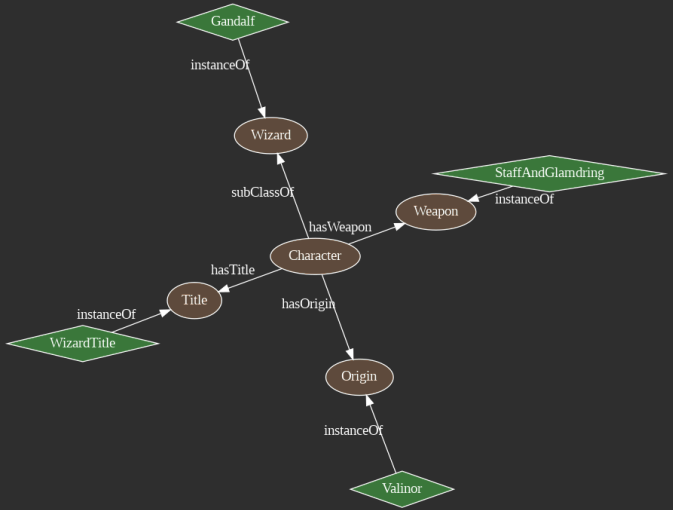


Figure 9: Ontology visualisation of the selected concepts of the Lord of the Rings novel by J. R. R. Tolkien

Ontologies

An ontology is more than a visual representation of related concepts.

Ontologies

```
# Ontology
<http://example.org/lotr> a owl:Ontology .

# Classes
:Character a owl:Class .

:Wizard a owl:Class ;
  rdfs:subClassOf :Character .

:Title a owl:Class .
:Weapon a owl:Class .
:Origin a owl:Class .
```

```
# Object Properties
:hasTitle a owl:ObjectProperty ;
  rdfs:domain :Character ;
  rdfs:range :Title .

:hasWeapon a owl:ObjectProperty ;
  rdfs:domain :Character ;
  rdfs:range :Weapon .

:hasOrigin a owl:ObjectProperty ;
  rdfs:domain :Character ;
  rdfs:range :Origin .
```

```
# Individuals
:Gandalf a :Wizard ;
  :hasTitle :WizardTitle ;
  :hasWeapon :StaffAndGlamdring ;
  :hasOrigin :Valinor .

:WizardTitle a :Title ;
  rdfs:label "Wizard" .

:StaffAndGlamdring a :Weapon ;
  rdfs:label "Staff and Glamdring" .

:Valinor a :Origin ;
  rdfs:label "Valinor" .
```

Listing 5: Ontology serialisation using Turtle syntax

Ontologies

```
# Individuals
:Gandalf a :Wizard ;
  :hasTitle :WizardTitle ;
  :hasWeapon :StaffAndGlamdring ;
  :hasOrigin :Valinor .

:WizardTitle a :Title ;
  rdfs:label "Wizard" .

:StaffAndGlamdring a :Weapon ;
  rdfs:label "Staff and Glamdring" .

:Valinor a :Origin ;
  rdfs:label "Valinor" .
```

```
individual(gandalf).
type(gandalf, wizard).

individual(wizard_title).
type(wizard_title, title).
label(wizard_title, "Wizard").

individual(staff_and_glamdring).
type(staff_and_glamdring, weapon).
label(staff_and_glamdring, "Staff and Glamdring").

individual(valinor).
type(valinor, origin).
label(valinor, "Valinor").

has_title(gandalf, wizard_title).
has_weapon(gandalf, staff_and_glamdring).
has_origin(gandalf, valinor).
```

Listing 6: Ontology individual serialisation using Turtle syntax vs. ontology individual as Prolog expressions

Ontologies

An ontology is a shared vocabulary of a chosen domain.

```
8  <http://example.org/lotr> a owl:Ontology .
```

Ontologies

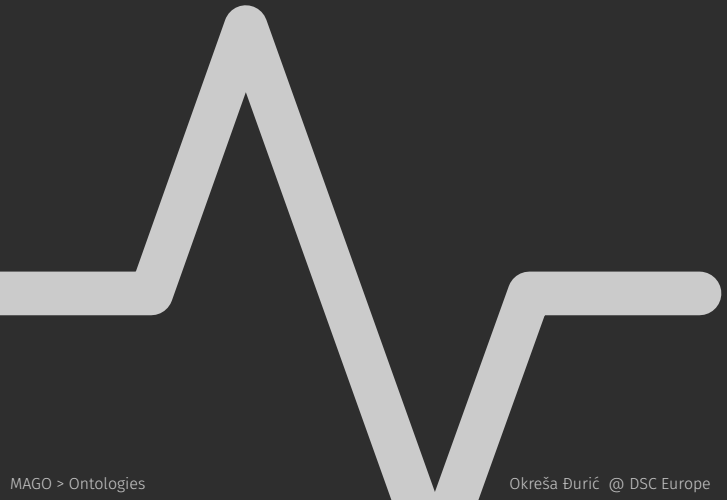
Figure 10: Concepts of the MAGO-Ag ontology

RecipeWorld example

RecipeWorld [7] is an agent-based model comprising three distinct foundational elements: recipes, orders, and agents. In general, agents within recipeWorld can be considered as service providers and service consumers.

