UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Rafhael Rodrigues Cunha

Using Purpose to Specify the Environmental Consequences of Artificial Institutions

Florianópolis
2023

Rafhael Rodrigues Cunha

Using Purpose to Specify the Environmental Consequences of Artificial Institutions

Tese submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do título de doutor em Engenharia de Automação e Sistemas.
Supervisor:: Prof. Jomi Fred Hübner, Dr.
Co-supervisor:: Prof. Maiquel de Brito, Dr.

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Cunha, Rafhael Rodrigues
Using Purpose to Specify the Environmental Consequences of Artificial Institutions / Rafhael Rodrigues Cunha ; orientador, Jomi Fred Hübner, coorientador, Maiquel de Brito, 2023.
137 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Sistemas Multi agentes. 3. Instituições Artificiais. 4. Ações Institucionais . 5. Propósitos. I. Hübner, Jomi Fred . II. de Brito, Maiquel. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

Rafhael Rodrigues Cunha

Using Purpose to Specify the Environmental Consequences of Artificial Institutions

O presente trabalho em nível de doutorado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof.(a) Gustavo Alberto Giménez Lugo, Dr.
Instituição DAINF/UFTPR

Prof.(a) Viviane Torres da Silva, Dra.
Instituição IBM Research Brazil

Prof.(a) Eder Mateus Nunes Gonçalves , Dr.
Instituição C3/FURG

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de doutor em Engenharia de Automação e Sistemas.

Documento assinado digitalmente
JULIO ELIAS NORMEY RICO
Data: 17/03/2023 17:56:34-0300
CPF: ***.840.859-**
Verifique as assinaturas em https://v.ufsc.br

Coordenação do Programa de Pós-Graduação

Documento assinado digitalmente
Jomi Fred Hubner
Data: 17/03/2023 17:46:58-0300
CPF: ***.334.769-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Jomi Fred Hübner, Dr.
Supervisor:

Documento assinado digitalmente
Maiquel de Brito
Data: 17/03/2023 13:29:20-0300
CPF: ***.657.150-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Maiquel de Brito, Dr.
Co-supervisor:

Florianópolis, 2023.

À Milena, Maria, Andressa e Isadora,
maiores amores que tenho na vida.

ACKNOWLEDGEMENTS

Agradeço primeiramente à Deus, por sua infinita bondade, em me proporcionar, com saúde e condições físicas, emocionais e financeiras, todos os dias necessários para à realização deste trabalho.

A minha noiva e parceira de vida, Milena, que esteve comigo durante todo esse tempo, chorando junto comigo nos momentos de dificuldade e vibrando e sorrindo em tantos outros de alegria. Agradeço por sempre se fazer presente, seja através de carinhos, mensagens de afeto ou até mesmo de incentivo. O teu apoio foi fundamental para que este trabalho pudesse ser realizado e concluído. Obrigado por compreender os tantos períodos de ausência que se fizeram necessários para a realização deste trabalho e por me dar o melhor presente da vida. Estendo esse agradecimento aos seus pais, meus sogros, Ari e Juraci Silveira, pelo afeto e confiança.

Ao nosso cachorro, Lipinho, amigo peludo que foi meu parceiro nas incontáveis horas de estudos, escrita de artigos, apresentações de trabalhos em eventos e escrita da tese. Lipinho me mostra todos os dias que não precisa de palavras para expressar amor, carinho e companheirismo.

A minha família, em especial a minha mãe, Maria, por sempre me incentivar a trilhar o caminho do bem, seja através do seu próprio exemplo, seja por meio de seus esforços em nos proporcionar acesso ao estudo e a educação. Sou e serei eternamente grato por todo teu estímulo durante todos esses anos. As minhas irmãs, Andressa e Isadora, por estarem comigo em todos os momentos necessários. Por fim, estendo meu agradecimento aos demais membros da minha família, devidamente representados através das tias Marla e Lucia, que endereçaram à mim diversas mensagens de estímulo e carinho no decorrer desta jornada.

Aos professores e orientadores Jomi e Maiquel, por aceitarem me conduzir nessa caminhada e terem realizado suas funções com maestria, alertando-me quando as minhas ideias eram equivocadas ou apoiando-me sempre que elas eram relevantes. Grato também pela disponibilidade e parceria em inúmeras reuniões, revisões de escrita de artigos, apresentação de trabalhos e escrita da tese. Por fim, mas não menos importante, sou grato por me mostrarem através de exemplos e incentivos que sempre podemos entregar mais do imaginamos. Vocês me ensinaram muitas coisas nesses anos que eu jamais encontrarei em livros. Obrigado por fazerem essa jornada valer a pena!

Ao professor e amigo, Roger Heinrich, que em pouco mais de dois anos, elevou meu nível de inglês de semianalfabeto para entusiasta na apresentação de trabalhos em eventos científicos estrangeiros.

Aos irmãos da vida, Marcos, Clau, Dudu e Edinho, pela parceria nos diferentes momentos da minha vida, em especial à aqueles em que a tensão da realização deste trabalho foi trocado pela leveza de conversas descontraídas.

Ao Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS), em especial ao campus Vacaria, pela concessão do afastamento integral, essencial para a realização

deste trabalho e, ao governo federal que, através de suas políticas públicas, oportunizou a realização de tantos sonhos.

"Faça o seu melhor, na condição que você tem,
enquanto não tem condições melhores,
para poder fazer melhor ainda."
(Cortella, Mario Sérgio, 2016)

RESUMO EXPANDIDO

USANDO PROPÓSITOS PARA ESPECIFICAR AS CONSEQUÊNCIAS AMBIENTAIS EM INSTITUIÇÕES ARTIFICIAIS

Introdução. Sistemas multi-agentes abertos têm incorporado mecanismos sociais para tratar de algumas questões tais como a conciliação da autonomia dos agentes com os interesses do sistema, a orientação dos agentes em agir em sistemas para os quais agentes não foram inicialmente projetados, etc. Um mecanismo social frequentemente usado é a realidade institucional que, quando adaptada para um sistema multi-agentes, traz a noção de instituição artificial (ou simplesmente instituição). Instituições consideram que alguns fatos que ocorrem no ambiente constituem (ou contam como) fatos institucionais. Por exemplo, individuos agindo em um sistema de comércio eletrônico constituem (ou contam como) *compradores* na instituição, enquanto algumas de suas ações podem contar como *pagamentos*. Essas ações são conhecidas na literatura como *ações institucionais* porque elas produzem efeitos no ambiente devido ao significado atribuido a elas pela instituição. Além disso, essas mudanças podem combinar com o interesse dos agentes. Por exemplo, um agente pode realizar alguma ação que conta como *pagamento* (por exemplo, entregar uma nota de papel) se a consequência de constituir pagamento satisfaz seu interesse (por exemplo, adquirir um livro). Enquanto modelos de instituição artificial tratam do processo de constituição através da atribuição de funções de status a fatos que ocorrem no ambiente, o mesmo não acontece com relação aos efeitos que o processo de constituição produz no ambiente.

Objetivos. O objetivo dessa tese é propor um modelo que torne explícitas as consequências no ambiente da execução de ações institucionais. Para atingir esse objetivo, toma-se, como inspiração, a teoria do filósofo John Searle, que observa que os status atribuídos às ações concretas permitem que essas ações executem funções que não podem ser explicadas por suas virtudes físicas. Searle afirma que essas funções são atribuidas para satisfazer os interesses práticos dos agentes que são nomeados por ele como *propósitos*. Para atingir o objetivo dessa tese, é necessário definir uma abstração apropriada para representar os propósitos, meios para acoplar tal representação às representações de realidade institucionais existentes e formas de os agentes se beneficiarem dessa representação, utilizando-a no seu processo de raciocínio e deliberação para atingir seus interesses.

Contribuições. Esta tese propõe um modelo de propósitos para expressar as consequências de ações institucionais no ambiente em que os agentes atuam. A partir desse modelo, define-se que seu acoplamento com os modelos de realidade institucional existentes ocorre por meio das funções de status, pois são os status que possibilitam a execução das funções que modificam o mundo para atender aos propósitos dos agentes. Além disso, os propósitos são relacionados aos objetivos dos agentes e, a partir disso, eles podem ser considerados no seus processos de raciocinio e deliberação para satisfazer seus interesses. Com o modelo proposto, os agentes podem acessar e raciocinar sobre as consequências das ações institucionais e adaptar-se a diferentes cenários. Eles podem perceber que (a) alguns propósitos apontam para estados do ambiente que correspondem aos seus interesses e, portanto, úteis para alcançar seus objetivos ou (b) evitam esses propósitos porque apontam para estados semelhantes aos seus anti-objetivos. Em ambos os casos, o agente têm mais informações para decidir se uma determinada ação o

ajudará ou não a satisfazer seus interesses. O modelo proposto permite aprimorar os agentes nos aspectos de adaptabilidade, racionalidade e flexibilidade, além de tornar explícito esse conceito e suas relações.

Conclusões. O modelo de propósito proposto concebe abstrações apropriadas para capturar as consequências no ambiente da execução de ações institucionais. A partir desse modelo, é possível conceber instituições artificiais que permitam expressar tanto as consequências institucionais das ações executadas no ambiente quanto as consequências ambientais que tais execuções acarretam. Tem-se, assim, instituições que podem auxiliar agentes a executar ações que podem satisfazer seus interesses.

Palavras-chave: Propósitos. Ações Institucionais. Instituições Artificiais.

ABSTRACT

Open multi-agent systems have incorporated social mechanisms to address issues such as reconciling agents' autonomy with the system interests, the orientation of agents to act in systems for which they were not designed, and so on. A social mechanism frequently used is the institutional reality that, when adapted to the MAS, brings the notion of artificial institution (or simply *institution*). Institutions consider that some concrete facts occurring in the environment constitute (or count as) institutional facts. For example, individuals acting in an e-commerce scenario may constitute (or count as) buyers in the institution, while some of their actions may count as payments. These actions are known as institutional actions because they are actions performed in the environment that produce consequences in the institution. Furthermore, they can produce consequences in the environment resulting from the consequences that occur in the institution. These environmental consequences may match the interests of the agents. For example, an agent can perform some action that counts as payment (e.g., delivering a paper note) if the consequence of constituting payment satisfies its interest (e.g., holding a book in hand). While current institutional models provide information about constitutions, the same does not happen about the consequences of the constitution in the environment. Having an explicit representation of the environmental consequences of institutional actions is an open question in the design of multi-agent systems. In response to this issue, this thesis proposes a purpose model that represents the environmental consequences of institutional actions. Furthermore, it presents some algorithms that agents can use to consider this information in their reasoning and deliberation process. The proposed model is positioned between existing frameworks that use social abstractions to support incoming agents and evaluated through application examples highlighting the disadvantages and advantages of the model in designing a multi-agent system.

Keywords: Purposes. Institutional Actions. Artificial Institutions.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
ASP	Answer Set Programming language
BDI	Belief, Desire and Intention
ICT	Information and Communication Technologies
MAS	Multi-Agent Systems
SAI	Situated Artificial Institution
SWRL	Semantic Web Rules Language

LIST OF SYMBOLS

$\mathcal{T}$	set of all properties that the system can present
$\mathcal{S}$	set of all possible states of the MAS
$\mathcal{E}$	The set of all events that may happen in the environment
$\mathcal{A}$	The set of all agents that can act in the MAS
$\mathcal{G}$	The set of all agents and their goals
$\overline{G}$	The set of all agents and their anti-goals
$\mathcal{F}$	The set of all the event-status-functions
$\mathcal{C}$	The set of all constitutive rules
$\mathcal{P}$	The set of all purposes
$\mathcal{F}_P$	The set of all relations between status-functions and purposes
$\mathcal{G}_P$	The set of all relations between purpose and goals and anti-goals

CONTENTS

1 INTRODUCTION

Recent advances in computer technology have enabled computing to move from a single computer (where computing is a process driven by the CPU) to computer networks (where computing is driven by the interaction of distributed computing units) (WHITWORTH, 2011). This new paradigm, dubbed “Computing as Interaction" (LUCK; MCBURNEY; PREIST, 2003), promotes a certain degree of autonomy and openness among diverse software entities (DEMOLOMBE, 2010). These systems are classified in the literature as open systems because they are decentralized (i.e., without a centralized decision), concurrent (i.e., multiple components run concurrently with others) and loosely coupled (i.e., neither component has access to the internal state or structure of the other). Examples of open systems include Wikipedia and their bots (GEIGER, 2009, 2018), financial markets and their algorithm traders (FARJAM; KIRCHKAMP, 2018), smart-grids (MERABET et al., 2014), collaborative platforms for crisis management (THÉVIN et al., 2015), e-commerce systems (ARANDA-CORRAL; DÍAZ; MARTÍN, 2015), auction organizer or contract negotiations (ESTEVA et al., 2008; GOVERNATORI et al., 2001; SARDIS; VOUROS, 2007), and social network systems (BERGENTI; FRANCHI; POGGI, 2012; CALVARESI et al., 2019; ABREU, 2021; PÉREZ-MARCOS et al., 2020; AMARAL; HÜBNER, Jomi Fred, 2019).

Artificial Intelligence (AI) in general, and Multi-Agent Systems (MAS) in particular, have developed approaches to implement this type of open, decentralized system (OSSOWSKI et al., 2007). According to Felicissimo et al. (FELICÍSSIMO et al., 2006), an Open MAS can be considered an open system when it is made up of agents that may act autonomously (GONZÁLEZ-BRIONES et al., 2018), are created and implemented by different parties (FORNARA; VIGANÒ; COLOMBETTI, Macro, 2004) and can interact with one other (WOOLDRIDGE, 2009). Open MAS are thus decentralized, loosely coupled, and concurrent. Besides agents, open MAS may include other components, such as environment and institutions.

In Open MAS, the concept of environment is fundamental. It is usually conceived as the set of non-autonomous elements (e.g., sensors and actuators, printers, networks, databases, web services, etc.) that are perceived and acted upon by the agents (RUSSELL, 2010; WEYNS; OMICINI; ODELL, 2007; RICCI; PIUNTI; VIROLI, 2011). In other words, the environment is the computational or physical place where agents are *situated*. Some fundamental features of the agent abstraction are directly or indirectly related to the environment: reactivity is an obvious example, but also pro-activeness, referring to the interests of agents that can be reached by resources available in the environment (ARGENTE et al., 2013).

Open MAS may incorporate social mechanisms to address issues such as the conciliation of agents' autonomy and system goals, the orientation of agents to act in systems for which they were not designed, etc. (WHITWORTH, 2011). In this regard, the institutional reality is an important resource present in human societies that has been implemented in open MAS. There are several theories and scholars (TESTA, 2017; HINDRIKS, F., 2015; SMIT; BUEKENS;

DU PLESSIS, 2014; SEARLE, J. R., 1995; SEARLE, J., 2010) that attempt to explain this portion of reality that arises from the interpretation, by individuals, of the concrete world composed of people, actions, etc. They usually consider that some concrete facts occurring in the environment *constitute* (or *count as*) institutional facts (CLIFFE; VOS; PADGET, 2006; CARDOSO, HENRIQUE LOPES; OLIVEIRA, EUGÉNIO, 2007; BRITO, M. d. et al., 2016; FORNARA, 2011). In Open MAS, these facts usually give special meaning to concrete facts related to the application domain. For example, individuals acting in an e-commerce scenario may constitute (or count as) *buyers* in the institutional reality, while some of their actions may count as *payments*. When adapted to MAS, the existence of an institutional reality brings the notion of the artificial institution (or simply *institution*). Several works (CLIFFE; VOS; PADGET, 2006; CARDOSO, HENRIQUE LOPES; OLIVEIRA, EUGÉNIO, 2007; DE BRITO; HÜBNER, Jomi Fred; BOISSIER, 2018; FORNARA, 2011) use the notion of the institution as a first-class abstraction[1] to represent the relation between institutional facts and concrete facts present in the environment. That is to say, from a system composed of entities such as agents, web services, messages, databases, etc. (ultimately conceived to be deployed as software pieces), the institution brings into the system elements such as buyers, payments, etc.

Independent of the open MAS design, heterogeneous agents can enter the system at any time, use its infrastructure to satisfy their interests, and eventually exit (PITT; MAMDANI; CHARLTON, 2001; WOOLDRIDGE, 2009). We assume that *agents' interests* involve achieving desired situations (henceforth referred to as goals) and avoiding undesired situations (henceforth referred to as anti-goals). Some of these interests are satisfied by actions performed within institutions. Consider the following scenario: an agent is designed to achieve its goal of publishing information on the social network Twitter. In this system, some concrete action (e.g., sending a string to a server, uploading a photo, etc.) that counts as *tweet* results in *publishing information*. This result (i.e., the agents' interest) is a consequence of the constitution of *tweet*. Therefore, if the agent performs some concrete action that counts as *tweet*, it satisfies its goal. The physical virtues of the action can not explain such a result. Instead, it is the consequence of the action to be performed within an institutional context.

1.1 PROBLEMS

Some actions in the institution have particular consequences due to their constitutions and not the action itself. These actions are known in the literature as *institutional actions* (SEARLE, J. R., 1995; SEARLE, J., 2010). The consequences of carrying out institutional actions may imply concrete changes in the environment. Figure 1 depicts these two connections between the environment and the institution. The first connects the environment to the institution (through count-as rules), and the second connects the institution to the environment (through the effects) in the opposite direction.

[1] According to Netto (ABREU NETTO, 2010), the first-class abstraction is a block that encapsulates its responsibilities independently of the agents.

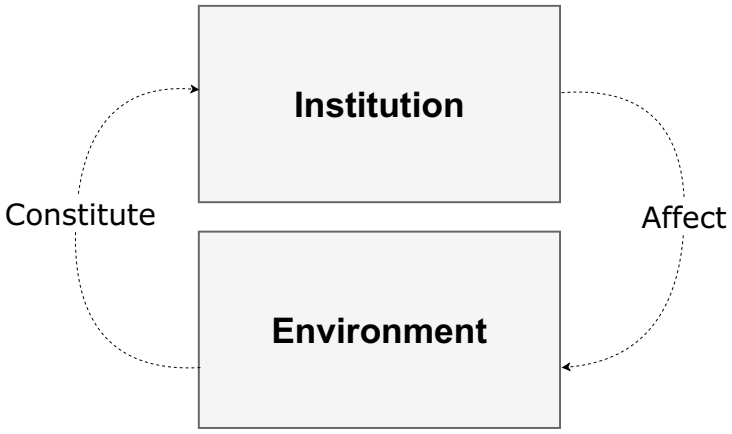

Figure 1 – Relations between environment and institution.

For agents to act appropriately in institutions, they should know both connections. For example, an agent can perform any action that counts as *tweet* (e.g., sending a string) if the consequence of constituting *tweet* satisfies the agent's interest (e.g., publishing information). While current institutional models support the first connection, the same does not happen concerning the second. Having an *explicit* representation of the environmental consequences of institutional actions is a key issue for designing agents capable of satisfying their interests in systems composed of institutions. The problem addressed in this thesis is *the design of artificial institutions that not only specify count as relations, but that also have the means of specifying the consequences in the environment of the execution of institutional actions to be exploited by the agents*.

From the outlined, we can see it as a threefold problem. It requires a definition of (i) what are the appropriate abstractions to represent the consequences in the environment of institutional actions, (ii) how the abstractions that represent institutional reality and the environment can be coupled with the representation of the environmental consequences of institutional actions and (iii) how agents can exploit them to act according to their interests.

Regarding the first point (i), the consequences in the environment of institutional actions are not sufficiently represented by the elements that are within the institution (i.e., constitutive rules, status functions, etc.). They are also not sufficiently represented by the elements that are in the environment because, even though these consequences reflect on the environment, they are associated with elements of the institution (which are outside the environment). The representation we are looking for, in a way, connects elements of these two dimensions of the MAS (institution and environment). The current state of the art does not have an appropriate abstraction to represent the consequences in the environment of institutional actions. For this reason, these consequences are implicit. For example, consider the case of agents acting on Twitter with the instruction to perform some action that counts as *tweet*. A priori, these agents can know the effects of *tweet* if the relation between *tweet* and its consequences in the environment are encoded within them.

