

From Programming Agents to *Educating* Agents

A Jason-based Framework for Integrating Learning in the Development of Cognitive Agents

Michael Bosello Alessandro Ricci

Alma Mater Studiorum – Università di Bologna
Department of Computer Science and Engineering, Cesena Campus

EMAS 2019

Outline

- 1 Introduction
- 2 Reinforcement Learning Basic Concepts
- 3 Related Works
- 4 BDI-RL Model
 - Hard Plans and Soft Plans
 - Model
- 5 Jason Implementation
 - Model on Jason
 - Extended Reasoning Cycle
 - First Evaluation
- 6 Future Works

Next in line...

- 1 Introduction
- 2 Reinforcement Learning Basic Concepts
- 3 Related Works
- 4 BDI-RL Model
 - Hard Plans and Soft Plans
 - Model
- 5 Jason Implementation
 - Model on Jason
 - Extended Reasoning Cycle
 - First Evaluation
- 6 Future Works

Vision

Machine learning

- Machine learning will be increasingly
 - ▶ an important feature for software products [Jones, 2014]
 - ▶ side by side with developers [Meijer, 2018] [Karpathy, 2017]
- Intersection of learning and software engineering still needs to be explored [Arpteg et al., 2018] [Khomh et al., 2018]
 - ▶ we pursue engineering features like modularity, reusability, testability

Purpose

- We want to systematically exploit ML in (agent-oriented) programming activities
- BDI-RL integration as a representative instance of AOP and ML integration

Basic idea

- The developer writes some plans and let the agent learns others
- Use them in a seamless way
- As a feature of the agent platform

Next in line...

- 1 Introduction
- 2 Reinforcement Learning Basic Concepts**
- 3 Related Works
- 4 BDI-RL Model
 - Hard Plans and Soft Plans
 - Model
- 5 Jason Implementation
 - Model on Jason
 - Extended Reasoning Cycle
 - First Evaluation
- 6 Future Works

Agent-environment interaction

- In RL, an agent learns how to fulfill a task by interacting with its environment
- The interaction between the agent and the environment can be reduced to three signals

Signals

State Every information about the environment useful to predict the future

Action What the agent do

Reward A real number that Indicates how well the agent is doing

From states to *observations*

- Sometimes, an agent has only a partial view of the world state
- The agent gets information (observations) about some aspects of the environment. From those, it reconstruct a sort of state
- Observations are a generalization of states – We use observations from now on

Terminologies

Policy

- A map from states to actions used by the agent to choose next action

Episode

- A complete run from one of the initial states to a final state

Episodic task

- A task that has an end
- Like an achievement goal in BDI

Continuing task

- A task that goes on without limit
- Like a maintenance goal in BDI

Next in line...

- 1 Introduction
- 2 Reinforcement Learning Basic Concepts
- 3 Related Works**
- 4 BDI-RL Model
 - Hard Plans and Soft Plans
 - Model
- 5 Jason Implementation
 - Model on Jason
 - Extended Reasoning Cycle
 - First Evaluation
- 6 Future Works

Agents and learning integration

- A main research topic in agents and MAS literature since their roots [Weiß, 1996]

Adaptivity

- A lot of works focus on the adaptivity problem
 - ▶ in particular on improving the plans selection
- Some references: [Guerra-Hernández et al., 2004] [Singh and Hindriks, 2013] [Singh et al., 2011] [Norling, 2004] [Airiau et al., 2009]

Other approaches

- Using Jason to implement RL methods [Badica et al., 2015] [Badica et al., 2017]
 - ▶ to face the RL problem with a more appropriate paradigm
- Once a policy is learned, a BDI agent is generated from it [Feliu, 2013]

Instances of BDI & learning

BDI-FALCON architecture

- [Tan et al., 2011] is a great example of BDI-learning integration
- TD-FALCON is a neural network based reinforcement learner
- Goals are represented with a target vector and an attainment function that defines the degree of achievement – used for reward
- Plans have confidence proportional to the probability of success
- The FALCON module updates the confidences according to the outcome of a plan execution
- If a plan is not available, the FALCON module decides the actions to perform and, when a successful sequence is found, a new plan is crafted

- Also [Karim et al., 2006] proposes a hybrid BDI-FALCON architecture

Differences between existing works and our proposal

- We focus on the representation of the state/action spaces and the management of the learning modules in a cognitive architecture
- The developer decides when using learning and for which tasks
- We seek to manage the boundary between adaptivity and controllability

Next in line...

