Using Institutional Purposes to Enhance Openness of Multi-Agent Systems

Rafhael R. Cunha[1,2](✉), Jomi F. Hübner[2], and Maiquel de Brito[3]

[1] Federal Institute of Education, Science and Technology of Rio Grande do Sul (IFRS), Campus Vacaria, Vacaria, Brazil

[2] Department of Automation and Systems Engineering, Federal University of Santa Catarina, Florianópolis, Brazil
`rafhael.cunha@posgrad.ufsc.br`, `jomi.hubner@ufsc.br`

[3] Control, Automation, and Computation Engineering Department, Federal University of Santa Catarina, Blumenau, Brazil
`maiquel.b@ufsc.br`

Abstract. In this paper, we consider a programmer who needs to develop an agent to publish information on different existing social networks. Several works in the literature allow the interaction of agents with a social network, but as far as we know, they are not focused on the interaction with several networks. Considering this problem, the paper highlights two options the programmer can take to enable the agent to interact with existing social networks and achieve its goal. The first option requires that the programmer knows the features present in the APIs used to interact with social networks, and the consequences that the functions can bring when executed in the system. The second option uses the institutional notion of *purpose* aiming to reduce the amount of previous knowledge required by the programmer. The paper aims to discuss the advantages and disadvantages of these options considering the development of open multi-agent system. A JaCaMo implementation of both options is used the help the evaluation.

Keywords: Agent programmer · Artificial institutions · Purposes

1 Introduction

We have recently witnessed a massive diffusion of social networking based on web applications that have quickly become an unprecedented cultural phenomenon [7]. Such web applications allow members to publish personal information in a semi-structured form and to define links to other members with whom they have relationships of various kinds [4]. These applications are available for both human and software agents, allowing them to enter and leave the

This study was supported by the Federal Institute of Education, Science and Technology of Rio Grande do Sul (IFRS).

F. Dignum et al. (Eds.): PAAMS 2022, LNAI 13616, pp. 88–99, 2022.
https://doi.org/10.1007/978-3-031-18192-4_8

system to achieve their goals. Therefore, these applications can be considered as an open multi-agent systems (MAS).

Software agents may pursue the same goals in the different available social network systems. In this context, imagine an agent developer who needs to program the agent *Bob* whose goal is to publish information on different social networks. Existing proposals for interaction between agents and social networks [1,2,8,16] adopt ad-hoc solutions that require that the developer knows each social network and implements the necessary actions for the agent to reach its goal in these systems. The main drawback of this approach is that the developer needs to know the peculiarities of each social network application and program the agents with the necessary actions to interact with each one or, conversely each particular system needs to be programmed in a way that is compatible with the agent specifications.

Besides requiring a specific program for every different system, there are other disadvantages of current approaches. To understand them, consider that *Bob* is programmed to achieve its goal (i.e., information published) by sending a text through a method called *tweet* on the social network *Twitter*. Also consider that *Bob* has an anti-goal (i.e., a state of the system it avoids [3]) of *fake news spread*. First, suppose that new actions become available or existing ones are modified. The developer needs to know about these new methods or modifications and change Bob's program to keep it working on this system. Second, suppose that other actions produce similar effects. For example, consider that for some reason the social network *Twitter* is currently unavailable. To achieve its goal, *Bob* could publish this information on another social network (e.g., *Instagram*). However, the developer did not specify in Bob's program that a similar action performed on another social network produces similar practical effects. To handle this situation, the developer has to know the practical effects of actions and program *Bob* to act on different systems, or the different systems have to be compatible with Bob's specification. Third, suppose that the actions programmed into the agent's program bring unwanted effects to the system. For example, consider that the action *tweet* produces two effects on the system: (i) it publishes information, that is the goal of *Bob*, but (ii) it spreads fake news, that is an anti-goal of *Bob*. The developer needs to know all the effects of the action and evaluate whether or not to program the agent to perform the action to achieve the agent's goal.