Regarding the second point (ii), assuming that an external abstraction is needed to

connect the institution with the environment (first point), the problem is in defining (a) which types of institutional abstractions, among all possible ones, can have this associated external abstraction; (b) which, among all the brute facts, can represent the consequences associated with institutional actions; and (c) how the proposed representation connects with those institutional abstractions and with the brute facts. The lack of this connection raises some problems. First, agents can only act in institutions whose status functions are known to them. For example, consider again the case of agents acting on the social network Twitter. If, for some reason, the status function *tweet* is replaced by the status function *post*, agents may have difficulty acting in this system. Second, institutions need to specify all status functions used by all agents that potentially will act on them. Consider that the status function *post* brings the same consequence as the status function *tweet* when constituted. As this connection does not exist, the institution needs to add the *tweet* status function again so that agents can act in the system.

Regarding the third point (iii), assuming that the agents should reason about the external abstraction to act according to their interests, the lack of this abstraction and its connection with institutional abstractions and brute facts may cause some problems for the agents. They may have difficulty reasoning about institutional actions that result in (a) desired situations and (b) undesired situations. Consider two scenarios involving the example where agents are designed to achieve their goal of publishing information on the social network Twitter. In the first scenario, agents do not know what action they should perform to satisfy their goals. In the second scenario, the agents know that they should perform an action that counts as *tweet* to satisfy their goals, but they do not know that this action also produces the undesired outcome of spreading fake news. In the first scenario, these agents achieve their goals only if they are previously coded with the institutional action they should perform. In the second scenario, agents avoid the undesired situation only if the consequences of institutional actions are encoded within them.

In short, the existing works on artificial institutions are mainly concerned with specifying and managing the institutional interpretation of facts occurring in the environment (shown on the left arrow of Figure 1). However, institutional actions may enable new facts in the system that potentially lead to environmental changes. These environmental consequences of the constitutions remain unexplored (shown on the right arrow of Figure 1). Some related problems are the difficulty of predicting the environmental consequences arising from the execution of institutional actions and the limitation in the agents' reasoning about satisfying their interests that involve the performance of institutional actions. Having an artificial institution that is independent but connected to the consequences that it can bring to the environment is an open question for the open MAS.

1.2 HYPOTHESIS

An inspiration to answer these questions is the social reality theory by John Searle (SEARLE, J. R., 1995; SEARLE, J., 2010). He considers that the abstract statuses attributed to concrete elements carry a set of functions that their physical virtues cannot explain. Searle calls them status functions. Searle claims that the functions associated with statuses are *agentive functions* because they are assigned to *satisfy the practical interests of agents*. Searle names the practical interests of agents as *purposes*. Searle's theory has indeed inspired some works on MAS. However, these works usually take inspiration only from a rough notion of "count as", i.e., facts from the environment have some status functions (or count as other facts) in institutional reality.

However, Searle's proposal is wider than assigning status functions to environmental elements. In addition, it proposes to represent an institutional structure that reflects in the environment the interests of the members who live in the society. Thus, Searle's theory seems to point a direction to answer the previous questions, as it has a conceptual apparatus to capture the notion of purpose arising from social arrangements that reflects the interest of society members in the environment.

1.3 OBJECTIVES

The main objective of this thesis is to develop a purpose model to express the environmental consequences of the execution of institutional actions. To achieve this general objective, some sub-objectives are considered:

1. To define suitable abstractions to represent purposes;
2. To define how models of artificial institutions and representation of the environment, proposed in the literature, integrate the proposed representation of purposes;
3. To define how the interests of agents, expressed through goals or anti-goals, are related to the proposed representation of purposes.

1.4 MOTIVATIONS

As already discussed, open MAS can have an institutional reality that allows a common interpretation of concrete facts attributing institutional facts to these elements. Having institutions where it is also possible to express the consequences of institutional facts in the environment brings some advantages to the design of Open MAS.

The concept of *purpose* is important to establish a common representation of how institutional actions affect the environment, establishing a connection between elements of these two dimensions of the MAS (institution and environment). For example, consider again the example of agents acting on Twitter with the instruction to perform some action that counts as *tweet*. In such a scenario, with the relationship between *tweet* and the consequence

of publishing information properly represented with the concept of *purpose*, agents do not need to have these consequences encoded within themselves and can consult them in the system at any time.

The concept of *purpose* can be associated with status functions on the institutional side and system states on the environmental side. We consider that agents act concretely within the institution through the institutional actions they perform. For this reason, the concept of *purpose* is associated with status functions. On the other hand, we consider that the states of the system are the types of facts relevant for the agents' interests. For this reason, the concept of *purpose* can be associated with (or point to) the states of the system that satisfy the interests of the agents. Finally, the connection between purpose, status functions and system states is established through institutional actions. Institutional actions, when performed, may have consequences in the environment that reflect the states of the system pointed out by the purposes. This connection brings some advantages to the system. First, agents can decide how act in the system even if they have no prior knowledge of the status functions. For example, consider again the case of agents acting on the social network Twitter. If agents are designed to act according to the consequences of institutional actions, it does not matter to agents whether they should constitute tweets, posts, etc. Second, institutions can be designed with status functions that may be unknown to agents at design time. Since the purpose makes the consequences of institutional actions explicit, the designer can specify institutions without worrying about compatibility with the agent's code. The compatibility is moved to an appropriate concept, making the institutional specification more stable.

The concept of *purpose* brings some advantages to agents, such as reasoning about institutional actions that result in (a) desired situations and (b) undesired situations. Knowing the actions available in the institution and their consequences, agents can identify which actions can satisfy their goals and anti-goals. In other words, making the consequences of institutional action explicit helps the agents better understand the actions they can perform, allowing them to make the best decision for their interests. The agent's capability to reason about the consequences of the institutional actions and adapt to different scenarios is an important advance, especially in open systems (ALDEWERELD, Huib; DIGNUM, Virginia, 2010; ZAMBONELLI; JENNINGS; WOOLDRIDGE, 2000).

1.5 DOCUMENT STRUCTURE

In the following, Chapter 2 presents the theoretical elements required to achieve the goals of this work, including the state of the art. These elements underlie the purpose model proposed in this work. Chapter 3 presents the model of purposes and algorithms that agents can use to satisfy their interests. This chapter also presents a proposal to implement the model in a data structure and integrate the model with the MAS development framework called JaCaMo. Chapter 4 presents application examples that help us to evaluate the proposal of this work. This chapter also presents a position between the proposed model and existing frameworks

that use social abstractions to support incoming agents. Finally, Chapter 5 summarizes the contribution of this work, pointing some perspectives of future work.

2 BACKGROUND AND STATE OF THE ART

From a social point of view, Information and Communication Technologies (ICT) have gone from being a tool to foster productivity in a company to a mediator in social relationships (WHITWORTH, 2011). Research in MAS has adopted (and adapted), in different ways, concepts such as norms, institutions, etc. to enable such mediation. Among all the social aspects involved in agent societies, this chapter focuses on those relevant to this thesis. More precisely, it looks at works that conceive artificial institutions based on the theory of John Searle. Considering the problems posed in Section 1.1, relevant aspects of Searle's theory on institutions are described in Section 2.1. Moving to the MAS field, Section 2.2 describes how institutional reality has been used in works in the area. Section 2.3 describes how agents have used institutional reality to reason about their interests. Sections 2.2.3 and 2.3.1 discuss some open questions in the area that have inspired the development of this thesis.

2.1 INSTITUTIONS ACCORDING TO JOHN SEARLE

One of the main questions motivating Searle's theory is: How can we, if at all, reconcile a certain conception of the world described by physics, chemistry or other basic sciences with what we know or think we know about ourselves as human beings? (SEARLE, J., 2010, p.3). In other words, how from a world formed by elements that can be described by the basic sciences, one arrives at a world composed by more concrete elements (presidents, money, universities, etc.)? The explanation for these questions lies in the existence of two types of facts: brute and social.

Searle argues that there are facts that are explained by basic science, such as water being composed of hydrogen and oxygen, that do not depend on any mental attitude of agents to exist. These facts are considered *brute facts*, as their existence is independent of any perception or mental state. Examples of these elements are a piece of paper, a mountain, ice, etc. However, according to Searle, there are facts in the world that are also objective but that exist only because we believe in their existence (e.g., money and president). The basic sciences cannot explain that a piece of paper is money and that some citizen is the president. For a piece of paper to be recognized as money or a human being as president, it is necessary for members belonging to a community to collectively accept or recognize that the object, person, etc. is regarded as money, president, etc. For this reason, the facts of a piece of paper being considered money and a person being considered a president are considered *social facts*.

Social facts make up social reality and need collective intentionality to exist (SEARLE, J., 2010). Intentionally is the term used for the mind's ability to direct its will about something (e.g., objects, states of the world, etc.). For example, an intentional attitude is to prefer cabernet sauvignon over pinot noir. In this case, i am in an intentional state. Intentional states are always about, or refer to, something. When intentional states are shared by members of a community, there is collective intentionality. For example, for a piece of paper to be

considered money it is necessary for people to share intentional states, such as beliefs, desires and intentions (SEARLE, J. R., 1995, p.24).

A subclass of social facts are institutional facts. These are facts that exist only within human institutions. For example, two people moving pieces on a board randomly is a social fact. Two people respecting rules and moving pieces according to their meanings on a board can be considered a game, and in this case, it is an institutional fact, because they are in the context of an institution (i.e., the game of chess). The institution — the game of chess — assigns meaning and purpose to the pieces and moves performed in the game. Without the institution, there would be no moves like checkmate, Zug or pieces like queen, king, etc. The *Institutional Reality* is part of the *Social Reality*. According to Searle, human institutions are based on the following elements:

- **Status Function:** Human beings have the ability to impose functions on objects or people that enable them to perform activities that go beyond their physical virtues.In order to perform these functions, the element must have a collectively recognized status in the society. For example, an individual may have the status (and hence the role) of *professor* at a university. Those statuses that assign functions to elements of the concrete world, such as *professor*, are called *Status Functions*.

- **Collective Intentionally:** For a *Status Function* to be effective within the institution, there must be collective acceptance or recognition that the object, person or other type of entity has the *Status Function* assigned. Therefore, *Status Functions* depend on collective intentionality. Only by virtue of this, a piece of paper can be regarded as a bill of money or a person as the president of a country (SEARLE, J., 2010, p.9). Although the phenomenon is collective, intentionality exists only within individual human brains. Therefore, it is necessary that the agents coexisting in the society assume that the other members also share the same beliefs about the *Status Function*.

- **Deontic Powers:** An important element for the functioning of societies is what Searle calls deontic powers (SEARLE, J. R., 1995; SEARLE, J., 2010). Without exception, *Status Functions* carry deontic powers. These are rights, duties, obligations, etc. It defines what are the expected behaviors and the ones that should be avoided of the members in a society. For example, an individual may constitute the status (and consequently have the function) of *professor* at a university. However, along with the status of *teacher*, some deontic powers linked to the status are defined such as *teach*, *correct exams*, *write assignments*, etc. that individuals with the status of *teacher* should follow or avoid.

- **Actions independent of desires and physical constraints:** The combination of *Status Functions* and deontic powers constitutes an effective system that supports human societies (BRITO, M. d. et al., 2016). According to Searle, *Status Functions*

and deontic powers are the "glue that holds society together" (SEARLE, J., 2010, p.9). The claim is justified by the fact that *Status Functions* always carry deontic powers and these always provide reasons to act independently of desires and physical constraints implemented in the brute reality. For example, a high wall can be a physical restriction to prevent unwanted people from entering private property. But if the wall is replaced by a line of stones signifying the boundary demarcations of property and the legislation institutes a deontic power that prohibits individuals from entering private property without the owner's permission, these citizens have reason not to cross the stone line, although they are physically capable of doing so.

- **Constitutive Rules:** The constitutive rules define the constitution of *Status Functions*. They have the form: *X count-as Y in context C*. For example, in the game of chess, a player making a final move, i.e., covering his opponent's king (X) count-as *checkmate* (Y) in the board context (C). This behavior not only regulates the movement of the piece around the board, but acting in accordance with this rule is also a logically necessary condition for playing chess, since the game does not exist beyond the rules. For example, driving on the right is a rule that regulates behavior, but driving exists independently of the rule. Other rules, constitutive rules, not only regulate but also create the possibility of much of the behavior they are regulating (SEARLE, J., 2010, p.9).

Through these elements it is possible to define institution. According to Searle, *institutions* are systems of *constitutive rules* that enable the existence of *Institutional Reality* (SEARLE, J., 2010). The institutional reality is an interpretation, on the part of the *institution*, of the facts occurring in the environment. By systems of *constitutive rules*, Searle means that *institutions* define, through constitutive rules, a particular interpretation of brute reality, expressed in terms of assigned *Status Functions*, to base the *deontic powers*. These exist within *institutions* and only when the *Status Function* is assigned. For example, the rule "the president should enact a new law during his term of office" is effective only when the institution constitutes *president* and *law*. Therefore, in addition to *institutions* being systems of *constitutive rules* that enable *institutional reality*, they also define the expected behavior of individuals. Institutions perform these tasks through the following operations:

- **Creation of *Status Function*:** The *Status Functions* are not elements that exist by themselves in the world. They need to be created. In (SEARLE, J., 2010, p.59), Searle argues that functions are always relative to intentionality (hence mind-dependent) and are also a cause *serving a purpose*. In (SEARLE, J. R., 1995, p.20), Searle argues that some functions are called *agentive functions* because they are assigned from *practical interests of the agents*.
- **Definition of Purpose:** Looking at the definitions of status functions, it seems clear that these functions are assigned to satisfy the practical interests of the agents involved in that institution. Searle calls these interests of *purpose*. According to him,

these functions and their purposes must be created based on the intentional states (i.e., it requires collective intentionality) present in the minds of the individuals interested in their creation. Searle also argues that individuals must be *able to understand what the thing is for*, or the function could never be assigned (SEARLE, J. R., 1995, p.22). Understanding a function requires understanding what it is for (i.e., its purpose). For example, in the game of chess, when the player moves some piece with assigned status by performing the “checkmate" function, the purpose of this function is to *win the game*. The purpose is in line with the interests of the agents who are playing the game (that is, it is understood by the people involved in the institution).

- **Constitution of *Status Function*:** The constitutive rules define the constitution of *Status Function*. This constitution can occur in two ways:

  Assigning a *Status Function* Y to a concrete element X: In traditional cases, a *Status Function* Y is assigned to an element of the concrete world X. Element X can be explained by the basic sciences and is therefore a brute fact. It constitutes an institutional fact only after the assignment of the Y status, which can occur in a generic or specific way:

  1. Generic: When a *Status Function* Y is assigned to a generic element X, that is, any element X that satisfies a set of conditions, enabling it to receive the status Y. For example, in any electoral scenario element that wins the elections and meets the specific requirements of the position can count as president (Y) in the context of their country (C).
  2. Specific: When a *Status Function* Y is assigned to a specific element X, that is, an element X that has been chosen to have the status Y without considering a set of conditionals equal to those present in the generic form. Exemplifying this condition, element Bob (X) can count as president (Y) in the context of his country (C).

  Independent assignment: The independent assignment is called *Freestanding Y* by Searle. It encompasses cases where a *Status Function* exists but is not assigned to any element of the concrete world. In (SEARLE, J., 2010, p.98) an example of the creation of a company by decree is presented. In the California constitution, there are some rules that enable any entity that satisfies certain conditions can create a company by performing only a declarative speech act, and then that company will exist in perpetuity unless some other condition occurs. It simply begins to exist within the institution, being created by decree. But the fact is that there are not only people or groups of people that can be considerations as a company. In general, companies have buildings and equipment that also constitute it. However, even if

there is a change of personnel within the company, it remains in existence, therefore not being physically real.

- **Power Creation**: As described above, all *Status Functions* carry deontic powers. These powers are deontologies that correspond to rights, duties, obligations, etc. So, power creation relates this set of rights and duties to *Status Functions*. For example, the teacher has some obligations that other members of society do not have, such as teaching classes, correcting tests, preparing papers, etc.

2.2 INSTITUTIONS IN MAS

Works on MAS have proposed computational counterparts of human institutions, referred to in this work as *artificial institutions*. In different ways, these works use abstractions proposed by Searle to represent institutional reality in computer systems. This section reviews the state of the art in this topic. The question to be answered is: How do artificial institutions deal with the consequences of institutional facts in the environment?

The research source used in this work is Periódicos Capes.[1] This search engine indexes many world-renowned databases, such as IEEEExplore, ACM, ScienceDirect, Scielo, among others.

Inclusion and exclusion criteria were used to properly select works that relate concepts on MAS to social reality theories. The inclusion criteria are: i) articles in the area of MAS that present a computational model of some theory of Social reality; ii) articles with more than six pages, containing introduction, development and conclusions. The exclusion criteria are the following: i) articles that are not complete, that is, without introduction, development or conclusion; ii) articles that do not develop theories of Social Reality and iii) articles that do not belong to the MAS area. Inspired by the classification proposed in (BRITO, M. d.; HÜBNER, Jomi F., 2014) and the work carried out in (CUNHA; HÜBNER, Jomi F; BRITO, M. de, 2019), the results are divided into two groups presented in Sections 2.2.1 and 2.2.2. The first group, includes models that relate brute facts to changes in the state of some social mechanism (e.g., norms, organization, etc.). The second group includes models that relate brute facts to social abstractions related to the application domain, analogous to the status functions proposed by Searle.

2.2.1 Institutional Reality as a functional Issue

Some works use concepts of institutional reality from a functional perspective. These works use institutional abstractions to solve the interoperability gap between the environment and the social representations. The *count-as* relation is used to specify changes in the state of some social mechanism (e.g., norms, organization, etc.). For example, some brute fact occurring in the environment count-as the violation of a norm.

[1] https://www.periodicos.capes.gov.br/

The approach of Campos et al. (CAMPOS et al., 2009) proposes an adaptation mechanism for electronic institutions that employ staff agents named governors. They monitor the facts occurring in the environment defining whether these facts count as something from the regulative perspective. In short, this work represents the *count-as* rules within the staff agent itself. This agent inspects the environment and interprets these rules, connecting environment with norms.

The approach of Dastani et al. (DASTANI et al., 2009) proposes a programming language to represent norms and relate them to facts in the environment. *Count-as* rules are used to represent norms. For example, *an agent on the train without a ticket* count-as *a violation of the norm*. In short, this work considers that *count-as* rules link actions that agents perform to normative consequences (e.g., violation, prohibition, etc.).

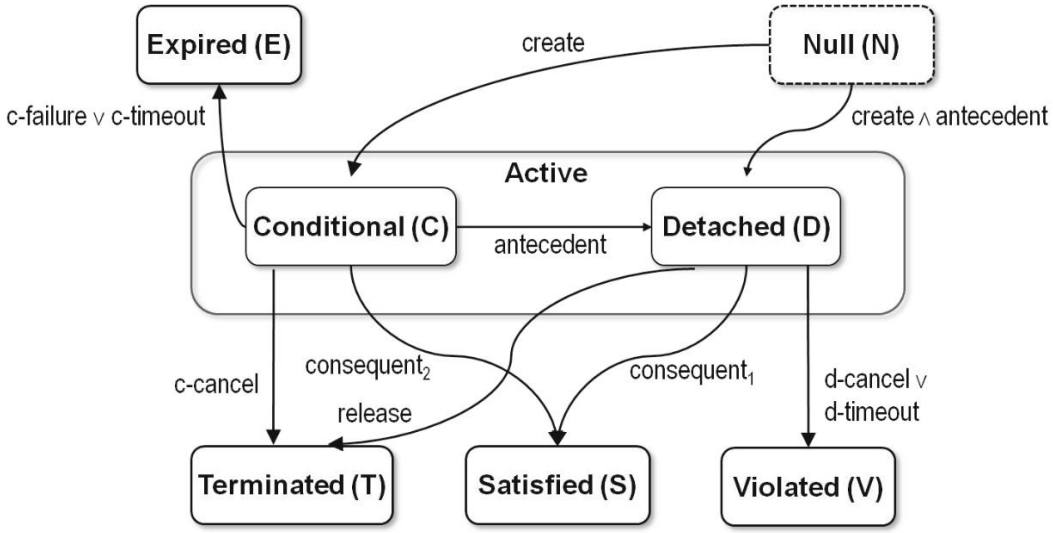

Figure 2 – Possible commitment lifecycle state transitions (DASTANI; VAN DER TORRE; YORKE-SMITH, 2012)

$$1 \quad offer(x,y,p,q,d_1,d_2) \Longrightarrow_{cr} C^c(x,y,p,q,d_1,d_2)$$
$$2 \quad tell(x,y,q) \wedge C^c(x,y,p,q,d_1,d_2) \wedge \neg d1 \wedge q \Longrightarrow_{cr} C^s(x,y,p,q,d_1,d_2)$$

Figure 3 – Constitutive rules for commitments (DASTANI; VAN DER TORRE; YORKE-SMITH, 2012)

In a similar direction, the approach of Dastani et al. (DASTANI; VAN DER TORRE; YORKE-SMITH, 2012) proposes an operational semantics for agent interactions within organizational settings (e.g., online marketplace). *Count-as* rules relate brute facts to commitment states. Figure 2 illustrates the states of a life cycle of a *commitment* in this work. Figure 3 illustrates some examples of constitutive rules following this approach. The line 1 defines that the offering of an agent x to an agent y to perform q before the instant d_2, conditioned to the performance of p by y before d_1, counts as a commitment from x to y with respect to q being in the state conditional (C^c). The line 2 specifies that the agent x informing to y

that he has been done q before d_1 counts as the satisfaction of the commitment, that moves from the state conditional to satisfied (C^s). In short, this work considers that *count-as* rules relate brute facts occurring in the environment to the states *conditional*, *expired*, *detached*, *terminated*, *satisfied*, or *violated* of commitments.