- 1 Introduction
- 2 Reinforcement Learning Basic Concepts
- 3 Related Works
- 4 BDI-RL Model**
 - Hard Plans and Soft Plans
 - Model
- 5 Jason Implementation
 - Model on Jason
 - Extended Reasoning Cycle
 - First Evaluation
- 6 Future Works

Main idea

Hard Plans and Soft Plans

- The development process is extended with a learning phase – *education*
- The developer can:
 - ▶ write some plans — **Hard Plans**
 - ▶ let the agent itself learn other plans — **Soft plans**
- At runtime they are treated in a uniform way
- For soft plans, one must set up the learning phase

Learning module

- With soft plans we obtain a notion of *learning module*
- We can *replace* it, *reuse* it, *test* it

Concepts representation in BDI I

RL	Proposed BDI+ construct	Representation in BDI
Observations	Belief about Learning	Belief subset
Actions	Actions	Relevant Plans
Rewards	Motivational Rule	Belief Rule
Episode	Terminal Rule	Belief Rule
Policy	Learning Module	Plan
Behavior	Behavior	Intention

Model

- It represents the general concepts of RL
- It abstracts from the specific algorithm and problem

Concepts representation in BDI II

RL	Proposed BDI+ construct	Representation in BDI
Observations	Belief about Learning	Belief subset
Actions	Actions	Relevant Plans

Belief about Learning

- Belief useful to the learning process

Actions

- Subset of the plan library – plans can represent simple actions and compound actions
- Different levels of planning granularity
- It is called *option framework* in RL
- Context of plans used to define different action sets for different states

Concepts representation in BDI III

RL	Proposed BDI+ construct	Representation in BDI
Rewards	Motivational Rule	Belief Rule
Episode	Terminal Rule	Belief Rule
Policy	Learning Module	Plan
Behavior	Behavior	Intention

Motivational Rule

- Reflects the agent desires
- Generators of internal stimuli in the agent like a reward signal in neuroscience

Terminal Rule

- When one of these rules evaluates to true, the episode ends

RL Reasoner

- Elaborates the information and execute the learning algorithm
- Defines the behavior by suggesting the action to perform

Graphic representation of BDI-RL

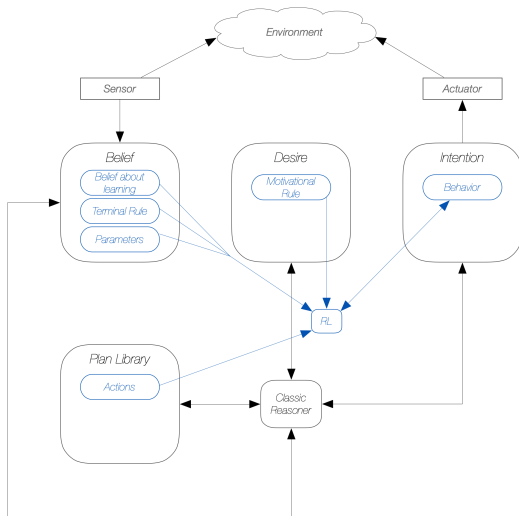


Figure: A graphic representation of the BDI model with the addition of our constructs