Considering these limitations, we propose a solution that moves part of the programmer task to the agent, using its reasoning capabilities. For that, we instrument the system based on social concepts already proposed in the literature, which are institutions [20,21] and purposes [10,11]. Both concepts, when combined, allow the system to make explicit to the agents what are the effects in the environment of performing certain actions in the system. The agents can then relate these effects to their (anti) goals. The connection between available actions and their effects is moved from the agents program (as defined by the programmer) to the institution where they act. Possible changes in these connections, in the actions, or in their effects require changes in the institutional specification while the agents remain unchanged. Besides, the agents can move along different systems looking for satisfying their goals based on the desired effects of the available actions even if these actions are unknown a priori. In this paper, we illustrate this approach through a practical example, which supports a discussion about the advantages and disadvantages of this programming model.

This paper is organized as follows: Sect. 2 introduces the main background concepts necessary to understanding the model used in this work and its required interfaces. Section 3 illustrates how the use of artificial institutions and purposes facilitates the development of agents capable of acting in different systems. Section 4 identifies and discusses some limitations and advantages that the use of the purpose model offers for MAS programming from the agent programmer's perspective. Finally, Sect. 5 presents some conclusions about this work and suggests future works.

2 Artificial Institutions and Purposes

The essential elements of the model used in this work are *agents*, *(anti) goals*, *institutions*, and *purposes*, depicted in the Fig. 1. *Agents* are autonomous entities that can interact within a dynamic environment composed of non-autonomous elements to achieve their goals [23]. The literature presents several definitions of *goal* that are different but complementary to each other (see more in [5,13–15,18,22]). In this work, *goals* are states that agents aim to achieve. According to Aydemir, et.al [3], *anti-goal* is an undesired situation of the system. In this work, *anti-goal* represents states that agents aim to avoid for ethical reasons, particular values, prohibitions, etc. Moreover, agents can perform actions that trigger events in the MAS. *States* are formed by one or more properties that describe the characteristics of the system at some point of its execution [9].

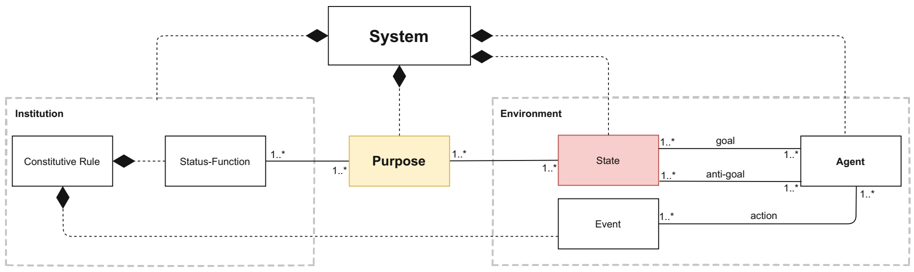

Fig. 1. Overview of the model.

Institutions provide social interpretations for the environment elements that compose the system. They are simplified here to be based on two concepts: *Status-functions* and *Constitutive rules* [20,21]. *Status-functions* are *status* that assign *functions* to the elements. These functions cannot be explained through their physical virtues. For example, the status *buyer* assigns to an agent some functions such as performing payments, taking loans, etc. *Constitutive rules* specify the assignment of status-functions to elements with the following formula: *X* count-as *Y* in *C*. For example, a piece of paper count-as money in a bank, where X represents the environmental element, Y the status-function, and C the context where the constitution is valid. The assignment of status-functions to environmental elements is called *constitution* and creates institutional facts, which gives rise to institutions. Artificial Institutions (or simply *institutions*) are the component of the MAS that is responsible for defining the conditions for an agent to become a *buyer*, or an action to become a *payment* [20,21].