The approach of Piunti et al. (PIUNTI et al., 2010) proposes embodied organizations, which are organizations whose dynamics are animated by facts that occur in the environment. *Count-as* rules relate brute facts to operations on ORA4MAS artifacts, which are the technological support for the Moise organizational model (HÜBNER, Jomi F et al., 2010). Figure 4 shows an example of this approach. In the application where this snippet was extracted, the rule specifies that an object *Terminal* producing the event *sendFee* count-as the operation *setGoalAchieved* being executed in the artifact *monitorSchBoard*. This operation will cause the change in the state of an organizational objective. In short, this work uses *count-as* rules as triggers for operations on ORA4MAS artifacts.

```
+op_completed("Terminal",
    Ag, sendFee)
-> apply("monitorSchBoard",
    setGoalAchieved(Ag, send_fee)).
```

Figure 4 – Constitutive rule by (PIUNTI et al., 2010)

In a similar direction, the approach of Brito et al. (BRITO, M. d.; HÜBNER, Jomi F; BORDINI, 2012) proposes a model and a language to specify and program the institutional dynamics as consequence of events and state changes occurring in any of the two component dimensions of the system (environment and institution). *Count-as* rules define properties that the organization should have depending on the facts that occur and the states that prevail in the environment. The mechanisms that implement the organization are responsible for making it really have such properties. Figure 5 shows a constitutive rule specifying that the event *prepareSite* occurring in the environment count-as the property *goalState(bhsch,site_prepared,Ag,Ag,satisfied)* holding in the organizational platform. That is, the occurrence of the event *prepareSite* means achieving the organizational goal *site_prepared*. In short, this work considers that *count-as* rules relate brute facts occurring in the environment to properties holding in the organization.

```
+ prepareSite[agent_name(Ag),artifact_name(housegui)]
count-as
  goalState(bhsch,site_prepared,Ag,Ag,satisfied)[source(bhsch)].
```

Figure 5 – Constitutive rule by (BRITO, M. d.; HÜBNER, Jomi F; BORDINI, 2012)

2.2.2 Institutional Reality as an ontological issue

Some works treat institutional reality as an ontological issue. The constitutive rules are used to assign status to elements in the environment. For example, in a system that contains the norm *every PhD student is obliged to submit their qualification proposal within a period of up to 24 months*, the ontological approach defines what a PhD student is, what it means to submit a qualification proposal, etc. These works consider that brute facts count-as status functions while keeping an institutional specification, which is mostly applied in the specification of norms.

The approach of Fornara et al. (FORNARA; COLOMBETTI, Marco, 2009, 2010; FORNARA, 2011; FORNARA; TAMPITSIKAS, 2012) proposes to represent artificial institutions using ontologies, Semantic Web rules language (SWRL) and a Java program. Institutional events are the main concept of this approach. They are a particular type of event whose effect can change properties and elements of the institution itself. For example, the agent Robert can execute an action called *act01* at *instant2* that creates a new institutional space called *auction* if the action is successful. In this work, the authors consider that agents may try to execute institutional actions in specific spaces of the environment (which they call institutional spaces). The program is used to interpret the events and update the system if the pre-conditions established for the execution of the event are respected. In short, this work considers that the institutional reality is composed of institutional events that are a counterpart of events occurring in the environment.

The approach of Boella and van der Torre (BOELLA, Guido; VAN DER TORRE, 2004; BOELLA, Guido; TORRE, L. van der, 2004; BOELLA, Guido; TORRE, Leendert van der, 2006) proposes to represent the institutional reality using modal logic. The institutional specification describes states of the world, institutional facts, constitutive rules, etc. A brute fact (a fact that describes a state of the world) count-as an institutional fact if the normative agent considers that it is true in a specific context. For example, consider a society where the fact that *field has been protected by an agent count-as the fact that the agent owns the field*. In this example, the field protected by an agent is a brute fact, while the fact that someone owns it is an institutional fact attributed to the beliefs of the normative agent. In short, in this work, the representation of institutional reality depends on the normative agent's beliefs.

The approach of Vazquez-Salceda et al. (VÁZQUEZ-SALCEDA et al., 2008) proposes to represent the institutional reality using ontologies to define the concepts used in the specification of norms. For example, consider the norm "every citizen should be at least 18 years of age to be transplanted". The institutional reality defines concepts such as *citizen*, *age*, *transplant*, etc. The constitutive rules are also represented in the ontology. In these rules, the considered brute facts are events occurring in the environment.

The approach of Grossi et al. (GROSSI et al., 2006) also proposes to represent the institutional reality using ontologies. The authors developed the idea of contextual ontologies, formalizing some more general concepts into generic ontologies that can later be refined into

domain-specific ontologies. Subsequently, the specified concepts used in the specification of norms are related to concrete actions. In short, in this work, the objective of ontologies is to represent constitutive rules, specifying abstract concepts — in domain-specific ontologies — and later relating them to concrete actions — in more generic ontologies.

The approach of Cliffe et al. (CLIFFE; VOS; PADGET, 2006) proposes to represent the institutional reality using the answer set programming language (ASP). The model is based on concepts such as *observable events* — capture events from the physical world (e.g., shooting someone) — and *institutional events* — generated by society, with meaning only within it (e.g., murder). Count-as rules define events that occur in the environment count-as institutional events. In short, in this work, the institutional reality classifies the events that occurred in institutional events. The nomenclature of institutional events (e.g., shooting someone) is used in the normative specification. These represent violations and a set of institutional actions that must be performed as punishment.

The approach of Aldewereld et al. (ALDEWERELD, Huib et al., 2010) proposes to represent the institutional reality using Drools Tool[2] — an engine for declarative reasoning in Java. The representation proposed by them uses *count-as* rules to relate brute facts (i.e., events occurring in the system) to institutional facts and also institutional facts to normative consequences. For example, *exchange a message with content ok count-as inform* and *inform count-as norm reached*. In short, this work also only dynamizes the normative specification, classifying the events that occurred in institutional events and after then relating them to normative issues (norms reached, violated, etc).

The approach of Cardoso and Oliveira (CARDOSO, HENRIQUE LOPES; OLIVEIRA, EUGÉNIO, 2007) proposes to represent the institutional reality using ontologies. The representation proposed by them is directly related to the definition of contracts that make the agents' commitments explicit. For example, constitutive rules define which messages exchanged by agents count as contract formalization. In short, this work considers that the institutional reality is formed by some messages exchanged by agents that are categorized as institutional facts and are used to specify commitments.

The approach of Vigano and Colombetti (VIGANÒ, 2007; VIGANÒ; COLOMBETTI, Marco, 2008) proposes a framework to describe institutions using institutional concepts (*status functions*, institutional events, etc.) defined through a model. The representation proposed by the authors considers that the institution only imposes status functions to agents. These status functions are described in terms of deontic relationships, representing which actions are activated, required, etc., for an agent. In short, this work represents the institutional reality for assigning status functions to agents (and with it a series of actions) that are conditioned to normative issues.

The approach of Brito et al. (BRITO, M. d. et al., 2016; DE BRITO; HÜBNER, Jomi Fred; BOISSIER, 2018) proposes to represent the institutional reality through a model called

[2] https://www.drools.org/

Situated Artificial Institution (SAI). The representation proposed by them uses constitutive rules to assign status functions to elements present in the environment (i.e., agents, events and states). In short, this work represents the institutional reality totally uncoupled from normative issues.

2.2.3 Discussion

To discuss the presented works, it is important to remember the problem addressed in this thesis, which is *the design of artificial institutions that not only specify count as relations, but that also have the means of specifying the consequences in the environment of the execution of institutional actions to be exploited by the agents*. Artificial institution is a well-addressed topic in MAS. There are many approaches to representing the institutional reality that arises from interpreting elements present in the environment. However, these approaches do not support the representation of the consequences in the environment of carrying out institutional actions. There are some issues, discussed in Section 1.1, that illustrate this disadvantage, such as problems for agents to reason about the consequences in the environment of carrying out institutional actions, inconsistencies between the actions provided for in the agents' code and those available in the institutional specifications and conflicting institutional actions (which achieve goals and anti-goals at the same time).

The functional approach proposes that count-as rules are used to specify changes in the state of some social mechanism (e.g., norms, organization, etc.). However, as far as we know, no work in this category has proposed ways of relating brute facts to their environmental consequences. In fact, in our view, these works do not consider the constitution of status functions, which conditions the environmental consequences of institutional actions.

The ontological approach proposes ways of relating brute facts to abstract concepts related to the application domain, analogous to the status functions proposed by Searle. However, as far as we know, the status functions in these works are used exclusively in the specification of norms. This relationship allows norms to be specified with abstract concepts, making them stable. However, such a relationship does not allow solving the problems that we are pointing out in this thesis. The constitution of status functions can bring normative (covered by these works) and environmental consequences. We are interested in the latter.

In short, current models of artificial institutions allow expressing the institutional consequences of concrete facts but do not allow expressing the consequences of institutional facts in the environment. It limits the use of models only to support regulative norms. However, these works ignore some of the motivations for creating status functions. Searle's theory, presented in Section 2.1, makes it clear that the functions associated with statuses are created to satisfy the practical interests of the agents. Aguilar et al. (RODRIGUEZ-AGUILAR, Juan A. et al., 2015) back up this conclusion by arguing that institutions have not yet addressed how to help agents make decisions to achieve their interests. These limitations and conclusions inspire the purpose model introduced in the next chapter.

2.3 REASONING ABOUT INSTITUTIONAL REALITY

This section describes works that address agents reasoning about the institutional reality to satisfy their interests (i.e., its goals or anti-goals). For this reason, the works that propose ways for agents to reason about normative consequences (activation, violation, etc. of norms) are not considered. There are two reasons to explain it. First, most of the works presented in Section 2.2 consider the agents' side in relation to institutional reasoning for the satisfaction of norms. Second, the behaviour prescribed by the norms may be not related, or even may conflict, with the goals or anti-goals of the agent. Furthermore, the norms do not include explicit means to connect the prescribed behaviour to the goals or anti-goals of the agents and the consequences that the regulated actions can provoke in the environment.

The approach of Criado et al. (CRIADO et al., 2014) proposes ways for the agent to reason about institutional facts to satisfy its interests. This approach adds the reasoning about constitutive rules to the Belief Desire Intention (BDI) reasoning cycle (BRATMAN; ISRAEL; POLLACK, 1988). The reasoning about constitutive rules consists of finding the brute fact that constitutes status functions whose identifier is the same as the identifier of the goal of the agent. For example, if the agent has the goal of *marriage* and there is a constitutive rule stating that *signing a contract counts as marriage* the agent concludes that by *signing the contract* it achieves its goal. From this information, agents then follows the course of actions required to sign the contract, the status function is constituted, and the goal of the agent is satisfied.

2.3.1 Discussion

To discuss the work presented, it is important to remember the problem involved in this thesis, which is the lack of representation of the consequences of institutional actions. The approach by Criado et al. (CRIADO et al., 2014) proposes ways for the agent to reason about constitutive rules only if the agent's goal has the same identifier as the status function in the constitutive rule. For example, if the agent's goal is *marriage* and there is a constitutive rule stating that signing a contract counts as *marriage*, the agent can reason about the rule. However, if the agent's goal is to *form a family*, it cannot reason about the same constitutive rule even if the practical effect of the constitution is the same. In short, this work does not allow agents to reason about the consequences of constitutions considering the states of the world that can be reached (which may or may not be in the interests of agents). Agents' declarative goals (see more in Section 3.1) describe states of the world that the agents want to achieve. Status functions (see more in Section 2.1) describe abstract concepts that are assigned to concrete elements through the constitution process. These statuses are just identifiers that do not seem to express the same idea as the agent's goal. Therefore, relating them does not seem to be appropriate.

In short, current works still do not consider ways for the agent to reason about the

consequences that constitutions produce in the environment. The result of this limitation is that agents may find it difficult to reason about constitutive rules in terms of satisfying their goals or avoiding anti-goals. Allowing the agent to reason about the institution in such a way is an open question that also inspires the model of purpose presented in the next chapter.

3 PURPOSE PROPOSAL FOR MAS

The previous chapter explained how Searle's theory deals with the environmental consequences of institutional actions. These consequences are called Purpose by Searle (SEARLE, J. R., 1995; SEARLE, J., 2010). Resuming the definition, purposes are "consequences that happen in the environment from the constitution of status functions that are directly related to the interests of agents". For example, an agent performs some action that constitutes *payment* because the consequence of that action on the environment matches its goal of *holding a book*.

The concept of purpose inspires the model proposed in this thesis, which considers that status functions are associated with purposes that point to the states that can be achieved from the constitution of these status functions. We assume that these states are always of the interests of the agents (i.e., states they want to achieve or prefer to avoid). The main novelty of this model in relation to the literature in the area of artificial institutions is the fact that it makes explicit the consequences that occur in the environment from the constitution of status functions. These consequences are ignored in current work on artificial institutions, being implicit in the mind of the system designer, limiting the actions of agents in the aspects addressed in Section 1.1. Considering the objectives of this thesis, stated in Section 1.3, this chapter (i) defines suitable abstractions to represent the purposes, (ii) defines how the models of artificial institutions, proposed in the literature, integrate the proposed representation of purposes and (iii) defines how the interests of agents, expressed through goals or anti-goals, are related to the proposed representation of purposes.

In the following, Section 3.1 presents an overview of the model. Section 3.2 presents the formalization of the institutional concepts presented in Section 3.1. The elements presented in Sections 3.1 and 3.2 are used to instrument the institutions with information regarding the consequences of institutional actions. Section 3.3 presents some functions and algorithms that agents can use to act in systems composed of institutions. Section 3.4 presents the use of algorithms in a case study. Sections 3.3 and 3.4 illustrate how agents can use the information from Sections 3.1 and 3.2 in their reasoning and deliberation processes. Section 3.5 presents the implementation of the model in an ontology. Section 3.6 presents the integration of the ontology, which implements the purpose model, in a MAS development framework. Finally, Section 3.7 presents the Jason implementation of some algorithms described in Section 3.3.

3.1 PURPOSE MODEL - OVERVIEW

The essential concepts composing the proposed model are *agents*, *(anti) goals*, *institutions*, and *purposes*, depicted in the Figure 6. *Agents* are autonomous entities that can interact within a dynamic environment composed of non-autonomous elements to achieve their goals (WOOLDRIDGE, 2009). The literature presents several definitions of *goal* that are different but complementary to each other (see more in (WINIKOFF et al., 2002), (HINDRIKS,

K. V. et al., 2000), (RIEMSDIJK; HOEK; MEYER, 2003), and (NIGAM; LEITE, 2006)). In this thesis, *goals* are states of the environment that agents aim to achieve. According to Aydemir et.al (AYDEMIR; GIORGINI; MYLOPOULOS, 2016), *anti-goal* is an undesired situation of the system. In this thesis, *anti-goal* are states that agents aim to avoid for ethical reasons, particular values, prohibitions, etc. *States* are formed by one or more properties that describe the characteristics of the system at some point of its execution (CASSANDRAS; LAFORTUNE, 2008). Moreover, agents can perform actions that trigger events in the MAS. We assume that the process of converting actions to events happens automatically in the system.

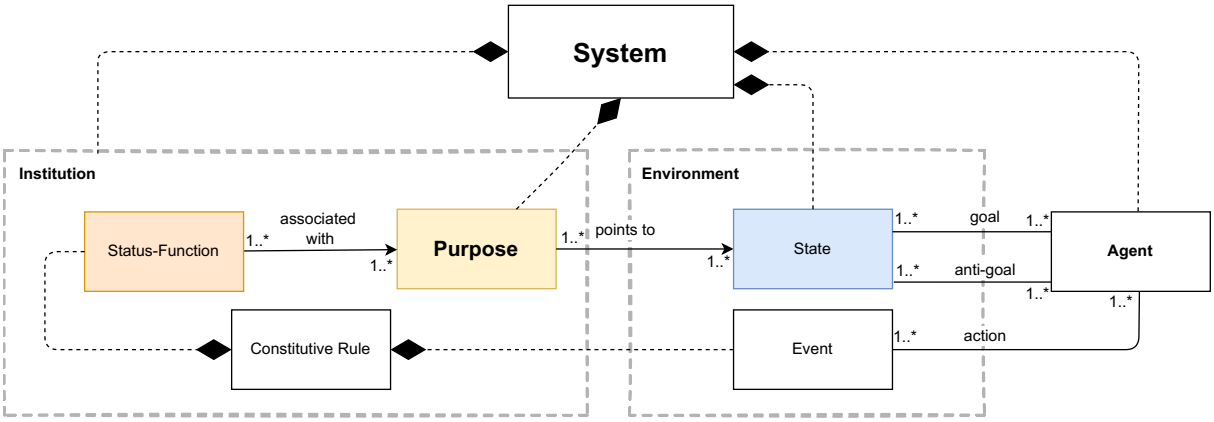

Figure 6 – Overview of the model.

Institutions provide the social interpretation of the environmental elements of the MAS as usually proposed in the literature (CLIFFE; DE VOS; PADGET, 2006; FORNARA, 2011; BRITO, M. d. et al., 2016; CARDOSO, HENRIQUE LOPES; OLIVEIRA, EUGÉNIO, 2007; VIGANÒ; COLOMBETTI, Marco, 2008; ALDEWERELD, Huib et al., 2010). This social interpretation occurs through the interpretation of constitutive rules that assign status to environmental elements, as described in Section 2.1. It is beyond the scope of this thesis to propose a model of artificial institutions. Instead, it considers this general notion of the institution as the entity that constitutes status functions, which is adopted by several models in the field of MAS.

The functions associated with status functions can satisfy the practical interests of agents (SEARLE, J. R., 1995, p.20). From the institution's perspective, these interests are called *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*. Actions performed by agents trigger intermediate actions that reach the states pointed out by the purposes, satisfying the agents' goals or anti-goals. For example, when an agent performs an action that constitutes *tweet*, this makes possible the execution of other intermediate actions (e.g., server receives the message, filters the message if necessary, etc.) that bring the system to states such as *published information* (i.e., the agent goal) or *fake news spread* (i.e., the agent anti-goal). In our proposal, the states pointed to by the purposes are expected to occur

in the system as long as the intermediate actions are successfully executed. However, these actions are outside the institution's control and depend on other elements that make up the system. For this reason, this work only points to the expected states without caring about their achievement. We consider that the agent's interest refers to the expected state independent of intermediate steps.