Extended BDI practical reasoning

```

1.  $B \leftarrow B'$ ; /*  $B'$  are initial beliefs */
2.  $I \leftarrow I'$ ; /*  $I'$  are initial intentions */
3. RL Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\epsilon > 0$ 
4. Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$ 
5.
6. while true do
7.   get next percept  $p$  via sensors;
8.    $B \leftarrow \text{brf}(B, p)$ ;
9.    $D \leftarrow \text{options}(B, I)$ ;
10.   $I \leftarrow \text{filter}(B, D, I)$ ;
11.   $\pi \leftarrow \text{plan}(B, I, Ac)$ ; /*  $Ac$  is the set of actions */
12.  while ( ( $\pi$  is learning plan and  $S$  is not terminal) or
13.         not ( $\pi$  is learning plan and empty( $\pi$ )) ) and
14.         not (succeeded( $I, B$ ) or impossible( $I, B$ )) do
15.    if  $\pi$  is learning plan then
16.       $O \leftarrow \text{related}(B, I)$ 
17.       $S' \leftarrow \text{state}(O)$ 
18.       $R \leftarrow \text{motivationalRule}(B)$ 
19.      Choose  $A'$  from  $\mathcal{A}(S')$  using policy derived from  $Q$  //(e.g.  $\epsilon$ -greedy)
20.      if  $S$  not null and  $A$  not null then
21.         $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A) - Q(S, A)]$ 
22.      end-if
23.       $S \leftarrow S'$ 
24.       $A \leftarrow A'$ 
25.      execute( $A$ )
26.    else
27.       $\alpha \leftarrow \text{first element of } \pi$ ;
28.      execute( $\alpha$ );
29.       $\pi \leftarrow \text{tail of } \pi$ ;
30.    end-if
31.    observe environment to get next percept  $p$ ;
32.     $B \leftarrow \text{brf}(B, p)$ ;
33.    if reconsider( $I, B$ ) then
34.       $D \leftarrow \text{options}(B, I)$ ;
35.       $I \leftarrow \text{filter}(B, D, I)$ ;
36.    end-if
37.    if not sound( $\pi, I, B$ ) then
38.       $\pi \leftarrow \text{plan}(B, I, Ac)$ 
39.    end-if
40.  end-while
41. end-while

```

Practical reasoning extended with RL

- The figure shows the pseudo code of a classic BDI agent reasoning cycle
- Extended with learning capabilities, specifically with the SARSA algorithm (adapted for the context)
- Additions are in red
- The function 'plan' in (11) is extended to include also soft plans

Next in line...

- 1 Introduction
- 2 Reinforcement Learning Basic Concepts
- 3 Related Works
- 4 BDI-RL Model
 - Hard Plans and Soft Plans
 - Model
- 5 Jason Implementation**
 - Model on Jason
 - Extended Reasoning Cycle
 - First Evaluation
- 6 Future Works

Framework Proof of Concept

Jason implementation

- We developed a PoC of the framework in Jason
 - Target: Pave the way to BDI-RL explorations
 - Limitation: Not ready to tackle real world problems

 - The state and action spaces dimensions are critical factor for an effective learning
- ⇒ The framework must make it possible for reduce the state/action spaces

Reference example: gridworld

- The agent can move in four directions: right, left, up, down
- The agent must reach a target block doing the minimum number of steps

Implementation of the framework available on GitHub

- <https://github.com/MichaelBosello/jacamo-rl>
- Includes the gridworld example

Constructs I

Soft plan goal

- A ground term identifies a soft plan goal
- ⇒ Enables learning of multiple tasks

Syntax `g`

Example `reach_end`

Belief about learning

- Declares the relevant beliefs for a task
- ⇒ Reduce the state space

Syntax `rl_observe(g, [o1, o2, ... on])`.

Example `rl_observe(reach_end, [position])`.

Constructs II

Motivational rule

- R is the reward and must be a number, possibly a variable set in the rule's body
- The final reward is the sum of all the rewards provided by the currently true motivational rules

```
Syntax rl_reward(g, R) :- <reward conditions>.
```

```
Example rl_reward(reach_end, 10) :- finish_line.  
        rl_reward(reach_end, -1) :- not finish_line.
```

Terminal rule

- When one of these rules evaluates to true, the task g ends

```
Syntax rl_terminal(g) :- <episode ends conditions>.
```

```
Example rl_terminal(reach_end) :- finish_line.
```

Constructs III

Action

- The action label declares the plans useful for a task
- ⇒ Reduce the action space
- One can use variables in action declaration. Variable type and range must be indicated in the label
- ⇒ Allows to specify *range* of actions (even *continuous* one)

```
Syntax @action[rl_goal(g1, ..., gn), rl_param(p1, ..., pm)]
```

Example

```
@action[rl_goal(reach_end),  
        rl_param( direction(set(right, left, up, down)) )]  
+!move(Direction) <- move(Direction).
```