The functions associated with status-functions can satisfy the practical interests of agents [21, p.20]. From the institution's perspective, we call these interests as *Purposes*. From the agents' perspective, these interests are their goals or anti-goals. Then, we claim that (i) *goals and anti-goals match with the purposes of status-functions* and (ii) *goals, anti-goals and purposes point to environmental states related to the status-functions*. For example, when an agent performs an action that constitutes *tweet*, this makes possible the execution of other intermediate actions that bring the system to states such as *information published* (i.e., the agent goal) or *fake news spread* (i.e., the agent anti-goal). The intermediate actions (e.g., server receives the message, filter the message if necessary, etc.) between the constitution of the status-functions and the environmental states being reached are ignored in our proposal, since we consider that the agent is not interested in these intermediate steps.

Shortly, this model provides two relationships: (i) between purposes and status-functions and (ii) between purposes and agent goals and anti-goals. Thus, if (i) there is a constitutive rule specifying how a status-function is constituted, (ii) a purpose associated with that status-functions, and (iii) an agent has a goal or anti-goal that matches with the states pointed to by the purpose, then (iv) it is explicit how the agent should act to achieve its goal or avoid its anti-goal. In the previous example, the programmer can use these two relations in the agent code to program two queries: (i) a query to find which states the purposes point to and that match the agent goals or anti-goals and (ii) a query to find out which status-function is associated with the found purposes. Thus, for example, the agent can find that the purpose of *transmitting information* points to the *information published* state which matches the goal and that the purpose of *transmitting information* is associated with the *tweet* status-function. Therefore, if the agent constitutes *tweet*, it achieves its goal in this system.

3 Implementing a Multi-agent System with and Without the Purpose Model

This section illustrates how the use of artificial institutions and purposes facilitates the development of agents towards acting in different systems. We describe the development of a multi-agent system without using and then using the model described in Sect. 2. For this, we recall the example where *Bob* wants to achieve its goal of *information published* on different social networks. It is assumed that different programmers develop each social network and that the development of the agent has no influence on this. While the example focuses on the achievement of goals, it is important to make it clear that the model could be used to deal with anti-goals. For example, *Bob* could have a belief in a schedule that describes its anti-goals. Knowing that a certain action could bring about an anti-goal state for it, it could reason about and avoid or not the execution of the action according to its interests.

The example is implemented through the components depicted in Fig. 2. Agents are programmed in Jason [6] and the environment in CArtAgO [17]. A CArtAgO artifact encloses specific APIs and provides actions for the agents to act upon each social network. For the artificial institution, we use an implementation of the Situated Artificial Institution (SAI) model [12]. It provides means to specify status-functions and constitutive rules and to manage the constitution process. The purpose model is implemented through an ontology encapsulated in a CArtAgO artifact which is accessible to the agents. The query and persistence of data in the ontology are enabled by the MasOntology[1], a set of tools developed in CArtAgO to interact with ontologies[2]

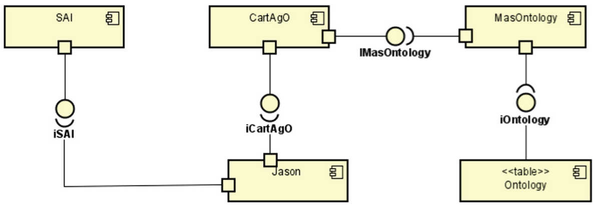

Fig. 2. Component diagram with the systems used to compose the example.

This section is organized as follows: Subsect. 3.1 describes a first implementation that does not use of the proposed model. For this implementation, the programmer needs to know all the actions that the agent must perform on each system it interacts with. Therefore, the program of the agents and the systems that it interacts with are tightly coupled. Subsection 3.2 describes a second implementation that uses the proposed model. In this implementation, the programmer does not need to know the actions that the agent should perform to interact with other systems because they can be discovered at runtime. Therefore, the program of the agents and the systems it interacts with are loosely coupled.

[1] https://github.com/smart-pucrs/MasOntology.

[2] An initial implementation of this platform can be found in https://github.com/rafhaelrc/psf_model..