Shortly, this model provides two relationships: (i) between purposes and status functions and (ii) between purposes and states of the world that match the goals and anti-goals of agents. The consequences of institutional actions, which are the result of the constitution of status functions, become explicit, and from this, agents can satisfy their interests in systems composed of institutions. Thus, if there is (i) a constitutive rule specifying how a status function is constituted, (ii) a purpose associated with that status function, 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. From these relationships, the agent can perform three queries to find out what action it can take to achieve its goal: (i) a query to find the purposes that point to states of the world that match with the goals of the agent, (ii) a query to find out which status function is associated with the found purposes, and (iii) a query to find which concrete actions can constitute that status functions. For example, the agent can find that the purpose of *transmitting information* points to the *published information* state, which matches the goal of the agent; the purpose of *transmitting information* is associated with the *tweet* status function and the action *broadcast a message* can constitute *tweet* in this system. Therefore, if the agent performs the action *broadcast a message*, it achieves its goal in this system.

3.2 PURPOSE MODEL - INSTRUMENTING INSTITUTIONS

This section formally[1] describes the model by specifying the connection of purposes (i) with the status functions, and (ii) with the consequences in the environment of constituted status functions. Although the concept of purpose is independent, it is used in conjunction with the states, agents, and institutions that make up the MAS. To illustrate the use of the model, we use as an example the scenario in which an agent (henceforth referred to as *Bob*) aims to publish information on different social networks.

Definitions 1 to 5 represent the environmental elements and their relationships. Definitions 6 and 7 are based on (BRITO, M. d. et al., 2016; DE BRITO; HÜBNER, Jomi Fred; BOISSIER, 2018) and represent the essential elements that make up institutions (expressed in the Institution rectangle in the Figure 6). The definitions 8 to 11 represent the purposes and its relations with status functions and states of the environment that are of interest of agents (expressed in the Purpose rectangle and its relations in the Figure 6).

[1] We formalize the model to make it more accurate and facilitate the development of algorithms that can be used to improve the agents' decision process.

Definition 1 (States) *A state is a set of properties. Properties are characteristics of the environment at some point of its execution. The set of all properties that the system can present is represented by $\mathcal{T}$.The state of the environment at some moment is the set of all the standing properties. $\mathcal{S}= 2^{\mathcal{T}}$ is the set of all possible states of the environment. For example, if $\mathcal{T} = \{published\_information, fake\_news\_spread\}$, then $\mathcal{S} = \{\{\},\{published\_information\}, \{fake\_news\_spread\}, \{published\_information, fake\_news\_spread\}\}$.*

Definition 2 (Events) *Event is an instantaneous occurrence within the system (CASSANDRAS; LAFORTUNE, 2008). Events may be both triggered by actions of the agents (e.g., sending of a message) and spontaneously produced by some non autonomous element (e.g., a clock tick). The set of all events that may happen in the environment is represented by $\mathcal{E}$. Each event is represented by an identifier. For example, in a MAS where the events that may occur in the environment are* broadcast_a_message *and* send_private_message, *$\mathcal{E} = \{broadcast\_a\_message, send\_private\_message\}$.*

Definition 3 (Agents) *The set of all agents that can act in the MAS is represented by $\mathcal{A}$. Each agent is represented by an identifier. For example, if the agents acting in the system are* Bob *and* Tom, *then $\mathcal{A} = \{Bob, Tom\}$.*

Definition 4 (Relationship between Agents and their goals) *In this work, goals are states of the world that agents desire to reach.[2] $\mathcal{G}$ relates agents and their goals ($\mathcal{G} \subseteq \mathcal{A} \times \mathcal{S}$). For example, the pair $\langle Bob, \{published\_information\}\rangle \in \mathcal{G}$ means that the agent Bob has the goal published_information.*

Definition 5 (Relationship between Agents and their anti-goals) *Anti-goals are states in the MAS that an agent does not desire. $\overline{G}$ relates agents and their anti-goals ($\overline{G} \subseteq \mathcal{A} \times \mathcal{S}$). For example, the pair $\langle Bob, \{fake\_news\_spread\}\rangle \in \overline{G}$ means that Bob has the anti-goal fake_news_spread. From a general point of view, there is no difference between an anti-goal and the denial of a goal. However, to avoid the addition of negated goals in the model, we opted to have explicit anti-goals. The intersection between agent goal and anti-goal should be empty ($\mathcal{G} \cap \overline{G} = \emptyset$).*

Definition 6 (Status Functions) *A* status *is an identifier that assigns to the environmental elements an accepted position, especially in a social group. It allows the environmental elements to perform* functions *(associated with the status) that cannot be explained through its physical structure (SEARLE, J., 2010, p.07). The set of all the status functions of an institution is represented by $\mathcal{F}$. For simplicity, in this formalization we only consider statuses assigned to events. For example, if the existing status functions in a system are* tweet *and* story, *then $\mathcal{F} = \{tweet, story\}$.*

[2] We focus on declarative goals (i.e., goals that describe desirable situations) because declarative goals are state-related goals.

Definition 7 (Constitutive rules) *Constitutive rules specify the constitution of status functions from environmental elements. Searle proposes to express these rules as* X count-as Y in C*, as explained in Section 2.1. Since the process of constitution is beyond the scope of this thesis, the element C can be ignored. For simplicity, a constitutive rule is hereinafter expressed as* X count-as Y*. The set of all constitutive rules of an institution is represented by $\mathcal{C}$. A constitutive rule $c \in \mathcal{C}$ is a tuple $\langle x,y \rangle$, where $x \in \mathcal{E}$ and $y \in \mathcal{F}$, meaning that x count-as y. For example, $\mathcal{C} = \{\langle broadcast\_a\_message, tweet \rangle\}$ defines a MAS with a single constitutive rule. This rule states that* broadcast_a_message *count-as* tweet*.*

Definition 8 (Purposes) *The purposes are related to the agents' practical interests. The set of all purposes is represented by $\mathcal{P}$. Each purpose is represented by an identifier. For example, $\mathcal{P} = \{transmit\_information\}$ defines the unique purpose for the MAS.*

Definition 9 (Relationship between status functions and purposes) *We define that purposes are associated with the constitution of status functions. Thus, there must be a relationship between these two concepts. This relation is represented by $\mathcal{F}_P \subseteq \mathcal{F} \times \mathcal{P}$. For example, $\{\langle tweet, transmit\_information \rangle\} \in \mathcal{F}_P$ means that the constitution of the status function tweet is associated with the purpose transmit_information.*

Definition 10 (Relationship between purposes and agent's goals and anti-goals) *The relationship between purpose and agent goal and anti-goal considers that a purpose point to one or more states in the MAS that matches the agents goals and anti-goals. $\mathcal{G}_P$ is the relation between purpose and goals and anti-goals ($\mathcal{G}_P \subseteq \mathcal{P} \times \mathcal{S}$). For example, $\{\langle transmit\_information, \{published\_information, fake\_news\_spread\} \rangle\} \in \mathcal{G}_P$ means that purpose transmit_information points to a state with properties published_information and fake_news_spread, which may also be goal or anti-goal of some agent.*

Definition 11 (Model) *The overall model of a MAS with purpose is thus a tuple $\langle \mathcal{S}, \mathcal{E}, \mathcal{A}, \mathcal{G}, \overline{\mathcal{G}}, \mathcal{F}, \mathcal{C}, \mathcal{P}, \mathcal{F}_P, \mathcal{G}_P \rangle$, which elements as defined previously.*

3.3 PURPOSE MODEL - HELPING AGENTS TO ACT IN SYSTEMS COMPOSED OF INSTITUTIONS

Considering the problems of this thesis, discussed in Section 1.1, this section defines functions and algorithms that help the agents to explore the connections defined by the Purpose model to satisfy their interests. In the following, Section 3.3.1 introduces some functions that define relationships among the sets formalized in Section 3.2. These functions are important for extracting information useful for agents' reasoning. Section 3.3.2 introduces some algorithms that instrument agents to use this information in their reasoning and deliberation processes. The use of these algorithms and practical implementation are presented in the rest of this thesis.

3.3.1 Functions to retrieve information for the agents

In this section, we formalize some functions that an agent can use to discover (i) which institutional action can be taken to achieve its goal or avoid its anti-goal and (ii) the environmental effects of performing an institutional action. For (i), we need purposes that point to the goals and anti-goals of agents (Definition 12), the status functions that enable the purposes (Definition 13), and the events that can constitute status functions (Definition 14). For (ii), we need the status functions related to the events produced by an action (Definition 15), the purposes of these status functions (Definition 16), and the states pointed by these purposes (Definition 17).

Definition 12 (Mapping agents interests to purposes) *The set of purposes related to some state (and thus related to the agents' interests) is given by the function* $fsp : \mathcal{S} \rightarrow 2^{\mathcal{P}}$ *s.t.* $fsp(g) = \{p \mid \langle p, s\rangle \in \mathcal{G}_P \wedge g \subseteq s\}$. *For example, if* $\mathcal{G}_P = \{\langle transmit\_information, \{published\_information, fake\_news\_spread\}\rangle\}$, *then* $fsp(\{published\_information\}) = \{transmit\_information\}$.

Definition 13 (Mapping purposes to status functions) *The set of status functions associated with a purpose is given by the function* $fpsf : \mathcal{P} \rightarrow 2^{\mathcal{F}}$ *s.t.* $fpsf(p) = \{f \mid \langle f,p\rangle \in \mathcal{F}_P\}$. *For example, if* $\mathcal{F}_P = \{\langle tweet, transmit\_information\rangle\}$, *then* $fpsf(transmit\_information) = \{tweet\}$.

Definition 14 (Mapping status functions to events) *The events that constitute a status function are given by the function* $fca : \mathcal{F} \rightarrow 2^{\mathcal{E}}$ *s.t.* $fca(f) = \{e \mid \langle e,f\rangle \in \mathcal{C}\}$. *For example, if* $\mathcal{C} = \{\langle broadcast\_a\_message, tweet\rangle\}$, *then* $fca(tweet) = \{broadcast\_a\_message\}$.

Definition 15 (Mapping events to status functions) *The status functions that are constituted by an event are given by the function* $fc : \mathcal{E} \rightarrow 2^{\mathcal{F}}$ *s.t.* $fc(e) = \{f \mid \langle e,f\rangle \in \mathcal{C}\}$. *For example, if* $\mathcal{C} = \{\langle broadcast\_a\_message, tweet\rangle\}$, *then* $fc(broadcast\_a\_message) = \{tweet\}$.

Definition 16 (Mapping status functions to purposes) *The set of purposes that are associated with a status function is given by the function* $fp : \mathcal{F} \rightarrow 2^{\mathcal{P}}$ *s.t.* $fp(f) = \{p \mid \langle f,p\rangle \in \mathcal{F}_P\}$. *For example, if* $\mathcal{F}_P = \{\langle tweet, transmit\_information\rangle\}$, *then* $fp(tweet) = \{transmit\_information\}$.

Definition 17 (Mapping purposes to states) *The set of states pointed by a purpose is given by the function* $fsw : \mathcal{P} \rightarrow \mathcal{S}$ *s.t.* $fsw(p) = \{s \mid \langle p, s\rangle \in \mathcal{G}_P\}$. *For example, if* $\mathcal{G}_P = \{\langle transmit\_information, \{published\_information, fake\_news\_spread\}\rangle\}$, *then* $fsw(transmit\_information) = \{\{published\_information, fake\_news\_spread\}\}$.

3.3.2 Algorithms to help the agent reason about purpose

This section presents algorithms that use the functions presented in Section 3.3.1. These algorithms can be used by the agents to reason and deliberate about actions that can be used to achieve their goals or avoid anti-goals in institutional contexts.

Algorithm 1 can be used by the agent to find which institutional action can achieve its goal. The algorithm can be summarized in the following steps: (1) verify if the agent's goal (i.e., the state) is pointed by some purpose (line 4); if true, go to the next step, otherwise return an empty set (line 14); (2) consider all purposes that are pointed to by the agent's goal (line 5); (3) for each status function that may be associated with such purposes (line 6), find the events that may constitute such status function (line 7), the actions that can produce such event (line 8), and add the actions in a set (line 9); and finally, (4), return the set of actions (line 14).

Algorithm 1 Find institutional actions to achieve a goal g

```
1:  Input: agent goal g
2:  Output: The set of actions ac
3:  ac ← {}
4:  if fsp(g) ≠ {} then                    ▷ If the goal g is a state appointed by the purposes
5:      for p ∈ fsp(g) do                  ▷ p is the set of purposes that point to the agent goal g
6:          for f ∈ fpsf(p) do             ▷ f is the set of status functions associated with the purpose set p
7:              for e ∈ fca(f) do          ▷ e is one event that constitute f
8:                  acpe ← action that can produce the event e
9:                  ac ← ac ∪ {acpe}       ▷ add action that can produce the event e
10:             end for
11:         end for
12:     end for
13: end if
14: return ac
```

The Algorithm 2 can be used by the agent to find out which are the environmental effects of executing an action in an institutional context. The algorithm can be summarized in the following steps: (1) verify whether the action is an institutional action, i.e., if it produces events that constitute status functions in the institution (lines 4 and 5); if true, go to the next step, otherwise return an empty set (line 12); (2) consider all status functions related to the action (line 6); (3) consider all purposes of such status functions (line 7); and (4) for each purpose, add the states it points to the algorithm return set (line 8).

The Algorithm 3 can be used by the agent to verify whether some action in an institutional context can produce states considered agents' goals. The algorithm can be summarized in the following steps: (1) use algorithm 2 to find the states that are reached if action *ac* is performed; (2) check whether these states match some goal in $\mathcal{G}$.

Algorithm 2 Find the effects of an action ac in the environment

Input: an action ac
Output: the set of possible states after ac
s $\leftarrow \{\}$
e $\leftarrow$ event produced by action ac
if $fc(e) \neq \{\}$ **then** ▷ if the event e may constitute a status functions
  for f $\in fc(e)$ **do** ▷ f is the set of status functions that e count-as
    for p $\in fp(f)$ **do** ▷ p is the set of purposes that are associated with f
      s $\leftarrow$ s $\cup$ $fsw(p)$ ▷ add states pointed to by p
    end for
  end for
end if
return s

Algorithm 3 Verify whether some action ac can produce states considered as goals for agent ag.

Input: ac, ag
Output: returns true if ac implies an agent goal and false otherwise
se $\leftarrow$ algorithm 2(ac) ▷ se is the set of states pointed to by ac
return $\exists_{g \in \mathcal{S}}$ $\langle ag,g \rangle \in \mathcal{G} \wedge g \subseteq se$ ▷ checks whether goals are included in se

The Algorithm 4 can be used by the agent to verify whether some action in an institutional context can produce states considered agents' anti-goals. The algorithm can be summarized in the following steps: (1) use algorithm 2 to find the states that are reached if action *ac* is performed; (2) check whether the achieved states match some anti-goal in $\overline{G}$.

Algorithm 4 Verify whether some action ac can produce states considered as anti-goals for agent ag.

Input: ac, ag
Output: returns true if ac implies an agent anti-goal and false otherwise
se $\leftarrow$ algorithm 2(ac) ▷ se is the set of states pointed to by ac
return $\exists_{g \in \mathcal{S}}$ $\langle ag,g \rangle \in \overline{G} \wedge g \subseteq se$ ▷ checks whether anti goals are included in se

3.4 EXAMPLE OF USING PURPOSES

Returning to the example of the agent *Bob*, which has the goal *published information* on some social networks, this section illustrates the use of the algorithms formalized in Section 3.3 to achieve the goal on the social network *Instagram*. Furthermore, *Bob* prefers to avoid (i.e., it has the anti-goal) of *fake news spread*. Table 1 contains the sets that define this system.

From the formal representation of the elements that compose the system (Table 1), Algorithm 1 is used by *Bob* to find out an institutional action that can help it to achieve its goal. Table 2 shows the values of the variables when running Algorithm 1. In short, the

Table 1 – Formalization of the example

Element	**Set**
Agents in the MAS	$\mathcal{A} = \{Bob\}$
Possible environmental states	$\mathcal{S} = \{\{\}, \{published\_information\}, \{fake\_news\_spread\}, \{published\_information, fake\_news\_spread\}\}$
Events	$\mathcal{E} = \{broadcast\_a\_message\}$
Goals	$\mathcal{G} = \{\langle Bob, \{published\_information\}\rangle\}$
Anti-goals	$\overline{G} = \{\langle Bob, \{fake\_news\_spread\}\rangle\}$
Status functions	$\mathcal{F} = \{story\}$
Constitutive rules	$\mathcal{C} = \{\langle broadcast\_a\_message, story\rangle\}$
Purposes	$\mathcal{P} = \{transmit\_information\}$
Relation status function × purpose	$\mathcal{F}_P = \{\langle story, transmit\_information\rangle\}$
Relation purpose × state	$\mathcal{G}_P = \{\langle transmit\_information, \{published\_information, fake\_news\_spread\}\rangle\}$

algorithm takes as input Bob's goal (i.e., *published_information*) and returns the action that *Bob* can perform to achieve its goal (i.e., *broadcast_a_message*).

Table 2 – Execution of Algorithm 1 to help *Bob* achieve its goal of published information.

Line	**Situation**
1	Input: g = {*published_information*}
3	ac = {}
4	*fsp*({*published_information*}) = {*transmit_information*}
5	p = {*transmit_information*}
6	*fpsf*(*transmit_information*) = {*story*} then first f = *story*
7	*fca*(*story*) = {*broadcast_a_message*} then first e = *broadcast_a_message*
8	acpe = *broadcast_a_message*
9	ac = {*broadcast_a_message*}
12	Output: ac = {*broadcast_a_message*}

To illustrate the use of the other algorithms, consider that *Bob* knows that it needs to perform the action *broadcast_a_message* and wants to know what effects the action has on the environment. For this, *Bob* can run Algorithm 2. Table 3 shows the values of the variables when running this algorithm. In short, the algorithm takes as input the action that *Bob* intends to perform (i.e., *broadcast_a_message*) and returns the states that are produced by this action (i.e., *published_information* and *fake_news_spread*).

The agent *Bob* uses Algorithm 3 to find out whether the effects of an institutional action are compatible with its goal. For that, the parameters for the algorithm are the action *broadcast_a_message* and the agent *Bob*. In this case, the Algorithm 3 returns *true*, since Algorithm 2 returns {*published_information*, *fake_news_spread*} for the action *broadcast_a_message* and it contains *Bob*'s goal {*published_information*}.

Finally, *Bob* uses Algorithm 4 to find out if some effect of an institutional action are compatible with its anti-goal. Again, the parameters are the action *broadcast_a_message* and the agent *Bob*. The Algorithm 4 returns *true*, since Algorithm 2 returns {*published_information*, *fake_news_spread*} for the action *broadcast_a_message* and it contains *Bob*'s anti-goal {*fake_news_spread*}.

Based on the information returned by these algorithms, *Bob* can make a better decision about whether or not to act on the system. In the example, no action can be used to achieve its goal, which does not imply an anti-goal. Similar steps can be applied to achieve goals and avoid anti-goals in other social networks.

3.5 PURPOSE IMPLEMENTATION

This section illustrates how the purpose model can be implemented. We propose that the model can be implemented through a data structure that allows (i) specifying the status functions, purposes, and states of the world, (ii) associating the status functions with the corresponding purposes, and (iii) associating the purposes with the states of the world to which they point to. Any data structure that records this information and connections can be used.

We use ontology (FIKES; FARQUHAR, 1999; MAEDCHE; STAAB, 2001; GUARINO, 1998) to implement the purpose model. Figure 7 illustrates the model concepts, their connections, and how they are translated into the ontology. The rectangles represent concepts that the model uses. These concepts are converted into classes (circles) within the ontology. The status function concept is converted to the status function class, the purpose concept to the purpose class, etc. The exception is the state concept. It is converted to a corresponding class, but in order to comply with the definition of states (cf. Section 3.2) and with the representation of this concept in the framework used to develop the examples of this work (cf. Section 4.1), it was necessary to add two more classes called *Predicate* and *Parameter*. These classes make it possible to describe predicates that can have one or more parameters (i.e., *terms* in predicate

Table 3 – Execution of Algorithm 2 to help *Bob* discover what are the effects of an institutional action.