Constructs IV

Learning parameters

- Also the learning parameters are entered as beliefs
- ⇒ Allows the complete control of the learning process by the agent/developer
- ⇒ The agent can manage the exploration/exploitation phases

Syntax `rl_parameter(name, value).`

Example
`rl_parameter(alpha, 0.26).`
`rl_parameter(gamma, 0.9).`
`rl_parameter(policy, egreedy).`

Constructs V

Soft plan learning and execution

- The internal action *rl.execute* run the soft plan to achieve *g*
 - In the meantime, the reasoner updates (learns) the policy
- ⇒ The action performs one learning run
- One episode for an episodic task
Goes on continuously for a continuous task
- ▷ (one can set a limit e.g. performance, time, number of actions)
- One can use a belief parameter to choose between learn-and-act or act only
 - The internal action is implemented in Java, this allows to reuse existing RL libraries

Syntax `rl.execute(g)`

Constructs VI

Soft plan evaluation

- The internal action *rl.expectedreturn* gets the estimate of future rewards R for the goal g on the basis of the current state and learned policy, i.e. the *expected return*
- Could be used to understand the performance of the leaned soft plan
- We obtain a notion of *context* for soft plans
- E.g. if the expectation in the current state is poor, we can fall back on another plan

Syntax `rl.expectedreturn(g, R)`

How a learning agent looks like

```

rl_parameter(policy, egreedy).
rl_parameter(alpha, 0.26).
rl_parameter(gamma, 0.9).
rl_parameter(epsilon, 0.22).
rl_parameter(epsilon_decay, 0.99992).

rl_observe(reach_finish, pos).

rl_reward(reach_finish, 10) :- finish_line.
rl_reward(reach_finish, -1) :- not finish_line.

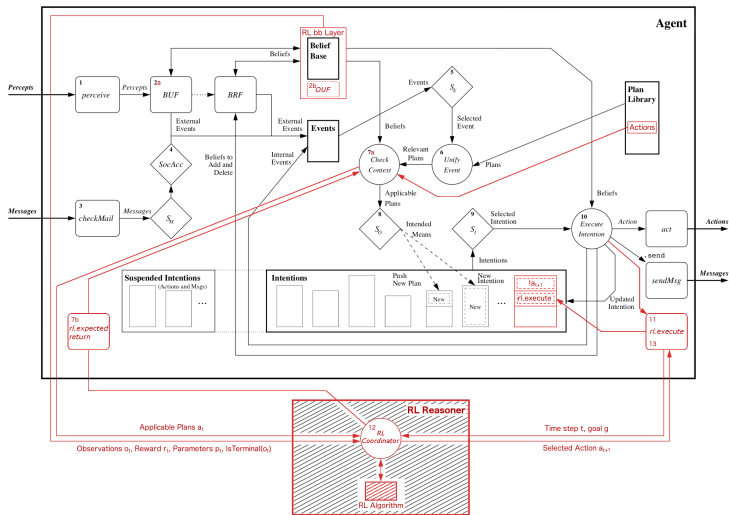
rl_terminal(reach_finish) :- finish_line.

@action[rl_goal(reach_finish),
        rl_param(direction(set(right, left, up, down)))]
+!move(Direction) <- move(Direction).

/* in this case, we run an infinite learning process - actually it
   could be stopped when the performance (expected return)
   is considered good enough */
!start.
+!start : true <- rl.execute(reach_finish); !start.