3.1 Implementation Without Institutions and Purposes

In this section, we consider a scenario where there is no institution or purposes. For this reason, the connection between the available actions and the goals must be coded in the agent. The programmer should thus previously know these implementation details for all the social network systems. This program corresponds to a first alternative for moving the system from the state *S1* (where *Bob* desires to reach its goal) to the state *S2* (where *Bob* has reached its goal).

```
1 !infoPublished.
2
3 +!infoPublished : knet(Twitter)   <- sendMessageByTwitter.
4 +!infoPublished : knet(Telegram)  <- talkWithBot.
5 +!infoPublished : knet(Instagram) <- uploadAPIcture.
6 +!infoPublished : knet(Facebook)  <- uploadAMessage.
```

Listing 1.1. Bob's program.

Listing 1.1 contains part of the Bob's program. In this implementation we assume that when the agent enters different systems, it acquires the belief $knet(S)$ where $S \in \{twitter, telegram, instagram, facebook\}$ is the name of the system where the agent is currently acting. Bob's goal is specified in line 1. The plans that *Bob* may execute to achieve its goal are outlined between lines 3 and 6. The plan that *Bob* executes depends on which social network it is interacting. Consider that *Bob* starts believing in *knet(Facebook)* when it enters the *Facebook* system. The next step, *Bob* chooses which plan should be executed to achieve its goal. Looking at the available plans and their respective contexts, Bob selects the plan in line 6. Then *Bob* performs the action called `uploadAMessage`. At this point it is important to be clear that the available actions (e.g., `uploadAMessage`, `talkWithBot`, etc.) are not provided by the social network systems. Rather, they are implemented in the CArtAgO artifact that provides access to the APIs of the social networks. This action causes some consequences in the system, including switching the system to a new state where *Bob* achieves its goal of *information published*. In this example, we highlight that, to write the program of *Bob*, the programmer of *Bob* needs to know the actions and their consequences for each system in which *Bob* interacts.

3.2 Implementation with Institutions and Purposes

In this section, we consider a scenario where the social network systems include implementations of institutions and purposes. For this reason, the connection between available actions and their effects is moved from the agents to the systems where they act. The programmer does not need to know the implementation details of the social network systems. Figure 3 depicts an overview of the implementation. This program is a second alternative for moving the system from state *S1* to state *S2*.

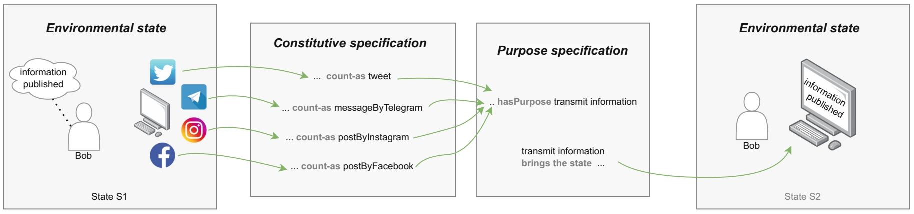
Environmental state
information published
Bob
State S1
Constitutive specification
... count-as tweet
... count-as messageByTelegram
... count-as postByInstagram
... count-as postByFacebook
Purpose specification
.. hasPurpose transmit information
transmit information brings the state ...
Environmental state
information published
Bob
State S2

Fig. 3. Implementation overview with institutions and purpose.

In this program, we assume that (i) each system has a specification of the constitutive rules that make up the institution and (ii) each system has a purpose specification that is related with (a) the status-functions and (b) the states of the world that the purpose points to. These specifications are shown in listings 1.2, 1.3, 1.4 and 1.5. We also assume that the agent has runtime access to the institutional and purpose specifications of each system.

```
status_functions: tweet
tweet has purpose of: transmit information
transmit information brings the state information published

Constitutive_rules:
1: sendMessageByTwitter count-as tweet.
```

Listing 1.2. Twitter Institutional and Purpose Specification

```
status_functions: messageByTelegram
messageByTelegram has purpose of: transmit information
transmit information brings the state information published

Constitutive_rules:
1: talkWithBot count-as messageByTelegram.
```

Listing 1.3. Telegram Institutional and Purpose Specification

```
status_functions: postByInstagram
postByInstagram has purpose of: transmit information
transmit information brings the state information published

Constitutive_rules:
1: uploadAPicture count-as postByInstagram.
```