Line	**Situation**
1	Input: ac = *broadcast_a_message*
3	s = {}
4	e = *broadcast_a_message*
5	*fc*(*broadcast_a_message*) = {*story*}
6	f = *story*
7	*fp*(*story*) = {*transmit_information*} then first p = *transmit_information*
8	*fsw*(*transmit_information*) = {*published_information*, *fake_news_spread*} then s = {*published_information*, *fake_news_spread*}
12	Output: s = {*published_information*, *fake_news_spread*}

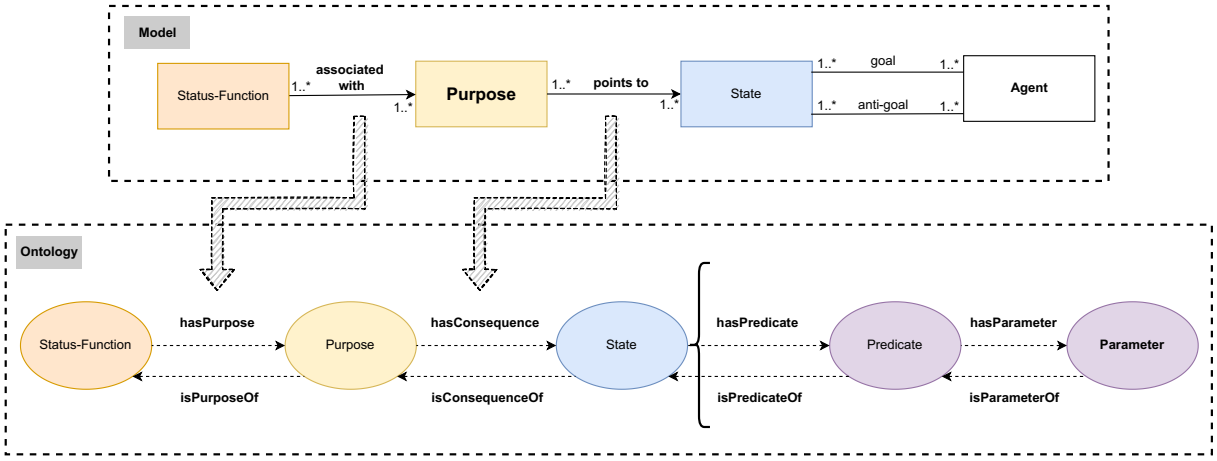

Figure 7 – Model implementation in an ontology.

logic) and relate them to one or more states. The order in which the parameters are associated with the predicate is defined through an annotation called `owl:position`.

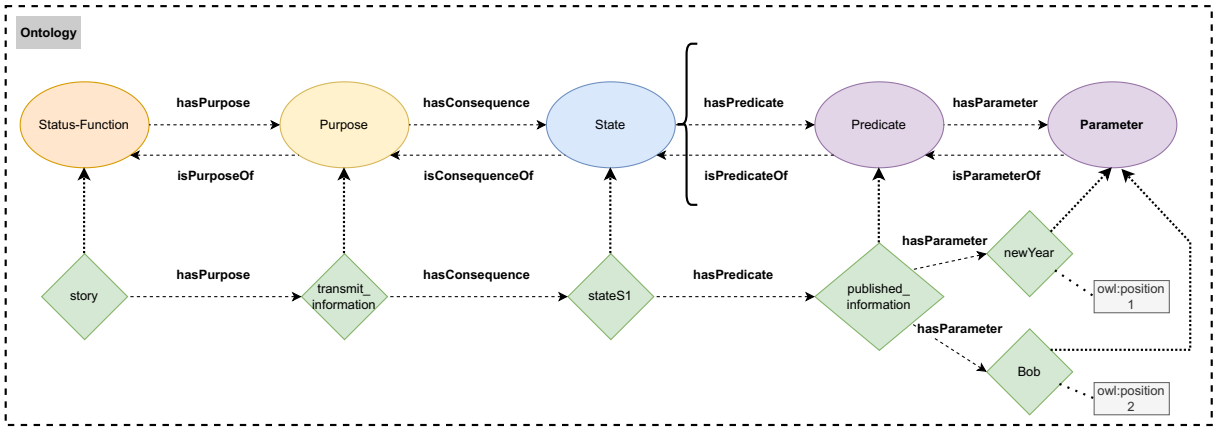

Figure 8 – Example of using the ontology that implements the purpose model.

Figure 8 illustrates some individuals exemplifying the use of the ontology to implement the purpose model. Consider the following statement: The *story* status function is associated with the *transmit_information* purpose. This one points to the *stateS1* state. Such a state is formed by the *published_information* predicate, which is composed of the parameters *newYear* and *Bob*. Converting this information to individuals in the ontology in question, `story` is an individual of the *status function* class, `transmit_information` is an individual of the *purpose* class, `stateS1` is an individual of the *State* class, `published_information` is an individual of the *Predicate* class, and `newYear` and `Bob` are individuals of the *Parameter* class. When adding `newYear` and `Bob`, we can add the values 1 and 2, respectively, to the *owl:position* property of each individual. Obviously, it is necessary to make the associations between the individuals of the ontology so that the information matches the statement. In this case, `story` is related to `transmit_information` through the *hasPurpose* object property, `transmit_information` is related to `stateS1` through the *hasConsequence* object property, `stateS1` is related to the `published_information` predicate through the *hasPredicate* object property, and finally,

`newYear` and `Bob` are related to `published_information` through of the *hasParameter* object property.

3.6 COUPLING THE PURPOSE MODEL IN AN MAS DEVELOPMENT FRAMEWORK

This section illustrates how the purpose model can be introduced into a MAS development framework. For it, we use a series of components depicted in Figure 9. The model can be used in other frameworks as long as it (i) contains means of specifying information about purposes and their connections (with states of the world and status functions) and (ii) allows agents to access this information.

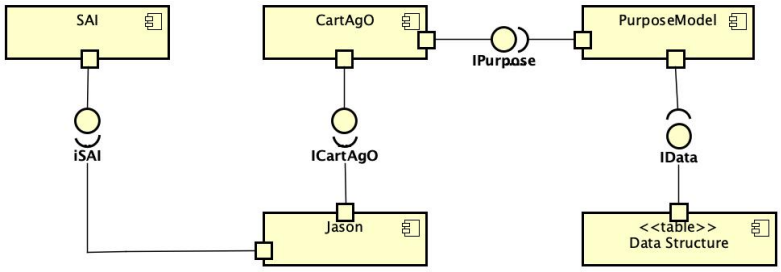

Figure 9 – Component diagram with the systems used to develop the examples.

The agents are programmed in Jason (BORDINI; HÜBNER, Jomi Fred; WOOLDRIDGE, 2007). Jason enables the implementation of BDI agents using high-level constructs, such as beliefs, desires, and intentions, from which the agent makes decisions about the course of action to follow.

The environment is programmed in CArtAgO (RICCI; PIUNTI; VIROLI, 2011), which provides a set of programming abstractions to represent the shared environment. These programming constructs concern first the notion of artifact, the basic entity of the environment that encapsulates computation or other forms of resources (BOISSIER et al., 2019). Agents can act upon the artifacts through their available operations and can inspect states through their observable properties. In this thesis, CArtAgO artifacts provide an interface for agents to interact with the purpose model implementation and SAI.

To introduce artificial institutions, we use an implementation of the SAI model (DE BRITO; HÜBNER, Jomi Fred; BOISSIER, 2018). It provides a model of institutional reality that is represented through status functions, constitutive rules, etc. Constitutive rules assign status functions to elements present in the environment through a process called constitution. SAI allows specifying the elements present in the institutional reality and managing the constitution process.

We use an ontology to implement the model in the examples used in this thesis (cf. Section 3.5). The query and persistence of data in the ontology are enabled by extending the CArtAgO artifact with MasOntology.[3] It is a set of functions developed in CArtAgO that

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

allows agents to interact with ontologies.

3.7 IMPLEMENTATIONS OF THE ALGORITHMS IN JASON PROGRAMS

Listing 3.1 illustrates the program in Jason to implement Algorithm 1. This program is essentially a plan that agents can execute to discover which institutional actions (`Actions` argument) can be performed to achieve a goal (`Goal` argument). We assume each institution has a CArtAgO artifact that allows the agent to access information related to an institutional specification. The plan has the following steps: (i) Create an empty queue to store the actions that may be executed in the environment (line 2); (ii) search for the purposes that point to the goal that the agent wants to achieve (line 3); (iii) discover the status functions that are associated with these purposes (line 5); (iv) find out which actions can constitute the status functions (line 8) and; (v) add these actions to the queue (line 9). The `getPurposeOfState` and `getStatusFunctionsFromPurpose` are operations provided by the artifact that gives access to the purposes, and `constitutive_rule` is a belief acquired through observing the observable properties of the artifact that implements the institution.

```
1  +!alg1(Goal, Actions) <-
2      .queue.create(Actions);
3      getPurposesOfState(Goal, Purposes);
4      for(.member(Purpose,Purposes)) {
5        getStatusFunctionsFromPurpose(Purpose, NameSF);
6        for(.member(StatusFunction, NameSF)){
7          ?constitutive_rule(Action,StatusFunction,_,_);
8          .queue.add(Actions,Action);
9        }
10     }.
```

Listing 3.1 – Program to find the institutional action that can help the agent to achieve its goal.

Listing 3.2 illustrates the program in Jason to implement Algorithm 2. The *alg2* discovers what are the consequences in the environment (`States` argument) from the execution of institutional actions (`Action` argument). The plan has the following steps: (i) Find the status functions that are constituted by the action informed as argument (line 1); (ii) create a queue to store the states that can be reached when the institutional action is performed (line 2); (iii) fetch the purposes that are associated with these status functions (line 3); (iv) fetch the predicates that represent the states pointed to by the purposes (line 5) and (v) add these predicates in the queue of states (line 7). The `getPurposesOfStatusFunctions` and `getPredicatesOfStatesRelatedToPurpose` are operations provided by the artifact that gives access to the purposes.

```
1  +!alg2(Action, States) : constitutive_rule(Action,SF,_,_) <-
```

```
2     .queue.create(States);
3     getPurposesOfStatusFunctions(SF, Purposes);
4     for(.member(Purpose,Purposes)){
5        getPredicatesOfStatesRelatedToPurpose(Purpose, Predicates);
6        for(.member(Predicate,Predicates)){
7              .queue.add(States, Predicate);
8        }
9    }.
10
11 +!alg2(Action, []). // If there is no status functions that can
       be assigned to the action
```

Listing 3.2 – Program to find the consequences in the system of performing an action.

4 EVALUATING THE PURPOSE MODEL

Considering the problems that motivate this work, we deal with the hypothesis that Searle's theory is an inspiration to make explicit the consequences in the environment of the execution of institutional actions. This Chapter presents some examples of applications that help us to evaluate if and how the purpose model - which is inspired by Searle's theory to approach the considered problems - confirms the proposed hypothesis. More precisely, we aim to illustrate that the proposed approach allows (i) to improve agents in aspects of adaptability, rationality, and flexibility and (ii) contribute to the literature in the area of artificial institutions by making these concepts and relationships explicit. This Chapter presents solutions for the following problem explained in Section 1.1: how agents can exploit them to act according to their interests. The other problems, explained in Section 1.1, are resolved and discussed in Chapter 3.

The rest of this Chapter is organized as follows: Section 4.1 presents some practical examples and criteria that are used to evaluate the advantages and disadvantages that agents gain from using the proposed model. Section 4.2 positions the purpose model in the literature, discussing the already existing proposals, the advances that the model provides, and some limitations that still need to be addressed.

4.1 USING THE PURPOSE MODEL FROM THE PERSPECTIVE OF AGENTS

This section illustrates some practical advantages and disadvantages that agents gain from using the purpose model in Open MAS. For that, we define some criteria and use examples to conduct the evaluation. The criteria are:

- **Adaptability**: refers to the capability of the agents to adapt to different social abstractions present in the systems in which they are entering.
- **Rationality**: considers situations where the agents have more information to reason and deliberate about their interests.
- **Flexibility**: refers to the situation where the agents can satisfy their interests in scenarios that are unknown to them at design time without the need to be modified.

We evaluate these criteria through three connections: (a) between actions and their institutional consequences, (b) between the institutional consequences and their environmental consequences, and (c) between the environmental consequences and the agents' goals. In addition, we also use the *implicit* and *explicit* notions. Here, we limit the institutional consequences to those related to the constitutions of status functions assigned to actions. Regarding implicit and explicit notions, they are inspired in (DIGNUM; ALDEWERELD; DIGNUM, F., 2011; LEMAÎTRE; EXCELENTE, 1998). We consider something to be explicit when it is represented as a first-class abstraction. The notion of implicit is defined as the inverse of explicit.

Furthermore, in these examples, for readability, the purposes are described using predicates, although they are implemented in ontologies described in OWL.

In the following, Section 4.1.1 presents an example of two agents in a book trading scenario. The objective of this example is to demonstrate the improvement in the *adaptability* of agents in applications that use social abstractions (in particular, the purpose model); Section 4.1.2 presents an example of an agent acting in a war scenario. The objective of this example is to show how agents' *rationality* improves in applications that use the purpose model; and Section 4.1.3 presents an example of an agent that wants to satisfy its goal in different social networks. The objective of this example is to show how the agent's *flexibility* improves when the purpose model is implemented in the application.

4.1.1 Application Example 1: Book trade

Adaptability is an important feature for agents when they plan to enter and participate in different Open MAS. This section evaluates the adaptability provided by the purpose combined with other social abstractions. This evaluation is based on an example considering a scenario where the agent *Bob* has the goal of *holding a book*. To satisfy its goal, *Bob* can buy a book owned by the agent *Tom*. To make the purchase, *Bob* needs to execute an action that count-as *payment* and waits for the agent *Tom* to acknowledge the action as a *payment* before handing to *Bob* the book. The environment in which agents act has some actions available that, when executed, cause perceptions for them.

4.1.1.1 Case 1 - Simple MAS

This section considers an application without status functions, norms, and purposes. Algorithm 5 describes the steps that an agent needs to perform to satisfy its `holdBook` goal. The Algorithm receives as input the book owner identifier and, from this input, executes the deliver paper note action and waits for the book to be delivered. Listing 4.1 describes the Jason program that implements Algorithm 5. In this program, the agent is implemented with the belief `bookOwner(tom)` where `tom` is the name of the book owner. When executing this program, three steps are executed. First, *Bob* identifies tom as the book owner by consulting its belief base. Second, *Bob* performs the action `deliver_paper_note` informing `tom` as parameter. Third, *Bob* waits until it has the *deliverBook* belief, which is acquired when it realizes the book has been delivered.

Algorithm 5 Algorithm for agents to achieve holdBook goal

1: **Input:** *bookOwner*
2: **do** (deliver_paper_note, *bookOwner*)
3: **wait** (deliverBook)

```
1 bookOwner(tom).  // agent belief
```

```
2
3 !holdBook.  // agent goal
4
5 +!holdBook : bookOwner(Seller)
6     <- deliver_paper_note(Seller);
7        .wait(deliverBook).
```

Listing 4.1 – Program for agents to achieve holdBook goal

The book owner is a reactive agent. In this case, its actions are triggered by reactions to facts perceived in the environment. Algorithm 6 describes the steps the agent needs to take to deliver the book. The algorithm starts when the agent reacts to *deliver_paper_note* action performed by the buying agent and, from this input, executes the action of delivering the book. Listing 4.2 describes the Jason program that implements Algorithm 6. The program starts when *Tom* perceives `deliver_paper_note(bob)`. When executing this program, the following step is executed: *Tom* performs the action `deliver_book` informing `bob` as parameter.

Algorithm 6 Algorithm for book owner agents to deliver the book to buyer agents

1: **when** (deliver_paper_note, buyer)
2: **do** (deliver_book, buyer)

```
1 +deliver_paper_note(Buyer) <- deliver_book(Buyer).
```

Listing 4.2 – Program for book owner agents to deliver the book to buyer agents

In this application, *Bob* is programmed to perform a specific action (i.e., *deliver_paper_note* action) that modifies the environment. *Tom* is programmed to react to this action by delivering the book. In this application, both agents do not depend on (or even consider) any social abstraction to achieve their goals and perform their actions. For this reason, in this example, there is no adaptability and the connections used to evaluate it are not considered. The problem with this approach is that (a) if the actions or the perceptions change, or (b) if environmental consequences change, or (c) if agents' goals change, in all cases, the agents need to be reprogrammed to adapt to changes.

4.1.1.2 Case 2 - MAS with status functions

This section considers an application with status functions and without norms and purposes. We assume that the system has an institutional specification with a constitutive rule stating that *deliver_paper_note count-as payment*. The constitution of the status function *payment* produces the effects in the environment that satisfy Bob's goal.

Algorithm 7 describes the steps that an agent needs to do to satisfy its `holdBook` goal. The Algorithm receives as input the identifier of the book owner and the status function that should be constituted. From that input, executes any action that constitutes the status function

and waits for the book to be delivered. Listing 4.3 describes the Jason program that implements Algorithm 7. In this Listing, the agent is implemented with beliefs `bookOwner(tom)` where `tom` is the name of the book owner and `statusFunction(payment(tom))` where `payment(tom)` is the status function that should be constituted. When executing this program, four steps are executed. First, *Bob* identifies tom as the book owner and payment as the status function that should be constituted by consulting its belief base. Second, *Bob* queries the constitutive rules of the system to verify which action constitutes `payment`. `deliver_paper_note` is the action. Third, *Bob* performs `deliver_paper_note`. Fourth, *Bob* waits until it has the *deliverBook* belief, which is acquired when it realizes the book has been delivered.

Algorithm 7 Algorithm for agents to achieve *holdBook* goal with status function

1: **Input:** *bookOwner* **and** *statusFunction*
2: *action* ← **look for** which *action* constitutes *statusFunction*
3: **do** (*action*, *bookOwner*)
4: **wait** (deliverBook)

```
1 bookOwner(tom). // agent belief
2 statusFunction(payment(tom)). // agent belief
3
4 !holdBook. // agent goal
5
6 +!holdBook : statusFunction(SF)  & bookOwner(Seller)
7    <- ?constitutive_rule(Action,SF,_,_);
8       Action;
9       .wait(deliverBook).
```

Listing 4.3 – Program for agents to achieve *holdBook* goal with status functions

Algorithm 8 describes the steps that the reactive agent needs to take to deliver the book. The algorithm starts when the agent perceives the constitution of a `payment`. From this input, it reacts by executing the action of delivering the book. Listing 4.4 describes the Jason program that implements Algorithm 8. The program starts when *Tom* perceives `payment(bob)` belief. *Tom* then performs the action `deliverBook` informing `bob` as parameter.

Algorithm 8 Algorithm for book owner to deliver the book to buyer with status functions

1: **when** (payment is constituted, buyer)
2: **do** (delivery_book, buyer)

```
1 +payment(Buyer) <- deliver_book(Buyer).
```

Listing 4.4 – Program for book owner to deliver the book to buyer with status functions

In this application, *Bob* is programmed to perform some action that counts as *payment*. Such a constitution has the environmental consequence of *holdBook* that matches Bob's goal.

On the other hand, *Tom* is programmed to react to the *payment* constitution and deliver the book to *Bob*. There are some improvements and problems with this approach discussed below.

The institutional consequence of actions is explicitly captured by the status function concept and is represented through the institutional specification. The connection between *deliver_paper_note* and *payment* is implemented by a constitutive rule. In this example, *Bob* has been coded to exploit this connection and perform any action that counts as *payment*. *Bob* can adapt in this system even if the set of actions that count as *payment* changes. Likewise, the reactive agent is programmed to react to the constitution of the status function. In this example, *Tom* is specified to react to any action that count as *payment*. In the same way, if the actions change but still constitute *payment*, *Tom* can also adapt to changes.

However, this system lacks an appropriate concept to capture the environmental consequences that result from the institutional consequences. Therefore, the connection between *payment* and *holdBook* is implicit. In this example, *Bob* has been coded to perform any action that counts as *payment* because is assumed that *payment* has the environmental consequence of *holdBook*. However, if the environmental consequence of this status function changes, the agent may not adapt to this change. For example, *Bob* would not be able to adapt to institutions where status functions other than *payment* produce the same effect *holdBook*. Even if there is a connection between actions and their institutional consequences, there is not a connection between institutional consequences and the environmental consequences that result from them.