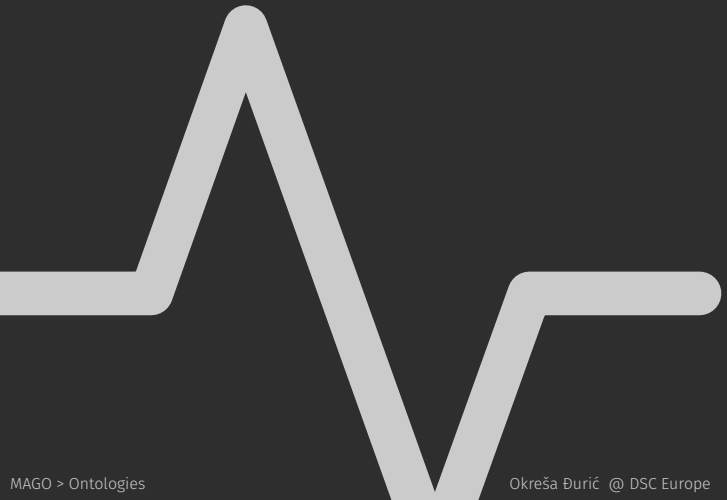
Order agents need services as detailed in their recipes, while factory agents provide a subset of all the services available in the system.

The original framework's goal is to simulate 'the emergence of networks out of a decentralized autonomous interaction.' [7]



Activity

Protégé



Activity

Finalised recipeWorld

Developing a FraMework for Agent Gamification Based on Ontologies (MAGO)

Developing a FraMework for Agent Gamification Based on Ontologies (MAGO)

The result of a cooperation between:

- University of Zagreb Faculty of Organization and Informatics (UNIZG FOI) and
- Universitat Politècnica de València (UPV), Valencian Research Institute for Artificial Intelligence (VRAIN).

The European Union and the Croatian Science Foundation fund this cooperation.

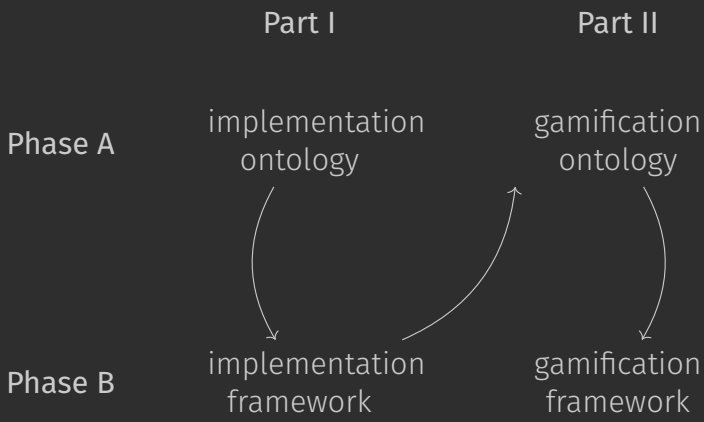


Figure 11: The flow between the parts and the phases of this research

Developing a FraMework for Agent Gamification Based on Ontologies (MAGO)

MAGO-Ag Ontology

MAGO-Ag Ontology

- An ontology comprising concepts applicable to implementing MASs as intelligent virtual environments (IVEs).
- The main goal of the ontology is to enable the modelling of a multiagent system in terms of implementation possibilities.
- The ontology contains a selection of modified and enriched concepts of the MAMbO5 ontology, a result of earlier cooperation [8].
  - e.g. Agent, Behaviour, Action, Process, Objective, Artefact

MAGO-Ag Ontology

Figure 12: Visual relationship of the concepts of the MAGO-Ag ontology

Developing a FraMework for Agent Gamification Based on Ontologies (MAGO)

MAGO-Ag Framework

MAGO-Ag Framework

- The main objective of the MAGO-Ag framework is to translate a MAS modelled using the MAGO-Ag ontology into an implementation template for a MAS comprising SPADE agents.