Listing 1.4. Instagram Institutional and Purpose Specification

```
status_functions: postByFacebook
postByFacebook has purpose of: transmit information
transmit information brings the state information published

Constitutive_rules:
1: uploadAMessage count-as postByFacebook.
```

Listing 1.5. Facebook Institutional and Purpose Specification

Listing 1.6 illustrates Bob's program. Bob's goal is specified on line 1 and does not change regardless of which system *Bob* interacts with. At this point it is important to be clear that the actions from lines 4 to 6 are available by the institutional infrastructures of the different systems in which *Bob* can interact. Each institution has a CArtAgO artifact that allows the agent to access information related to institutional specification and purposes. Each system has also a CArtAgO artifact developed by us that allows access to an API that interacts with that social network. Given Bob's program access to the institutional and constitutive specifications of each system, the programmer of *Bob* can implement it with a generic plan that helps *Bob* to achieve its goal on many systems. The plan has the following steps: (i) discover the purpose that points to the desired state (line 4), (ii) discover the status-function associated with the found purpose (line 5), (iii) discover which concrete action can constitute the status-function (line 6) and finally perform the concrete action (line 7). Figure 3 depicts, through the green arrows, the steps described in descending order (i.e., after the action is performed). As an example, consider that Bob's program should interact with the social network *Twitter* and Listing 1.2 which describes the institutional specification of purposes for this system. To do this, Bob's program identifies the plan to be executed (lines 3–7). When executing the plan, some information is discovered at runtime: First, *Bob* queries the purpose that points to the desired state. The name of the purpose is *transmit information*. Second, *Bob* queries which status-function is associated with the *transmit information* purpose. The status-functions name is *tweet*. Third, *Bob* queries what concrete action can constitute the *tweet* status-function. The action name is `sendMessageByTwitter`. Finally, *Bob* performs this action.

```
1 !infoPublished.
2 
3 +!infoPublished
4     <- getPurposesOfState(infoPublished,NamePurposes);
5        getStatusFunctionsFromPurposes(NamePurposes,
           NameStatusFunction);
6        ?constitutive_rule(Action,NameStatusFunction,_,_);
7        Action.
```

Listing 1.6. Bob's program using institutions and purposes.

The difference between this implementation and the one described in Sect. 3.1 is that it uses mechanisms (i.e., institutions and purposes) that make explicit (i) the statuses that can be assigned to concrete actions when executed in the system and (ii) the intrinsic functions related to these statuses, called purposes, which describe the states that can be reached in the system related to the status-functions. The execution of the selected action (line 7) constitutes a status-function whose purposes point to environmental states desired by the agent. The advantages of this approach from the programmer's point of view are discussed in the next section.

4 Discussion About both Implementations

In this section we discuss some limitations in Bob's program from Sect. 3.1 (called from now on *Program 1*) that are overcome in Sect. 3.2 (called from now on *Program 2*). These limitations are due to the *tight coupling between the agent specification and the systems in which it operates* and can be observed from (i) the point of view of the system implementation and (ii) the abstractions necessary to develop of the system.

From the perspective of system implementation, *Program 1* requires that the programmer code all the functions that are necessary for the agent to interact with different social networks and know their consequences when executed. Or, conversely, each social network should be compatible and prepared to work with the agent. However, (i) whenever new actions are added, or existing actions are extinguished in the social network API, Bob's program must be updated; (ii) every different action that produces the same effect must be coded in the agent to be exploited in running time; (iii) if the practical effects of the action bring other consequences not foreseen or unwanted by Bob's program, the program may not function properly.