The lack of a proper abstraction to represent the connection between the institutional consequences and their environmental consequences naturally leads to the impossibility of relating these environmental consequences to the interests of the agents. The connection between the environmental consequence *holdBook* and the agent's goal *holdBook* is implicit. In this example, *Bob* has been coded to perform any action that counts as *payment* because it is assumed, in design time, that this constitution has consequences in the environment that match Bob's goal. However, if the environmental consequence or agent's goal change, the agent may not adapt. For example, if Bob's goal changes, it would not be able to adapt to institutions even if they provided the means to achieve its new goal.

In short, in this case, the connection between the action and its institutional consequences became explicit. Agents can adapt to changes that occur in the set of actions available as long as these actions remain having the same institutional consequences. However, the agents' code is tightly coupled to the status functions. For example, if *payment* is replaced by another status function, agents may not be able to adapt. Furthermore, the other two connections are implicit because neither the environmental consequences nor the agents' goals that these environmental consequences serve have abstractions to represent them in the system. Therefore, (a) if the institutional consequences change, or (b) if the environmental consequences change or (c) if agents' goals change, in all cases, the agents may not be able to adapt and therefore need to be reprogrammed.

4.1.1.3 Case 3 - MAS with status functions and norms

This section considers an application with status functions and norms. We assume that the system has (i) an institutional specification that contains a constitutive rule stating that *deliver_paper_note count-as payment* and (ii) a norm stating that *Tom is obliged to deliver the book to the agent that constitutes* payment. This constitutive rule, when constituted, produces the effects in the environment that satisfy Bob's goal. This norm, when activated, obliges *Tom* to deliver the book to whoever constitutes `payment`.

Bob uses the same Algorithm and performs the same steps as described in Section 4.1.1.2. Algorithm 9 describes the steps that the reactive agent needs to take to deliver the book. The algorithm starts when the agent perceives the obligation established by the norm. From this input, it reacts to the norm by performing the action of delivering the book. Listing 4.5 describes the Jason program that implements Algorithm 9. The program starts when *Tom* perceives that a norm with its name is activated. *Tom* then performs the action `deliver_book` informing `bob` as parameter.

Algorithm 9 Algorithm for book owner to deliver the book to buyer with a norm

1: **when** obligation norm is **activated by** *buyer*
2: **do** (delivery_book, *buyer*)

```
1 +obligation(Ag,R,Goal,Deadline) : .my_name(Ag) <- Goal.
```

Listing 4.5 – Program for book owner to deliver the book to buyer with a norm

The benefits and drawbacks of the connections seen in this application example are the same as those discussed in Section 4.1.1.2. The difference, however, is in the reactive agent. Norms are a usual way of expressing the expected behavior of agents in society (BOELLA, Guido; VAN DER TORRE; VERHAGEN, 2008). The reactive agent is decoupled from the status functions when the norms are specified explicitly in the system. In this example, *Tom* is programmed to respond to a norm. This norm is activated whenever a `payment` is constituted. Even if other constitutions activate the norm, *Tom* can act accordingly in this system. Therefore, the reactive agent adapts to changes that occur in the system as long as they continue to activate the norm that involves it. In other words, the reactive agent remains institutionalized while decoupling from the status functions.

4.1.1.4 Case 4 - MAS with status function, norms and purpose

This section considers an application with status functions, norms, and purposes. We assume that the system has (i) a constitutive rule stating that *deliver_paper_note count-as payment*, (ii) a norm stating that *Tom is obliged to deliver the book to the agent that constitutes* payment, and (iii) a purpose that points to the state of the world `HoldBook` that is related to `payment` status function. In this scenario, *Bob* can constitute any status function

whose constitution produces effects in the environment that match its *holdBook* goal and activate the norm that obliges *Tom* to deliver the book to *Bob*.

Algorithm 10 describes the steps that an agent may do to satisfy its `holdBook` goal. The Algorithm takes the agent's goal as input and uses `alg1` to find a list of actions that might achieve the goal, executing the first element of the list. Listing 4.6 describes the Jason program that implements Algorithm 10. In this Listing, the agent is implemented with `holdBook` goal. When executing this program, three steps are executed. First, *Bob* calls `alg1` informing its goal as parameter. `alg1` (cf. Listing 3.1) returns to *Bob* a list with actions. Second, *Bob* gets the first action on this list. *deliver_a_paper_note* is the action found. Third, *Bob* executes this action. *Tom* uses the same Algorithm and performs the same steps as described in Section 4.1.1.3.

Algorithm 10 Algorithm for agents to achieve their goals with status functions and purposes

1: **Input:** *agentGoal*
2: *actionList* ← **call** `alg1` (*agentGoal*)
3: *action* ← first(*actionList*)
4: **do** (*action*)

```
1 !holdBook.  // agent goal
2
3 +!holdBook <-
4     !alg1(holdBook, Actions);
5     .queue.head(Actions,Action);
6     Action.
```

Listing 4.6 – Program for agents to achieve their goals with institution and purposes

In this application, *Bob* is programmed to look for (i) a purpose that points to its *holdBook* goal, (ii) a status function related to that purpose, in the case of this example, *payment*, and (iii) a concrete action that can constitute this status function, in the case of this example, *deliver_paper_note* action. This approach has some advantages, which are further discussed below.

The institutional consequences are captured and represented as described in Sections 4.1.1.2 and 4.1.1.3. The environmental consequence of the constitution of status function is now explicitly captured by the purpose concept and is represented through the institutional specification. The connection between *payment* and *holdBook* is implemented by an ontology. In this example, *Bob* has been coded to use `alg1`, which exploits this connection by looking for the environmental consequences that result from the institutional consequence of *payment*. *Bob* can adapt to this system even as the environmental consequences change as long as they remain matching its interest.

The explicitness of the connection between institutional consequences and environmental consequences enables the connection between environmental consequences and the agents' goals. The connection between *holdBook* that represents the environmental consequence of

payment and *holdBook* that represents bob's goal is explicit in the system. In this example, *Bob* is implemented to achieve *holdBook* goal. It use the steps already described to find the environmental consequences of institutional actions and then can relate them to its goal. Even if the environmental consequences or the agents' goals change, *Bob* can consult the new consequences and if these consequences match its goals before it constitutes any status function.

In short, all three connections are explicitly captured by appropriate concepts and properly represented. Agents can query at runtime which purposes point to states of interest, which status functions are related to these purposes, and finally, which concrete actions can constitute these status functions. Therefore, (a) if the actions or their institutional consequences change, (b) if their environmental consequences change, or (c) if the agents' goals change, agents can adapt to the scenario as long as there are institutional actions that have environmental consequences that match the agents' goals.

4.1.1.5 Discussion

This section discusses the adaptability that agents acquire in open MAS when some social abstractions become explicit. Table 4 summarizes the agents' awareness of social abstractions and the need to adapt agents to connections that use such abstractions in each case. We consider an agent aware of a social abstraction when it is explicitly represented in the system and the agent can adapt to it. The columns that make up the table are described below:

- Column *SF* means awareness of status functions.
- Column *N* means awareness of norms.
- Column *P* means awareness of purposes.
- Columns *SF*, *N* and *P* have the symbol ✓ when the agent is aware of the social abstraction to which the column refers to and uses such abstraction in its reasoning and deliberation process. The absence of this symbol in the column means the opposite.
- Column *CX1* means the connection between the action and its institutional consequence is captured by a social abstraction in the system and appropriately represented.
- Column *CX2* means the connection between the institutional consequences and its environmental consequences is captured by a social abstraction in the system and appropriately represented.
- Column *CX3* means the connection between the environmental consequences and the agents' goals is captured by a social abstraction in the system and appropriately represented.

- Columns *CX1*, *CX2* and *CX3* have no symbol if the agent needs to be reprogrammed because it does not adapt to the changes that may occur in the connection to which the column refers. Columns have the symbol ○ if the agent can partially adapt to the changes that may occur in the connection to which the column refers. For example, consider that the agent is programmed to constitute *payment*, as in case 2. It can adapt to this scenario if the system changes the set of actions that count as payment. However, if for some reason *payment* is replaced by another status function, the agent cannot adapt. Finally, the columns receive the symbol ● if the agent adapts itself to any changes that may occur in the connection to which the column refers. Consider again the example of the agent that needs to constitute *payment*. If *payment* is replaced by *deposit*, means that the agent is also able to adapt and perform some action that counts as *deposit*.
- The symbol — means that social abstraction and connection is not considered in that case for that agent.

Table 4 – Summary of social abstractions used in each case and the need to change the agent to adapt to them.

					Criterias		
Case	**Agent**	SF	N	P	CX1	CX2	CX3
C1	Bob						
C1	Tom			—		—	—
C2	Bob	✓			○		
C2	Tom	✓		—	○	—	—
C3	Bob	✓			○		
C3	Tom		✓	—	●	—	—
C4	Bob	✓		✓	●	●	●
C4	Tom		✓	—	●	—	—

C1: Case 1 - Simple MAS
C2: Case 2 - MAS with status functions
C3: Case 3 - MAS with status functions and norms
C4: Case 4 - MAS with status functions, norms and purposes

What can be observed through Table 4 and of the cases presented in this section is that agents become more adaptive to changes that may occur in the connections as social abstractions are added. For example, in case *C1*, because agents are unaware of social abstractions, any change in the *CX1*, *CX2*, or *CX3* connections requires reprogramming the agents to adapt to such changes. When social abstractions are incorporated into systems and agents become aware of them, they are able to adapt to changes that may occur in the connections. In case *C2*, for example, *Bob* and *Tom* are programmed to constitute and react to the constitution of a status function, respectively. In this case, if the actions change but the

institutional consequences remain the same, *Bob* and *Tom* can adapt to the changes without being reprogrammed. If the status function changes (symbol ○), the same is not true. Both agents are unable to adapt in this situation and need to be reprogrammed. The same holds true if the *CX2* and *CX3* connections are changed. The inclusion of norms (*N*) to the system makes the reactive agents adaptive. For example, in case *C3*, *Tom* is programmed to react to a norm that is activated whenever a status function is constituted. Therefore, *Tom* does not need to be aware of the status function to act accordingly, but at the same time, it remains institutionalized. The main advantage of this approach is that status functions can be modified as long as they remain activating the norm involving *Tom*. On the other hand, *Bob* remains with the same adaptation difficulties as in case *C2*. Finally, the addition of Purpose (*P*) makes cognitive agents adaptive (e.g., *Bob*). Purpose allows connecting to status functions and their environmental consequences (*CX2* connection) and associating these consequences with the agents' goals (*CX3* connection). The main advantage of this approach is that agents can be programmed to constitute status functions with environmental consequences that match their interests. Therefore, agents do not need to be reprogrammed and can adapt if connections *CX1*, *CX2*, and *CX3* are modified. Of course, these connections still need to be connected to the agent's goals.

Agents may ignore (or be unaware of) social abstractions and still achieve their goals. For example, the agents in case 4.1.1.1 are not programmed to consider any social abstractions. The scenario of case 4.1.1.4 has all social abstractions explicit and properly represented. Even without knowing the used social abstractions, agents in case 4.1.1.1, can act in the scenario of case 4.1.1.4 to satisfy their goals. However, for agents to act appropriately in this situation, they should be fully programmed for those conditions. *Bob* needs to perform *deliver_paper_note* action and this action needs to be institutionally considered as *payment*. If some social abstraction changes (e.g., *deliver_paper_note* no longer counts as *payment*), *Bob* no longer acts appropriately. In short, agents that do not know about social abstractions at design time can even act in scenarios where social abstractions are explicitly specified and represented. However, they need to be reprogrammed to adapt whenever a change occurs because they cannot benefit from such specifications.

It is important to be clear that the solution presented in this section solves the problem involved in the example but has some limitations. In this approach, only the agent that needs to act to achieve its goal can reason about purposes. If the other agent involved in the interaction also reasons about purposes, the solution may not work correctly. For example, consider that *Tom* has the goal of delivering the book and that the purpose associated with the status function *payment* points to a state of interest to *Tom*. In this case, if *Tom* reasons about purposes, it may infer that constituting *payment* might lead to satisfying its goal. However, it makes no sense for *Tom* to constitute a *payment*. On the contrary, *Tom* needs this constitution to be carried out by another agent to deliver the book and achieve its goal. The purpose model does not consider that agents that perform intermediate actions triggered by the institutional

action can also satisfy their goals. Currently, the model does not even consider the execution of intermediate actions so that the agent that performed the institutional action can achieve its goal. The model assumes that these actions are performed at some point, pointing to the states that might be reached if these actions are performed.

4.1.2 Application Example 2: Conquer territory

Rationality is an important feature for agents to consider as much information are available for their reasoning and deliberation process. This section evaluates the agents' rationality when social abstractions are used in open MAS. We are especially interested in the rationality that involves reasoning and deliberation about anti-goals. In this section, we use the scenario where the agent called *Bob* has the goal of *conquering a new territory* and the anti-goal of *not killing a soldier from the allied base*. There are two ways for the agent to achieve the goal in this system: (i) constituting any action that count as *forcing an attack* and (ii) constituting any action that count as *authorizing an attack*. The agent knows through some available guidelines that in this system the actions *broadcast a message* and *post on a web service* count as *forcing an attack* and *authorizing an attack* respectively. The difference between these constitutions are the effects they produce on the environment. While the first produces the effects *conquering new territory* and *killing a soldier from the allied base*, the second produces only the effect of *conquering new territory*. The distinction between the consequences is due to the status of the action. While the first implies forcing an action without regard for the consequences, the second is more frivolous. It has the status of authorizing only at the appropriate time. Section 4.1.2.1 describes an application whose connections are implicit in the agent code. Section 4.1.2.2 describes an application whose connections are explicit and agents can query them at any time. More details on this example and its implementation can be found in (CUNHA; HÜBNER, Jomi F; BRITO, M. de, 2022).

4.1.2.1 Case 1 - MAS with implicit status functions and purposes

This section considers an application where *Bob* is programmed to perform any action that counts as *forcing_an_attack*. The connections used to evaluate the example (cf. Section 4.1) are implicit in Bob's code. Therefore, the rationality of *Bob* is limited to the information available on previously specified connections.

Algorithm 11 describes the steps that an agent needs to do to satisfy its `territory_conquered` goal and avoid its `soldier_killed_from_allied_base` anti-goal. The algorithm receives as input a status function, its environmental consequences, and agent's anti-goal. From that input, it constitutes the status function if it does not have consequences that conflict with its anti-goal.

Algorithm 11 Algorithm for agents to achieve *territory_conquered* goal and avoid *soldier_killed_from_allied_base* anti-goal with status function

1: **Input:** *action* and its *statusFunction* **and** *statusFunctionConsequences* **and** *agentAntiGoal*
2: **if** *statusFunctionConsequences* $\supseteq$ *agentAntiGoal* **then**
3: **fail**
4: **else**
5: *action* $\leftarrow$ **look for** which *action* constitutes *statusFunction*
6: **do** *action*
7: **end if**

Listing 4.7 describes the Jason program that implements Algorithm 11. In this program, the agent is implemented with beliefs `anti_goal(soldier_killed_from_allied_base)` where `soldier_killed_from_allied_base` is the agent's anti-goal, `cause(forcing_an_attack,territory_conquered)`, `cause(forcing_an_attack,soldier_killed_from_allied_base)` where `territory_conquered` and `soldier_killed_from_allied_base` are the environmental consequences of constituting `forcing_an_attack` and `constitutive_rule(broadcast_a_message,forcing_an_attack,_,_)` where `broadcast_a_message` is the action available to constitute the status function `forcing_an_attack`. In this example, the agent is able to connect its goal of `territory_conquered` with the status function `forcing_an_attack` through the belief `cause(forcing_an_attack,territory_conquered)`. For that reason, it knows it should constitute `forcing_an_attack`. When executing this program, four steps are executed. First, *Bob* looks for which status function when constituted produces the desired effect. `forcing_an_attack` is the status function found. Second, *Bob* checks if the constitution of `forcing_an_attack` causes some effect that is anti-goal. Based on its belief base, it identifies that `forcing_an_attack` causes `soldier_killed_from_allied_base`. Then, the program fails. Supposing that the constitution could achieve the goal without without inflicting the anti-goal, the program would advance to the second step. Third, *Bob* queries what concrete actions can constitute `forcing_an_attack` status function. `broadcast_a_message` is the action found. Fourth, *Bob* performs `broadcast_a_message` action. Remember that in this example the program fails in the first step because `forcing_an_attack` causes `soldier_killed_from_allied_base`, which is also an anti-goal of *Bob*.

```
1  anti_goal(soldier_killed_from_allied_base).
2  cause(forcing_an_attack,territory_conquered).
3  cause(forcing_an_attack,soldier_killed_from_allied_base).
4  constitutive_rule(broadcast_a_message,forcing_an_attack,_,_).
5
6  !territory_conquered. // agent goal
7
8  +!territory_conquered : cause (SF,territory_conquered)
9       <- if (anti_goal(AG) & cause(SF,AG)){
10              .fail;
```

```
11          } else {
12              ?constitutive_rule(Action,SF,_,_);
13              Action;
14      }.
```

Listing 4.7 – Bob's program with implicit status functions and purposes.

In this application, the agent's rationality is limited to the information that the programmer previously encoded within it using beliefs. *Bob* is coded to not perform any action that constitutes a status function whose consequences match its anti-goal. From its belief base, *Bob* has enough information to infer that the constitution of `forcing_an_attack`, while satisfying its goal, also inflicts its anti-goal. This solution works because this information is hard-coded into *Bob*. However, this information may not be predictable in open systems and can change anytime. Furthermore, the agent is limited to considering only the constitution programmed within it. In this example, the system provides another constitution (i.e., *authorizing an attack*) that can satisfy Bob's goal without inflicting its anti-goal. However, *Bob* is not coded to exploit this constitution. In this solution, the agent cannot capture information from the scenario where it is acting at runtime to use it in its reasoning and deliberation process.

In short, all connections are implicit, generating a tight coupling between the agent and the solution. The problem with this approach is that the agent's reasoning and deliberation process is limited to the information previously programmed in its belief base. For this reason, (a) if the actions or their institutional consequences change, or (b) if the environmental consequences change, (c) if agents' interests change, or (d) even if there are other options of institutional actions to satisfy the agent's goal without inflicting the anti-goal, in all cases, the agents may not be able to reason about this information and eventually perform actions that inflict anti-goals or not perform any action (even if there is a possibility).

4.1.2.2 Case 2 - MAS with explicit status functions and purposes

This section considers an application where *Bob* is programmed to constitute any action as long as the status function produces the environmental consequences that match its goal and do not inflict its anti-goal. Furthermore, the connections used to evaluate the example (cf. Section 4.1) are explicitly captured by proper abstractions and represented in the institutional specification, shown in Listing 4.8. The rationality of *Bob* improves because it can query this institutional specification and discover information at runtime.

Listing 4.8 – Battle Institutional Specification

```
status_functions: forcing_an_attack, authorizing_an_attack

Constitutive_rules:
1: broadcast_a_message count-as forcing_an_attack;
2: posting_on_a_webservice count-as authorizing_an_attack.

Purpose Definition:
```

```
hasPurpose(forcing_an_attack,force_attack);
hasPurpose(authorizing_an_attack,authorize_attack);
hasConsequence(force_attack,[territory_conquered,soldier_killed_from_allied_base]);
hasConsequence(authorize_attack,[territory_conquered]).
```

Algorithm 12 describes the steps that an agent needs to do to satisfy its `territory_conquered` goal and avoid its `soldier_killed_from_allied_base` anti-goal. The algorithm receives as input an agent's goal and anti-goal. From that input, it tries to perform an action to satisfy the agent's goal by excluding actions that also satisfy the agent's anti-goal.