MAGO-Ag Framework

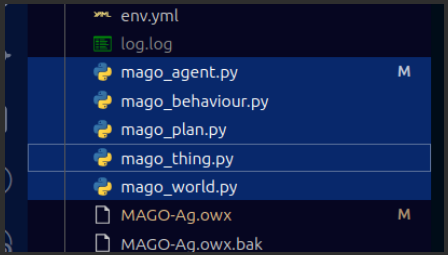


Figure 13: Essential files of the translation process

Activity

MAGO-Ag framework

MAGO-Ag Framework

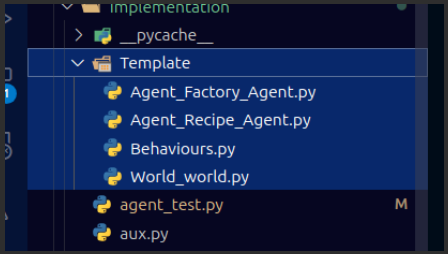


Figure 14: Generated template files for the modelled system

MAGO-Ag Framework

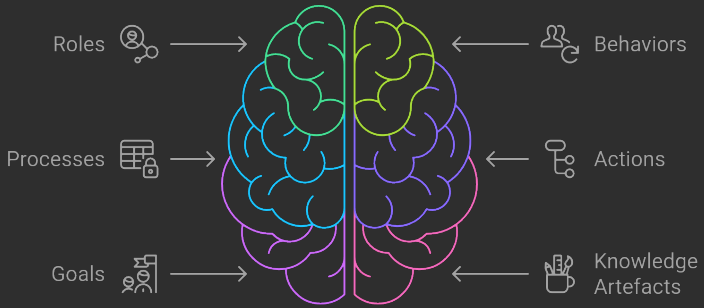


Figure 15: Pieces of knowledge available to agents after template generation

Conclusion

Conclusion

In its presented state, the framework is a work-in-progress package.
Further improvements are seen in:

- rendering strategy-related concepts using languages that allow for reasoning;
- implementing the framework as a distributed system that would focus on deploying agent implementation templates over several workspaces;
- use the MAGO-Ag ontology to facilitate sending agent information and whole agents;
- further testing the framework using different scenarios;
- adapting the implementation templates to different agent development frameworks;
- using the MAGO-Ag ontology in combination with a language model.

Bibliography

Bibliography i

[1] M. Wooldridge, 'Intelligent Agents,' in *Multiagent Systems*, ser. Intelligent Robotics and Autonomous Agents, G. Weiss, Ed., red. by R. C. Arkin, 2nd ed., Cambridge US-MA: The MIT Press, 2013, ISBN: 978-0-262-01889-0.

[2] S. J. Russell and P. Norvig, Eds., *Artificial Intelligence: A Modern Approach* (Pearson Series in Artificial Intelligence), 4th ed. Harlow, UK: Pearson Education Limited, 2022, 1166 pp., ISBN: 978-1-292-40113-3.

[3] V. Dignum, 'Multiagent Organizations,' in *Multiagent Systems*, ser. Intelligent Robotics and Autonomous Agents, G. Weiss, Ed., red. by R. C. Arkin, 2nd ed., Cambridge US-MA: The MIT Press, 2013, ISBN: 978-0-262-01889-0.

[4] N. Guarino, D. Oberle and S. Staab, 'What Is an Ontology?' In *Handbook on Ontologies*, ser. International Handbooks on Information Systems, S. Staab and R. Studer, Eds., red. by P. Bernus, J. Błażewicz, G. Schmidt and M. Shaw, 2nd ed., Berlin DE: Springer, 2009, pp. 1–17, ISBN: 978-3-540-70999-2. DOI: 10.1007/978-3-540-92673-3.

[5] W. N. Borst, 'Construction of Engineering Ontologies for Knowledge Sharing and Reuse,' Ph.D. dissertation, University of Twente, Enschede NL, 05/09/1997, ISBN: 90-365-0988-2.

[6] J. R. R. Tolkien, *The Lord of the Rings*, 50th Anniversary. London: HarperCollinsPublishers, 2007, 1178 pp., ISBN: 978-0-261-10325-2.

[7] M. Fontana and P. Terna, 'From Agent-based models to network analysis (and return): The policy-making perspective,' University of Turin Department of Economics and Statistics "Cognetti de Martiis", Torino IT, 201507, 2015, pp. 1–19.

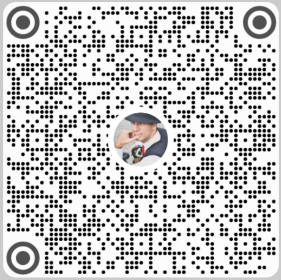
[8] B. Okreša Đurić, J. Rincon, C. Carrascosa, M. Schatten and V. Julian, 'MAMbO5: A new Ontology Approach for Modelling and Managing Intelligent Virtual Environments Based on Multi-Agent Systems,' *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 9, pp. 3629–3641, 2019, ISSN: 1868-5145. DOI: 10.1007/s12652-018-1089-4.

Acknowledgement

Acknowledgement

MOBODL-2023-08-5618
This project was funded by the European Union and the Croatian Science Foundation.

Bogdan Okreša Đurić
dokresa@foi.unizg.hr



University of Zagreb Faculty of Organization and Informatics Artificial Intelligence Laboratory
ai.foi.hr