```

Extended Jason reasoning cycle



Extended Jason reasoning cycle details

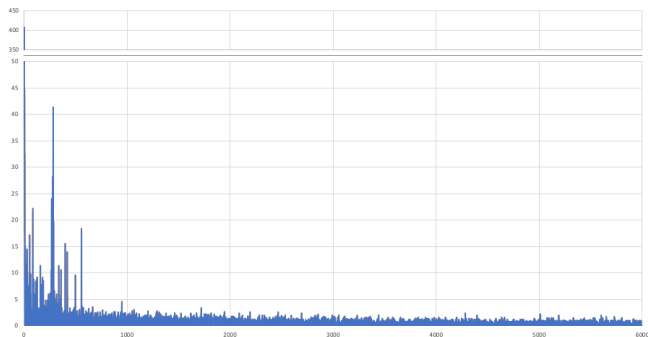
- We extended the Jason architecture to include learning aspects
Our additions are in red
 - (2b) Observations are updated
 - (7a-7b) The context of a plan can be bound to an expected return threshold
 - (11) rl.execute asks for the next action
 - (12) The RL reasoner gets all the needed information from the BB plus the relevant actions filtered by (7)
 - (13) rl.execute puts the next action on top of its intention
 - ▶ if the episode isn't over, another call of rl.execute is placed under the action in the same intention
- The next RL step is always performed after the execution of the previously selected action
- The RL reasoner is a black box for the agent
- The RL algorithm is a black box for the RL Coordinator

Tests

- We have implemented the SARSA algorithm into the framework to test it
- SARSA was adequate to address the gridworld problem
- The algorithm and the problem tested aren't enough to evaluate performance, scalability, and generality of the approach

Testing results (gridworld)

- At every episode the agent appears in a random place
- Policy: ϵ -greedy with decay
- Parameters have been: $\alpha = 0.26$, $\gamma = 0.9$, $\epsilon = 0.22$, ϵ -decay = 0.99992
- The agents learns the policy in about 1000 episodes
- The agents behaves near-optimal in about 5000 episodes (when random behavior is less than 5%)
- The image shows the average results of five trials with 6000 episodes



Key points

- The developer can bind the uncertainty of learning
- The developer can inject domain knowledge through proper high-level abstractions
 - ▶ coherent way to manage states with beliefs
 - ▶ action set shaping through plans
- General approach independent from the RL algorithm – enables the use of the more fitting algorithm for the single task
- Learning tasks obtain modularity and reusability through soft plans
- Hierarchical approach thanks to plans (composability)

Next in line...

- 1 Introduction
- 2 Reinforcement Learning Basic Concepts
- 3 Related Works
- 4 BDI-RL Model
 - Hard Plans and Soft Plans
 - Model
- 5 Jason Implementation
 - Model on Jason
 - Extended Reasoning Cycle
 - First Evaluation
- 6 Future Works**

Future works I

Engineering side

- Tests with complex cases — to evaluate performance, scalability, expressiveness
- Analyze the actual benefits of soft plans in terms of engineering features like modularity, extensibility, reusability, and composability
- Explore further the education process lifecycle and stages relations
- Verify what is the impact on AOSE (Agent-Oriented Software Engineering) methodologies
- Develop proper tools to be embedded in existing IDEs, including simulators and learning facilities

CS side

- Semantic formalization
- Study of computational complexity
- Check learning convergence guarantees

Future works II

Learning side

- Implement more effective RL algorithms
- Interesting RL extensions
 - ▶ Hierarchical RL – it allows to aggregate actions into reusable subroutines or skills
 - ▶ Reward Shaping – “education” through demonstrations
- Consider first-class abstractions such as artifacts in the process
- Explore the impact of environments built to support the agents
- Consider also the education process within a multi-agent system or an agent organization and its implications

● **Suggestions are welcome!**

Feedback and cooperation

- Anyone who is interested in this research line that wants to
 - ▶ discuss
 - ▶ cooperate with us
 - ▶ give feedback

can contact

Me `michael.bosello@studio.unibo.it`
Prof. Ricci `a.ricci@unibo.it`

- Any questions?

References

- Reinforcement Learning [Sutton and Barto, 2018]
- Agent-Oriented Programming [Shoham, 1993]
- BDI [Rao and Georgeff, 1995]
- Reasoning Cycle [Wooldridge, 2009]
- Jason [Bordini et al., 2007]
- HRL [Botvinick et al., 2009]
- Shaping in RL [Brys et al., 2015]
- Agents&Artifacts [Ricci et al., 2011]

References I



Airiau, S., Padgham, L., Sardina, S., and Sen, S. (2009).
Enhancing the adaptation of BDI agents using learning techniques.
Int. J. Agent Technol. Syst., 1(2):1–18.



Arpteg, A., Brinne, B., Crnkovic-Friis, L., and Bosch, J. (2018).
Software engineering challenges of deep learning.
In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 50–59.