Algorithm 12 Algorithm for agents to achieve their goals and avoid their anti-goals with status functions and purposes

Input: *agentGoal* **and** *agentAntiGoal*
actionList $\leftarrow$ **call** `alg1` (*agentGoal*)
while *actionList* $\neq \{\}$ **do**
    action $\leftarrow$ first(*actionList*)
    statesList $\leftarrow$ **call** `alg2` (*action*)
    if *antiGoal* $\subseteq$ *statesList* **then**
        actionList $=$ *actionList* $\setminus \{$*action*$\}$
    else
        do *action*
    end if
end while

Listing 4.9 describes the Jason program that implements Algorithm 12. In this program, the agent is implemented with the belief `anti_goal(soldier_killed_from_allied_base)` where `soldier_killed_from_allied_base` is the agent's anti-goal. When executing this program, four steps are executed. First, *Bob* identifies `soldier_killed_from_allied_base` as it anti-goal by consulting its belief base. Second, *Bob* uses `alg1` to find the actions that can satisfy `territory_conquered` goal (i.e., its goal) and gets a queue of actions as a result. Third, *Bob* informs the queue of actions to the plan `try_actions`. If the queue is empty, the program fails. If there are elements, the plan looks for the states that can be reached in the system if the first action in the queue is performed and proceeds to the next step. Fourth, the plan checks whether the action reaches a state that also corresponds to Bob's anti-goal. If true, the plan takes the next element in the queue and repeats the check again. If false, it executes the action and the program ends. In this example, the program receives in step two a queue with the actions `broadcast_a_message` and `posting_on_a_webservice` that are found in Listing 4.8. In the following steps, the program tries to perform action `broadcast_a_message` and gives up because it satisfies the anti-goal of `soldier_killed_from_allied_base` and then successfully performs action `posting_on_a_webservice`.

```
1 anti_goal(soldier_killed_from_allied_base).
2 !territory_conquered.  // agent goal
3
4 +!territory_conquered <-
```

```
5       !alg1(territory_conquered, Actions);
6       !try_actions(Actions).
7
8   +!try_actions(Actions) <-
9       if(.length(Actions) < 1){
10          .fail;
11      }
12      .queue.head(Actions,Action);
13      !alg2(Action, States);
14      if ( anti_goal(AG) & .member(AG,States) ) {
15          .queue.remove(Actions, Action);
16          !try_actions(R); // try another action
17      } else {
18          Action; // executes an action
19      }.
```

Listing 4.9 – Bob's program with explicit status functions and purposes.

In this application, the agent's rationality is improved since it is not limited to reasoning about the information previously encoded within it. This solution presents all evaluated connections explicitly captured by appropriate abstractions (i.e., status function and purpose), which are represented in the institutional specification. *Bob* is programmed to explore these connections by looking for (i) a purpose that points to its *territory_conquered* goal, (ii) status functions related to that purpose, in the case of this example, *forcing_an_attack* and *authorizing_an_attack*, and (iii) concrete actions that can constitute that status functions, in the case of this example, `broadcast_a_message` and `posting_on_a_webservice` actions. While these advantages are the same discussed in Section 4.1.1.4, this example also highlights that, with the introduction of purposes, the agent can evaluate the institutional actions from the consequence of the constitution that they produce and not just from the action itself. As a consequence, the agent can discard actions that produce anti-goals. In summary, this solution highlights two improvements in the agent's reasoning and deliberation process. First, it can reason about constitutions that also inflict anti-goal. Second, it can reason based on the status functions and the consequence of its constitution. This allows the agent to explore some status function available in the system, not just the ones it knows about in advance. This is possible because the agent can capture information from the scenario where it is entering and use them in its reasoning and deliberation process. In open systems, this solution seems to be more appropriate because this information may change and the agent needs to consult it to make the best decision according to its interests.

In short, all connections are explicit, captured by first-class abstractions, and represented in the institutional specification. For this reason, the agent becomes decoupled from the scenario in which it is entering. Therefore, (a) if the actions or their institutional consequences change, or (b) if the environmental consequences change, (c) if agents' interests change, or (d) even

if there are other options of institutional actions to satisfy the agent's goal without inflicting the anti-goal, in all cases, agents have the means to consult this information at runtime and use it in their reasoning and deliberation process according to their interests.

4.1.2.3 Discussion

This section discusses the rationality that agents acquire in open MAS when the evaluated connections become explicit. Furthermore, this section also explores the problem of possible consequences of status function constitutions that cause conflicts between goals and anti-goals. The first example illustrates the limitation of agents in considering previously encoded information in their reasoning and deliberation process. The solution is tightly coupled to the scenario and may no longer work if any connections are modified. In addition, the agent cannot explore other available constitutions in the system where it is acting because it cannot reason about the system at runtime. In other words, the agent is limited to considering only the previously codified constitution and its environmental consequences. If the system has better options that can satisfy its goal without inflicting an anti-goal, it cannot exploit them.

The second example overcomes the limitation of the agents' rationality considering the connections evaluated. This is because connections are captured by first-class abstractions and made explicit in the system. The agents can query this information at runtime and use it in their reasoning and deliberation process. The solution decouples agents from the scenario in which they are acting. Furthermore, agents can explore other constitutions available in the system that satisfy their goals without infringing on their anti-goals. In short, we have improved agent decision-making since it has more information available to help it to decide whether to achieve its goals or avoid its anti-goals. With the proposed model, agents can access and reason about the consequences of institutional actions and adapt themselves to different scenarios. They can reason that (a) some purposes point to states that are similar to their interests and therefore useful to reach their goals or (b) avoid these purposes because they point to states that are similar to their anti-goals. In both cases, the agent has more information while deciding whether a particular action will help it or not. This kind of reasoning is important for advances in agents' autonomy (RODRIGUEZ-AGUILAR, Juan A. et al., 2015).

4.1.3 Application Example 3: Posting information on social networks

Flexibility is an important feature for agents considering the possibility in open systems whose available actions and their consequences cannot be predicted in agents' design time. This section evaluates the agents' flexibility to act in this kind of system when they are endowed with status functions and purpose. The remainder of this section, consider a scenario where the agent *Bob* has the goal of *published_information* on different social networks. Furthermore, *Bob* should avoid its anti-goal of *fake_news_spread*. Section 4.1.3.1 describes a first implementation that does not use the proposed model. Section 4.1.3.2 describes a second implementation that uses the proposed model.

4.1.3.1 Case 1 - Social networks without status functions and purposes

This section considers an application without status functions and purposes. Algorithm 13 describes the steps that an agent needs to perform to satisfy its `published_information` goal and avoid its `fake_news_spread` anti-goal. The algorithm receives as input the actions' consequences, the name of the social network where it is entering and the agent's anti-goal. From that input, it selects the plan that corresponds to the social network that it is entering, identifies the action that should be performed and executes it if it does not infringe its anti-goal.

Algorithm 13 Algorithm for agents to achieve *published_information* goal and avoid *fake_news_spread* anti-goal without status functions and purposes

1: **Input:** plans **and** *actionsConsequences* **and** *socialNetwork* **and** *agentAntiGoal*
2: **do** select the plan that corresponds to (*socialNetwork*)
3: *action* $\leftarrow$ **look for** which *action* need to be performed
4: **if** *actionsConsequences* $\supseteq$ *agentAntiGoal* **then**
5: **fail**
6: **end if**
7: **do** *action*

Listing 4.10 describes the Jason program that implements Algorithm 13. In this program, we assume that when the agent enters different systems, it acquires the belief $knet(S)$ where $S \in \{twitter, telegram, instagram, facebook\}$ is the variable for the name of the social network the agent is currently acting. Furthermore, in this program, the agent is implemented with the beliefs `anti_goal(fake_news_spread)` meaning that `fake_news_spread` is the agent's anti-goal, `cause(sendMessageByTwitter,fake_news_spread)`, `cause(talkWithBot,fake_news_spread)`, `cause(uploadAPIcture,fake_news_spread)` and `cause(uploadAMessage,fake_news_spread)`, where `fake_news_spread` is the consequence of performing the actions `sendMessageByTwitter`, `talkWithBot`, `uploadAPIcture` or `uploadAMessage`. Consider that *Bob* starts believing `knet(facebook)` when it enters the *Facebook* system. When executing this program, three steps are executed. First, *Bob* identifies `fake_news_spread` as its anti-goal and also as a consequence of actions `sendMessageByTwitter`, `talkWithBot`, `uploadAPIcture` and `uploadAMessage` by consulting its belief base. Second, *Bob* selects the plan that has the context that corresponds to `knet(facebook)` belief. Third, *Bob* checks if the action to be performed (`uploadAMessage` in this case) conflicts with its anti-goal. If it is the case, the program fails. If it is not case, proceed to the next step. Fourth, *Bob* performs the action stated in the plan. In this example, the program fails in the third step because `uploadAMessage` causes `fake_news_spread`, which is the anti-goal of *Bob*.

```
1 anti_goal(fake_news_spread).
2 cause(sendMessageByTwitter,fake_news_spread).
3 cause(talkWithBot,fake_news_spread).
```

```
4  cause(uploadAPIcture,fake_news_spread).
5  cause(uploadAMessage,fake_news_spread).
6
7  !published_info. // agent goal
8
9  +!published_info : knet(twitter)
10    <- if (anti_goal(AG) & cause(sendMessageByTwitter, AG)){
11          .fail;
12       } else {
13          sendMessageByTwitter;
14       }.
15
16 +!published_info : knet(telegram)
17    <- if (anti_goal(AG) & cause(talkWithBot, AG)){
18          .fail;
19       } else {
20          talkWithBot;
21       }.
22
23 +!published_info : knet(instagram)
24    <- if (anti_goal(AG) & cause(uploadAPIcture, AG)){
25          .fail;
26       } else {
27          uploadAPIcture;
28       }.
29
30 +!published_info : knet(facebook)
31    <- if (anti_goal(AG) & cause(uploadAMessage, AG)){
32          .fail;
33       } else {
34          uploadAMessage;
35       }.
```

Listing 4.10 – Bob's program without status functions and purposes.

In this application, the agent's flexibility is limited because the agent can only act on social networks that it was previously programmed for. In this example, *Bob* has been coded to act on the social networks *Twitter*, *Telegram*, *Instagram* and *Facebook*. This solution only works on the aforementioned social networks and because the connections between actions, their consequences and agent's interests are encoded within it. We assume that no system has the necessary first-class abstractions to capture these connections explicitly. For this reason, *Bob* is not flexible enough to work on other systems that it does not know about at design time without having to be reprogrammed.

In short, all connections are implicit because it lacks social abstractions to represent them. They are encoded internally in the agent, and therefore, it is limited to act in the social networks that it is previously programmed, perform previously defined actions and needs to be reprogrammed to act in new systems. These limitations make the agent inflexible. For example, consider that for some reason, *Bob* needs to act on the social network *TikTok*. Even if the social network provides the means that allow *Bob* to achieve its goal, *Bob* is not prepared to exploit these means since it does not have any plan to interact with the social network *TikTok*. That is, *Bob* is only able to interact and perform actions on social networks that it previously knows. Therefore, (a) if the actions or their institutional consequences change, or (b) if the environmental consequences change, or (c) if the agent moves to a new system, or even (d) if the agent's interests changes, in all cases, the agent needs to be reprogrammed because it is not flexible.

4.1.3.2 Case 2 - Social networks with status functions and purposes

This section considers an application where *Bob* is programmed to constitute any status function whose constitution produces the environmental consequences that match its goal and do not inflict its anti-goal. Furthermore, each social network has an institution, whose the institutional specifications are shown in Listings 4.11, 4.12, 4.13 and 4.14. The institutional specifications explicitly and externally represent the connections considered in the evaluation of the work (cf. Section 4.1). Because of these connections, the agent's flexibility is increased.

Listing 4.11 – Twitter Institutional Specification

```
status_functions: tweet

Constitutive_rules:
1: sendMessageByTwitter count-as tweet.

Purpose Definition:
hasPurpose(tweet,transmit_information);
hasConsequence(transmit_information,[published_info,fake_news_spread]).
```

Listing 4.12 – Telegram Institutional Specification

```
status_functions: messageByTelegram

Constitutive_rules:
1: talkWithBot count-as messageByTelegram.

Purpose Definition:
hasPurpose(messageByTelegram,transmit_information);
hasConsequence(transmit_information,[published_info,fake_news_spread]).
```

Listing 4.13 – Instagram Institutional Specification

```
status_functions: postByInstagram

Constitutive_rules:
1: uploadAPicture count-as postByInstagram.

Purpose Definition:
hasPurpose(postByInstagram,transmit_information);
hasConsequence(transmit_information,[published_info,fake_news_spread]).
```

Listing 4.14 – Facebook Institutional Specification

```
status_functions: postByFacebook

Constitutive_rules:
1: uploadAMessage count-as postByFacebook.

Purpose Definition:
hasPurpose(postByFacebook,transmit_information);
hasConsequence(transmit_information,[published_info,fake_news_spread]).
```

Algorithm 12, explained in Section 4.1.2.2, can also be used to describe the steps an agent needs to take to satisfy its goal `published_information` and avoid its anti-goal `fake_news_spread`. Listing 4.15 describes the Jason program that implements Algorithm 12. In this program, the agent is implemented with the belief `anti_goal(fake_news_spread)` where `fake_news_spread` is the agent's anti-goal. When executing this program, four steps are executed. First, *Bob* identifies `fake_news_spread` as it anti-goal by consulting its belief base. Second, *Bob* uses `alg1` to find the actions that can satisfy `published_information` goal (i.e., its goal) and gets a queue of actions as a result. Third, *Bob* informs the queue of actions to a plan. If the queue is empty, the program fails. If there are elements, the plan looks for the states that can be reached in the system if the first action in the queue is performed and proceeds to the next step. Fourth, the plan checks whether the action reaches a state that also corresponds to Bob's anti-goal. If true, the plan takes the next element in the queue and repeats the check again. If false, it executes the action and the program ends. The queue of actions that the program receives in step 2 depends on which social network *Bob* is acting on. For example, consider that *Bob* is acting on *Facebook* social network. In this case, the queue has the action `uploadAMessage` that is found in Listing 4.14. In the following steps, the program tries to perform `uploadAMessage` action and gives up because it satisfies the anti-goal of `fake_news_spread`.

```
1 anti_goal(fake_news_spread).
2 !published_info.  // agent goal
3
4 +!published_info <-
5     !alg1(published_info, Actions);
6     !try_action(Actions).
7
```

```
8  +!try_actions([Action|R]) <-
9      if(.length(Actions) < 1){
10         .fail;
11     }
12     .queue.head(Actions,Action);
13     !alg2(Action, States);
14     if ( anti_goal(AG) & .member(AG,States) ) {
15         .queue.remove(Actions, Action);
16         !try_action(R); // try another action
17     } else {
18         Action; // executes an action
19     }.
```

Listing 4.15 – Bob's program with status function and purposes.

In this application, the agents' flexibility is improved because they are not limited to act in social networks that they previously know. This solution presents all evaluated connections explicitly captured by appropriate abstractions (i.e., status function and purpose) and they are represented in the institutional specification. *Bob* is programmed to explore these connections by looking for (i) a purpose that points to its *published_information* goal, (ii) status functions related to that purpose, and (iii) concrete actions that can constitute that status functions. In the case of this example, this information depends on which social network *Bob* is acting. Furthermore, *Bob* is also programmed to check the environmental consequences of constitutions and ignore those that also reach its anti-goal. Therefore, the explicitness of these connections allows *Bob* to act in different social networks that may even be unknown at design time and explore in these networks any actions that may satisfy its interests.

In short, all connections are explicit, captured by first-class abstractions, and represented in the institutional specification. For this reason, the agent becomes flexible since it is not limited to acting on social networks that it has previously programmed and can perform actions that it discovers at runtime. For example, consider again that agent *Bob* needs to act on the social network *TikTok*. Upon entering this system, *Bob* can query which action can satisfy its goal and if that action does not imply an anti-goal. That is, *Bob* can enter in social networks unknown at project time as long as the purposes related to status functions are explicit. Therefore, (a) if the actions or their institutional consequences change, or (b) if the environmental consequences change, or (c) if the agent moves to a new system, or even (d) if the agent's interests changes, in all cases, the agent does not need to be reprogrammed because it is flexible to scenarios that have the means to satisfy its interests.

4.1.3.3 Discussion

This section discusses the flexibility that agents acquire in open MAS when the evaluated connections become explicit. The first example illustrates the limitation of agents in acting

in previously known social networks and performing previously coded actions. The solution is tightly coupled to social networks that are considered at design time. Also, if any of the connections change at runtime, the agent may stop functioning properly. In other words, the agent is limited to act in specific social networks and cannot act in other social networks even if these networks provide the means to satisfy its interests.

The second example overcomes the limitation of agent flexibility considering the connections evaluated. This is because these connections are captured by first-class abstractions and made explicit in the institution. In other words, the connections between the agents' goals and anti-goals and the institutional actions are externalized: they are moved from the agent program to the institution, in the form of purposes. Given the external specification of purposes, the agent can find out: (i) which action can satisfy its goal, (ii) what are the system consequences of performing an institutional action, and (iii) how it can achieve a goal in the system without inflicting an anti-goal in whatever system they are acting on. In both cases, the agent has more flexibility to act in other systems without having to be reprogrammed, as long as these systems provide the means that allow it to satisfy its interests.

4.1.4 Practical conclusions

The objective of this section is to evaluate the characteristics of *adaptability*, *rationality* and *flexibility* from the agents' perspective. Some practical examples are developed to evaluate each of these characteristics in Sections 4.1.1, 4.1.2 and 4.1.3 respectively. We observe that as social abstractions are added and captured by status functions, norms, purposes, etc., agents become aware of them and approach the evaluated characteristics. In Section 4.1.1, as social abstractions are added, and agents become aware of them, they adapt to the system. In Section 4.1.2, agents gain the ability to reason about information that may be unknown at design time. In Section 4.1.3, they gain the flexibility to act in scenarios that are not initially designed for. Furthermore, social abstractions, especially purpose, explicitly add some connections that were hitherto implicitly specified. The explicitness of these connections allows them to be changed without requiring that agents are reprogrammed. That is, purpose makes it easier for agents to act in open systems, which is essential due to the characteristics of these systems.

Solutions that consider purposes are more generic and can be applied in any scenario that has the purpose specified and represented explicitly. For example, the agent codes in Sections 4.1.2.2 and 4.1.3.2 are essentially the same. The changes are in the agent's goals, which is natural since they are designed for different scenarios. Furthermore, the difference between the cases cited and the agent code in Section 4.1.1.4 is the absence of `alg2`, which is used to find the environmental consequences resulting from institutional actions. In this case, it is not used because the solution does not have the objective of filtering the actions that can also inflict the anti-goals of the agents. In other words, as long as the above information is modified, these agents' codes can be exchanged between these sections without affecting

their functionality.