In *program 2* the limitations from the perspective of system implementation are overcome because *the connection between available actions and their effects is moved from the agents to the systems where they act.* Possible changes in these connections, in the actions, or in their effects require changes in the systems while the agents remain unchanged. This modification brings advantages to the system: (i) if new actions are added, or existing ones are modified, the agent's program remains stable as long as the changes are incorporated by the institutions (through the addition of new constitutive rules that reflect these changes); (ii) The agent can exploit any action that produces the desired effect without any additional coding since it acts based on the effects of the actions (connected to the purposed) instead of acting based on the actions themselves; (iii) if some actions bring undesired practical effects by the agent program, the program can use this information in its decision process as long as these practical effects are pointed out through the purposes specification.

From the perspective of abstractions necessary to develop the system, *Program 1* requires the programmer to know all the abstractions needed to code *Bob* to interact with different systems to achieve its goal. The connections between (i) the actions that *Bob* has to perform, (ii) their consequences on the system, and (iii) the satisfaction of bob's goal are implicit (they exist only in the mind of the programmer). If the programmer does not know any of this information, s/he may have trouble programming the agent.

In *program 2* the limitations on the perspective of abstractions needed to implement the system are overcome because the connections between the actions that *Bob* has to perform and its effects are moved to the institutional and purpose specifications respectively, allowing agents to be able to relate them to the satisfaction of its goals. So, the programmer can focus on other aspects of the agent. The programming of the other dimensions of the system (institution, purpose, etc.) is left to other programmers.

5 Conclusions and Future Work

This paper aims to discuss the advantages in programming agents for an open MAS composed of artificial institutions and purposes. From the programmer's perspective, there are some advantages to using the solution presented in this paper. The first is related to the adaptability of the developed agent to act in different systems. If the program is designed to discover which purposes match the agent's goal and then which status-functions that when constituted creates the possibility of states pointed out by the purposes, the agent's program keeps working on any system that provides this information as long as the purposes related to status-functions match to states of the environment desired by the agents. Therefore, the programmer can code the agent to interact with different systems without modifying the program whenever the agent moves to another system. Second, by externalizing the effect of actions to other appropriate concepts (i.e., institutional purposes), part of the programmer task (i.e., to code plans for each system) is delegated to the agent reasoning.

From the agents' perspective, there are some advantages of using the solution presented in this paper. One of them is the improvement in the agent's decision-making since it has more information available to help it to decide whether to achieve its goals or avoid its anti-goals. Agents can access and reason about the environmental states pointed to by the purposes and adapt themselves to different scenarios. They can (a) notice that some purposes point to environmental states that match to their interests and therefore useful to reach their goals and (b) avoid these environmental states because they are match to their anti-goals. The agent understanding about what makes its goal satisfied or what it can avoid to not satisfy anti-goals are important advances in its autonomy [19]. In this case, the agent can reason about the actions and regulative rules that govern the system. In both cases, the agent has more information while deciding whether a particular action will help it or hinder. Some advantages can be observed from an institutional point of view, which are detailed in [10].

While social networks are used to illustrate the approach, the proposal can be applied in several other scenarios. The problem presented can be generalized to the fact that (i) the agent has a goal that can be satisfied in different systems and that (ii) these systems make explicit the connections that exist between their actions and effects. From this generalization, consider the scenario where an agent needs to make a purchase on different websites. The way to solve this problem is similar to that presented in this paper, considering that (a) the programmer codes the agent to search for the actions it has to perform on each website it accesses and (b) the websites describe the actions that are available and the effects of those actions when performed. Therefore, problems that present similar characteristics can use the solution presented in this paper to enable the solution.

As future work, we plan to explore some aspects related to the proposal, such as (i) the relation between purposes and the values of the agents, (ii) discovering the learning curve for the programmer to use the purpose model, (iii) investigating additional advantages of the use of the purpose model and (iv) assess whether the design of the complete system becomes more dynamic or prolonged if it externalizes the concepts that make the necessary connections to make the program more flexible.