Badica, A., Badica, C., Ivanovic, M., and Mitrovic, D. (2015).
An approach of temporal difference learning using agent-oriented programming.
In *2015 20th International Conference on Control Systems and Computer Science*, pages 735–742.



Badica, C., Becheru, A., and Felton, S. (2017).
Integration of jason reinforcement learning agents into an interactive application.
In *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 361–368.

References II



Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007).
Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology).
John Wiley & Sons, Inc., USA.



Botvinick, M. M., Niv, Y., and Barto, A. C. (2009).
Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective.
Cognition, 113(3):262 – 280.
Reinforcement learning and higher cognition.







Brys, T., Harutyunyan, A., Suay, H. B., Chernova, S., Taylor, M. E., and Nowé, A. (2015).
Reinforcement learning from demonstration through shaping.
In Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15, pages 3352–3358. AAAI Press.



Feliu, J. L. (2013).
Use of reinforcement learning (rl) for plan generation in belief-desire-intention (bdi) agent systems.

References III

-  Guerra-Hernández, A., El Fallah-Seghrouchni, A., and Soldano, H. (2004). Learning in BDI multi-agent systems. In *Proceedings of the 4th International Conference on Computational Logic in Multi-Agent Systems, CLIMA IV'04*, pages 218–233, Berlin, Heidelberg. Springer-Verlag.
-  Jones, N. (2014). The learning machines. *Nature*, 505(7482):146.
-  Karim, S., Sonenberg, L., and Tan, A.-H. (2006). A hybrid architecture combining reactive plan execution and reactive learning. In Yang, Q. and Webb, G., editors, *PRICAI 2006: Trends in Artificial Intelligence*, pages 200–211, Berlin, Heidelberg. Springer Berlin Heidelberg.
-  Karpathy, A. (2017). Software 2.0.
-  Khomh, F., Adams, B., Cheng, J., Fokaefs, M., and Antoniol, G. (2018). Software engineering for machine-learning applications: The road ahead. *IEEE Software*, 35(5):81–84.

References IV



Meijer, E. (2018).

Behind every great deep learning framework is an even greater programming languages concept (keynote).

In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018, pages 1–1, New York, NY, USA. ACM.



Norling, E. (2004).

Folk psychology for human modelling: Extending the BDI paradigm.



Rao, A. S. and Georgeff, M. P. (1995).

Bdi agents: From theory to practice.
pages 312–319.



Ricci, A., Piunti, M., and Viroli, M. (2011).

Environment programming in multi-agent systems: an artifact-based perspective.
Autonomous Agents and Multi-Agent Systems, 23(2):158–192.



Shoham, Y. (1993).

Agent-oriented programming.
Artif. Intell., 60(1):51–92.

References V



Singh, D. and Hindriks, K. V. (2013).

Learning to improve agent behaviours in goal.

In Dastani, M., Hübner, J. F., and Logan, B., editors, *Programming Multi-Agent Systems*, pages 158–173, Berlin, Heidelberg. Springer Berlin Heidelberg.



Singh, D., Sardina, S., Padgham, L., and James, G. (2011).

Integrating learning into a BDI agent for environments with changing dynamics.

In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pages 2525–2530. AAAI Press.



Sutton, R. S. and Barto, A. G. (2018).

Reinforcement learning : an introduction.

The MIT Press.



Tan, A.-H., Ong, Y.-S., and Tapanuj, A. (2011).

A hybrid agent architecture integrating desire, intention and reinforcement learning.

Expert Syst. Appl., 38(7):8477–8487.

References VI



Weiß, G. (1996).

Adaptation and learning in multi-agent systems: Some remarks and a bibliography.

In Weiß, G. and Sen, S., editors, *Adaption and Learning in Multi-Agent Systems*, pages 1–21, Berlin, Heidelberg. Springer Berlin Heidelberg.



Wooldridge, M. (2009).

Introduction to Multi-Agent Systems.
Wiley.

From Programming Agents to *Educating* Agents

A Jason-based Framework for Integrating Learning in the Development of Cognitive Agents

Michael Bosello Alessandro Ricci

Alma Mater Studiorum – Università di Bologna
Department of Computer Science and Engineering, Cesena Campus

EMAS 2019