References

1. Abreu, J.V.F.: AgentBotSpotter: a multi-agent system for online Twitter bot detection. Ph.D. thesis, Universidade de Brasília (2021)
2. Amaral, C.J., Hübner, J.F.: Jacamo-web is on the fly: an interactive multi-agent system IDE. In: Dennis, L.A., Bordini, R.H., Lespérance, Y. (eds.) EMAS 2019. LNCS (LNAI), vol. 12058, pp. 246–255. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51417-4_13
3. Aydemir, F.B., Giorgini, P., Mylopoulos, J.: Multi-objective risk analysis with goal models. In: 2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS), pp. 1–10. IEEE (2016)
4. Bergenti, F., Franchi, E., Poggi, A.: Enhancing social networks with agent and semantic web technologies. In: Collaboration and the Semantic Web: Social Networks, Knowledge Networks, and Knowledge Resources, pp. 83–100. IGI Global (2012)
5. Boissier, O., Bordini, R.H., Hubner, J., Ricci, A.: Multi-agent oriented programming: programming multi-agent systems using JaCaMo. MIT Press (2020)
6. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason, vol. 8. John Wiley & Sons (2007)
7. Boyd, D.M., Ellison, N.B.: Social network sites: definition, history, and scholarship. J. Comput.-Mediat. Commun. **13**(1), 210–230 (2007)
8. Calvaresi, D., Calbimonte, J.P., Dubosson, F., Najjar, A., Schumacher, M.: Social network chatbots for smoking cessation: agent and multi-agent frameworks. In: 2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI), pp. 286–292. IEEE (2019)
9. Cassandras, C.G., Lafortune, S.: Introduction to discrete event systems. Springer, Cham (2008)
10. Cunha, R.R., Hübner, J.F., de Brito, M.: Coupling purposes with status-functions in artificial institutions. arXiv preprint arXiv:2105.00090 (2021)
11. Cunha, R.R., Hübner, J.F., de Brito, M.: A conceptual model for situating purposes in artificial institutions. Revista de Informática Teórica e Aplicada **29**(1), 68–80 (2022)
12. de Brito, M., Hübner, J.F., Boissier, O.: Situated artificial institutions: stability, consistency, and flexibility in the regulation of agent societies. Auton. Agent. Multi-Agent Syst. **32**(2), 219–251 (2017). https://doi.org/10.1007/s10458-017-9379-3

13. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.C.: Agent programming with declarative goals. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, pp. 228–243. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44631-1_16
14. Hübner, J.F., Bordini, R.H., Wooldridge, M.: Declarative goal patterns for agentspeak. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006) (2006)
15. Nigam, V., Leite, J.: A dynamic logic programming based system for agents with declarative goals. In: Baldoni, M., Endriss, U. (eds.) DALT 2006. LNCS (LNAI), vol. 4327, pp. 174–190. Springer, Heidelberg (2006). https://doi.org/10.1007/11961536_12
16. Pérez-Marcos, J., Jiménez-Bravo, D.M., De Paz, J.F., Villarrubia González, G., López, V.F., Gil, A.B.: Multi-agent system application for music features extraction, meta-classification and context analysis. Knowl. Inf. Syst. **62**(1), 401–422 (2019). https://doi.org/10.1007/s10115-018-1319-2
17. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: an artifact-based perspective. Auton. Agent. Multi-Agent Syst. **23**(2), 158–192 (2011)
18. van Riemsdijk, B., van der Hoek, W., Meyer, J.J.C.: Agent programming in dribble: from beliefs to goals using plans. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 393–400 (2003)
19. Rodriguez-Aguilar, J.A., Sierra, C., Arcos, J.L., Lopez-Sanchez, M., Rodriguez, I.: Towards next generation coordination infrastructures. Knowl. Eng. Rev. **30**(4), 435–453 (2015). https://doi.org/10.1017/S0269888915000090
20. Searle, J.: Making the social world: The structure of human civilization. Oxford University Press, Oxford (2010)
21. Searle, J.R.: The Construction of Social Reality. Simon and Schuster, New York (1995)
22. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative and procedural goals in intelligent agent systems. In: International Conference on Principles of Knowledge Representation and Reasoning. Morgan Kaufman (2002)
23. Wooldridge, M.: An Introduction to Multiagent Systems. Wiley, Hoboken (2009)