4.2 POSITIONING OF THE PURPOSE MODEL IN THE ARTIFICIAL INSTITUTION LITERATURE

According to Luck et al. (LUCK, Michael et al., 2012), the complexity of designing MAS increases when they are open systems. Open MAS are populated by heterogeneous agents, possibly developed by different people using different languages and architectures, representing different parties, and acting to achieve individual goals (RICCI; PIUNTI; VIROLI, 2011). In this context, *social abstractions* (e.g., norms, constitutive rules) are used to help agents to act in open systems. These abstractions have been considered in several MAS approaches, varying the explicitness and externality of social abstractions.[1]

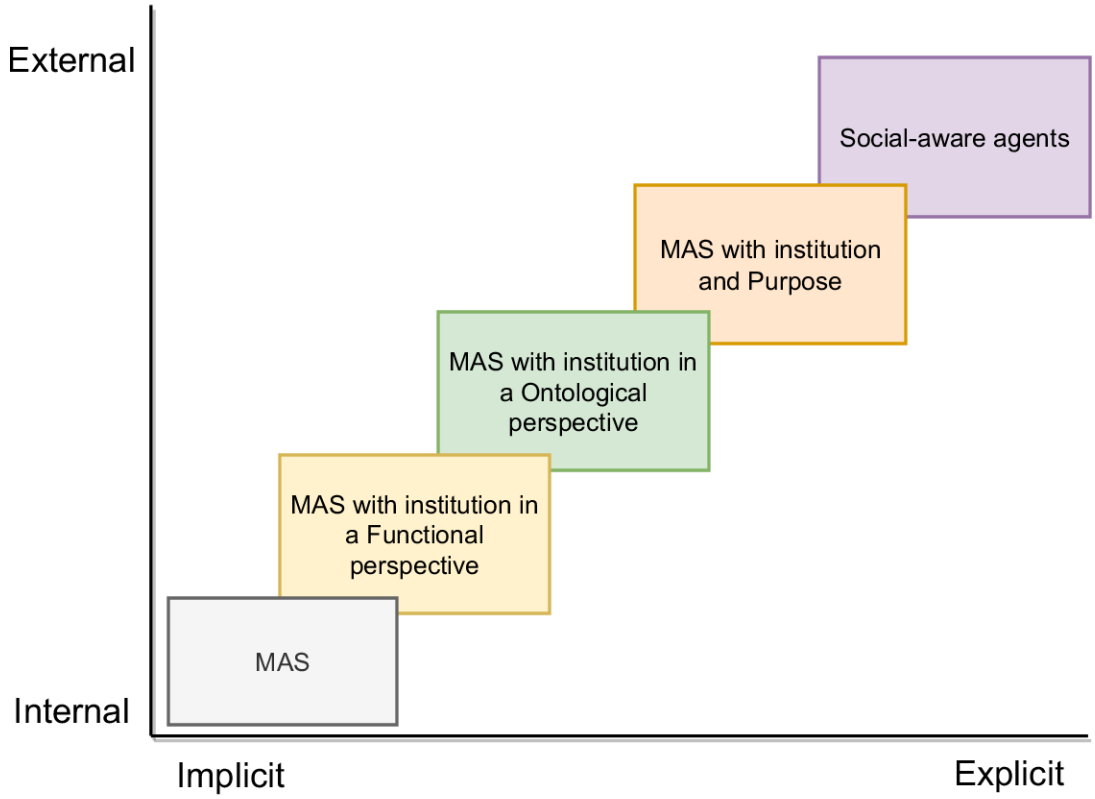

Figure 10 – Categories of systems based on the used social abstractions.

Figure 10 positions MAS with respect to implicit-explicit and internal-external characteristics as a function of the social abstractions that the system contains. Each rectangle in the figure represents an MAS category. The notions of explicitness and externality are inspired in (DIGNUM; ALDEWERELD; DIGNUM, F., 2011; LEMAÎTRE; EXCELENTE, 1998). The implicit-explicit horizontal axis represents the amount of explicit representation of social abstractions. Approaches on the far left have little or no use of social abstractions. Social abstractions

[1] Among the huge literature in the field, the interested reader can find detailed information on social abstractions in the COINE workshop series `http://www.pcs.usp.br/~coin/`

are usually implemented and reduced to concepts available in agent programming languages (e.g., the social concept of role is implemented by a set of beliefs). The approaches on the far right have social abstractions fully represented, as first-class abstractions. The internal-external axis represents the level of the externality of the representation of social abstractions (i.e., where they are stored). Bottom approaches have (parts) of social abstractions implemented within agents, while top approaches have a shared (common) representation of these social abstractions available outside the agents. As the social abstractions becomes more explicit and external, the system remains in execution regardless of which agents act in it. This is because social abstractions provide guidelines for any agents to act in these systems (ROCHA COSTA, 2014).

Dignum et al. (DIGNUM; ALDEWERELD; DIGNUM, F., 2011) use these notions to position existing frameworks concerning the use of organizational concepts. Inspired by this work, we use these notions to analyze each system category illustrated in Figure 10 from an institutional perspective. We focus on the social abstractions present in the system and on the support given to incoming agents.

MAS. In simple MAS, the social abstractions are mainly in the designer's mind. It is, in this sense, the absolute cornerstone of our diagram, as social abstractions are completely internal (represented in the agents' code) and implicit (all social abstractions are translated into concepts of agents). Only the designer knows whether institutional actions affect the environment and allow agents to achieve their goals or avoid their anti-goals. Therefore, changes in institutional actions or their consequences require that agents are reprogrammed. Examples of this case can be seen in Section 4.1.1.1.

MAS with institution in a functional perspective. Some works use the notion of institution and introduce constitutive rules (i.e., *count-as* rules) combined with some regulative representation, such as norms and organizations. These works are called functional because they connect *count-as* rules with the state of regulative representation (BRITO, M. d.; HÜBNER, Jomi F., 2014; CUNHA; HÜBNER, Jomi F; BRITO, M. de, 2019). In this way, they solve an interoperability gap between the different dimensions that make up the system. For example, Dastani et al. (DASTANI et al., 2009; DASTANI; VAN DER TORRE; YORKE-SMITH, 2012) propose count-as rules to define the environmental conditions that trigger changes in the state of the norm instances (i.e., brute facts count-as violations, fulfillments, etc. of the norms). In this case, one could specify that "broadcast a message" counts as the fulfillment of the norm "internauts are obliged to tweet". In a similar direction, Piunti et al. (PIUNTI et al., 2010) consider that environmental events count-as changes in the state of Moise organizations (HÜBNER, Jomi F; BOISSIER; BORDINI, 2011; HÜBNER, Jomi F et al., 2010). Some works (CAMPOS et al., 2009; MUNTANER-PERICH; DE LA ROSA ESTEVA, 2007) make explicit and external only the regulative consequence of the brute facts. However, the meaning of these consequences with respect to the application domain remain implicit and internal. The relationship of these consequences with the goals and anti-goals it is also implicit

and internal. The practical examples illustrated in Section 4.1 do not use institutions in this way.

MAS with institution in a ontological perspective. Some works use *count-as* rules to specify the constitution of institutional elements that belong to the application domain, that we refer in this work as status functions (ALDEWERELD, Huib et al., 2010; FORNARA et al., 2008; BOELLA, Guido; VAN DER TORRE, 2004; BOELLA, Guido; TORRE, L. v. d., 2006; BOELLA; REGULATIVE, n.d.; CARDOSO, Henrique Lopes; OLIVEIRA, EugENio, 2007; CLIFFE; VOS; PADGET, 2006; BRITO, M. d. et al., 2016; DE BRITO; HÜBNER, Jomi Fred; BOISSIER, 2018). These institutional elements are mainly used in the specification of norms (but not exclusively). The norms become stable since their specification do not change even in the case of changes in the constitution of the elements they refer to (DE BRITO; HÜBNER, Jomi Fred; BOISSIER, 2018). Compared with the previous category, this category of MAS adds external and explicit representation of the constituted elements. The agents can reason about the constitution of these elements even in institutions they do not know in advance. They can also reason about the normative consequences of constitution of status functions. However, the effect of the institutional action that allows an agent to satisfy its goal or prevent its anti-goal remains translated (implicitly) within the abstractions used to code the agent and is programmed (internally) into it. Examples of this case can be seen in Sections 4.1.1.2, 4.1.1.3, 4.1.2.1, and 4.1.3.1.

MAS with institution in a purpose perspective. As in the preceding categories, this category uses the notion of the institution and *count-as* to introduce a social meaning to the elements present in the system (agents, events, etc.). Furthermore, it introduces the concept of *purpose*, which describes the functions of the status functions in terms of the consequences that their constitution produces in the environment. The addition of this concept makes explicit the connections between the institutional actions available in the system and their effects in the environment. Making this information explicit and external helps agents to act in systems composed of an institution they did not know previously. They can consult information described in the institutional specification to reason about the consequences of their actions and deliberate on whether or not to act in their interests. The proposal presented in this thesis can be classified in this category: it extends previous works adding the purposes of status functions. In the proposal, purposes are explicit and external: they have an explicit specification (written in OWL) external to the agents (in environmental artifacts). Examples of this case can be seen in Sections 4.1.2.2, and 4.1.3.2.

Social-aware agents. The social-aware agents shown in the upper right corner of Figure 10 correspond to an ideal category where agents themselves combine the status functions that make up the institutions with the consequences of carrying out institutional actions in the environment they are interacting. The institution and the purposes related to institutional actions emerge from the interactions of agents (or are derived from their needs). These systems do not have institutional specifications provided by the system designers because they emerge

at runtime. The agent is part of the interaction that defines these social abstractions (knowing them since then). No current implementation of such systems is known at the moment. Examples of this approach do not yet exist because there is a lack of appropriate abstractions to capture the social combinations that must be carried out so that this solution can actually be implemented.

5 CONCLUSIONS

The main contribution of this work is a model of purpose that allows specifying the environmental consequences of the constitution of status functions in artificial institutions. This general contribution results from the work developed to solve the questions introduced in Section 1.1. These questions, described in the sequence, provide additional details on the contributions of this work.

1. *What are the appropriate abstractions to represent the consequences in the environment of institutional actions?* Answers to this question are given in Chapter 2. According to Searle's theory, the institutional reality exists because people assign functions to elements of brute reality that are not inherent in their natural virtues (i.e., status functions). These functions can modify the world according to the interests of the people who assign the functions. Searle calls these interests of *purpose*. Therefore, inspired by this conception, we consider that the consequences in the environment of the execution of institutional actions are *purposes* and are associated with the practical interests of agents present in artificial institutions.

2. *How the abstractions that represent institutional reality and the environment can be coupled with the representation of the environmental consequences of institutional actions?* Answers to this question are given in Chapter 3. We define that the conceptual coupling occurs through the status functions because they are the statuses that enable the execution of the functions that modify the world to satisfy the purposes of the agents. That is, purposes are associated with status functions on the institutional side and environmental states on the environmental side. The practical coupling between these concepts is slightly different. We define the purpose as an independent abstraction to be connected with the representations of status functions in the artificial institution implementations. Sections 3.5 and 3.6 describe a possibility of implementing the *purpose* in a MAS. This implementation specifies the purpose model using an ontology. The connection of purpose with the model that represents the institutional reality is performed in an ad-hoc way. That is, the designer of the system needs to equally specify the status functions in the artificial institution model and in the purpose model. The connection of purpose with the system states occurs through the ontology that describes the states that are pointed to by the purpose. There are disadvantages and advantages to the proposed practical coupling proposal. The disadvantages of this solution are (i) rework to doubly specify the status function and (ii) the possibility of inconsistency between status functions of both specifications. The advantages are (i) the flexibility of the model since it can be coupled to any model of institutional reality with an institutional counterpart for events that occur in the system and (ii) the non-influence of the purpose model on the institution's dynamics. In other words, the institution's dynamics, especially

the conditions for constituting the status functions and the constitution process, is totally independent of the purpose model.

3. *How agents can exploit them to act according to their interests?* Answers to this question are given in Section 4.1. One of the advantages of purposes in MAS regards the agents. We have an improvement in the decision-making process of the agents since they have more information available to decide how to achieve their goals and avoid their anti-goals. With the purpose model, agents can access and reason about the consequences of institutional actions and adapt themselves to different scenarios. They can notice that (a) some purposes point to states of the environment that are match to their interests and therefore useful to reach their goals or (b) avoid these purposes because they point to states that are similar to their anti-goals. In both cases, the agent has more information while deciding whether a particular action will help it or not. This kind of reasoning improves the agents' autonomy (RODRIGUEZ-AGUILAR, Juan A. et al., 2015). Furthermore, the agents become more adaptive. When purposes are external and explicit, agents can enter different institutions without the need to be reprogrammed. The proposed model allows an explicit connection between the purposes and the states of the world that are likely reached from the constitution of status functions. If an agent is programmed to achieve its goal or avoid its anti-goal considering purposes, it can act in different institutions that supports purposes without changes in its code. The agent's capability to adapt to different institutions is an important advance, especially in open systems (ALDEWERELD, Huib; DIGNUM, Virginia, 2010; ZAMBONELLI; JENNINGS; WOOLDRIDGE, 2000). In this case, the code of both agents and institution remains stable, as the link between them is externalized to the concept of purpose.

The previous answers are consequences of the main objective motivating this work, that is: *the design of artificial institutions that not only specify count-as relations, but that also have the means of specifying the consequences in the environment of the execution of institutional actions*. From all the developed work, we conclude that the consequences in the environment of the execution of institutional actions can be appropriately represented through the purposes and associated with the status functions on the institutional side and the states in the environment on the environmental side. This conclusion is directly related to the inspiration taken from the work by John Searle (SEARLE, J. R., 1995; SEARLE, J., 2010). It is important to remark that our computational representation is one among other possible adaptations of that work to the MAS field.

This work also contributes to advancing the notion of the institution in MAS. While current works limit themselves to employing institutional reality to represent a common interpretation of elements present in the environment, this work associates institutional reality with states in the environment that are of interest to agents.

5.1 FUTURE WORK

By developing this thesis, some points have been observed that could be investigated in future work. Some of them are described below:

1. **Explore the relationship between the purposes and values of agents.** Currently, the association between the purposes and interests of agents is developed in an ad-hoc manner. However, as discussed in Section 4.2, one possibility for future work is to make agents socially aware. The agents could combine the status functions that make up the institutions with the consequences of carrying out institutional actions in the environment in which they interact. The institution and the purposes related to institutional actions must emerge from the interactions of agents (or derive from their needs).

2. **Evaluate whether the purpose model must be further detailed.** Currently, the specification of purposes is used to point out the states that can be reached if the constitutions of actions are carried out. However, we have assumed that some intermediate actions must be performed for these states to be reached. In this thesis, these actions are ignored. A possibility for future work is the addition of these intermediate actions and the pre and post-conditions for their execution. Such an addition will allow the pointed states to be guaranteed.

3. **Evaluate the integration of the model with other models that implement artificial institutions.** Currently, the purpose model is used together with SAI (cf. Section 4.1). However, the integration is developed in an ad-hoc manner and requires the designer to know both models. A possibility for future work is to develop an API that integrates the model with other models of institutional reality without the need for the designer to know the details of the purpose model.

4. **Method to specify institutions.** The design of the purpose of the practical examples in Section 4.1 was done in an ad-hoc manner. A possibility for future work is the development of methods to properly carry out the design of these systems. It is interesting, for example, to analyze, from a use case, what is the best order to design the system, defining whether to start with status functions, constitutive rules, purposes, etc. Additional aspects that can be addressed are extracting purposes from use cases, connecting those purposes with status functions, etc.

5.2 RELATED PUBLICATIONS

Publications related to this thesis are enumerated in the sequence:

1. Cunha, Rafhael R., Jomi F. Hübner, and Maiquel de Brito. Instituições em sistemas multiagentes à luz da teoria da construção da realidade social. **XIII Workshop-**

Escola de Sistemas de Agentes, seus Ambientes e Aplicações. WESAAC, 2019. p. 71-82.

This article analyzes the state of the art on artificial institutions in MAS considering the theory of social reality proposed by John Searle.

2. Cunha, Rafhael R., Jomi F. Hübner, and Maiquel de Brito. Modelo Ontológico de Status-Functions em Instituições Artificiais. **XIV Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações. WESAAC, 2020.** p. 37-48.

   This article presents the first version of the purpose model through the conception of an ontology and discusses some properties of that model.

3. Cunha, Rafhael R., Jomi Fred Hübner, and Maiquel de Brito. A Conceptual Model for Situating Purposes in Artificial Institutions. **Revista de Informática Teórica e Aplicada. Vol.29 No.1, 2022.** p. 68-80.

   This article presents the conceptual model of the purpose model.

4. Cunha, Rafhael R., Jomi F. Hübner, and Maiquel de Brito. Implementing a purpose model of status-functions through ontologies to support the social reasoning of agents. **XV Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações. WESAAC, 2021.** p. 71-82.

   This article presents the first practical implementation of the purpose model using an ontology and the JaCaMo framework.

5. Cunha, Rafhael R., Jomi F. Hübner, and Maiquel de Brito. Environmental Consequences of Institutional Facts in Artificial Institutions. **International Workshop on Coordination, Organizations, Institutions, Norms, and Ethics for Governance of Multi-Agent Systems XIV. COINE 2021. Springer, 2022.** p. 44-61

   This article illustrates the disadvantages and advantages of using the purpose model for the development of a MAS by implementing a practical example using an ontology and the JaCaMo framework.

6. Cunha, Rafhael R., Jomi F. Hübner, and Maiquel de Brito. Supporting the Reasoning about Environmental Consequences of Institutional Actions. **International Workshop on Coordination, Organizations, Institutions, Norms, and Ethics for Governance of Multi-Agent Systems XV. COINE 2022. Springer, 2022.** p. 134-147

   This article presents a mathematical formalization for the model of purposes and algorithms that agents can use to reason about their anti-goals.

7. Cunha, Rafhael R., Jomi F. Hübner, and Maiquel de Brito. Using Institutional Purposes to Enhance Openness of Multi-Agent Systems. **International Conference**

on Practical Applications of Agents and Multi-Agent Systems. PAAMS 2022. Springer, 2022. p. 88-99

This article discusses the disadvantages and advantages that the purpose model provides for the development of a MAS from the point of view of the programmer of the system.

Submissions still awaiting response:

1. Cunha, Rafhael R., Jomi F. Hübner, and Maiquel de Brito. Helping agents to enter and exploit artificial institutions. **Autonomous Agents and Multi-Agent Systems - JAAMAS**.

   This article discusses the disadvantages and advantages of using the purpose model for the development of a MAS from the point of view of the agent.

REFERENCES

ABREU, Jefferson Viana Fonseca. AgentBotSpotter: a multi-agent system for online Twitter bot detection, 2021.

ABREU NETTO, Manoel Teixeira de. **Um Framework Baseado em Padrões para a Construção de Sistemas Multi-Agentes Auto-Organizáveis**. 2010. PhD thesis – PUC-Rio.

ALDEWERELD, Huib; ÁLVAREZ-NAPAGAO, Sergio; DIGNUM, Frank; VÁZQUEZ-SALCEDA, Javier. Making norms concrete. In: CITESEER. PROCEEDINGS of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. [S.l.: s.n.], 2010. P. 807–814.

ALDEWERELD, Huib; DIGNUM, Virginia. OperettA: Organization-oriented development environment. In: SPRINGER. INTERNATIONAL Workshop on Languages, Methodologies and Development Tools for Multi-Agent Systems. [S.l.: s.n.], 2010. P. 1–18.

AMARAL, Cleber Jorge; HÜBNER, Jomi Fred. Jacamo-web is on the fly: an interactive Multi-Agent System IDE. In: SPRINGER. INTERNATIONAL Workshop on Engineering Multi-Agent Systems. [S.l.: s.n.], 2019. P. 246–255.

ARANDA-CORRAL, Gonzalo A; DÍAZ, Joaquín Borrego; MARTÍN, David Solís. Iabastos: an intelligent marketplace for agricultural products. In: SPRINGER. INTERNATIONAL Conference on Practical Applications of Agents and Multi-Agent Systems. [S.l.: s.n.], 2015. P. 255–258.

ARGENTE, Estefanía et al. The role of the environment in agreement technologies. **Artificial Intelligence Review**, Springer, v. 39, n. 1, p. 21–38, 2013.

AYDEMIR, Fatma Başak; GIORGINI, Paolo; MYLOPOULOS, John. Multi-objective risk analysis with goal models. In: IEEE. 2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS). [S.l.: s.n.], 2016. P. 1–10.

BERGENTI, Federico; FRANCHI, Enrico; POGGI, Agostino. Enhancing social networks with agent and semantic web technologies. In: COLLABORATION and the semantic web: social networks, knowledge networks, and knowledge resources. [S.l.]: IGI Global, 2012. P. 83–100.

BOELLA, G; REGULATIVE, Torre van der L. Constitutive Norms in Normative Multiagent Systems. In: KR'04 Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning. [S.l.: s.n.]. P. 255–265.

BOELLA, Guido; TORRE, L. van der. Regulative and constitutive norms in normative multiagent systems. **KR**, v. 4, p. 255–265, 2004.

BOELLA, Guido; TORRE, Leendert van der. Constitutive norms in the design of normative multiagent systems. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, 3900 LNAI, p. 303–319, 2006. ISSN 03029743.

BOELLA, Guido; TORRE, Leendert van der. A logical architecture of a normative system. In: SPRINGER. INTERNATIONAL Workshop on Deontic Logic and Artificial Normative Systems. [S.l.: s.n.], 2006. P. 24–35.

BOELLA, Guido; VAN DER TORRE, Leendert. An agent oriented ontology of social reality. **Procs. of FOIS**, v. 4, p. 199–209, 2004.

BOELLA, Guido; VAN DER TORRE, Leendert; VERHAGEN, Harko. Introduction to the special issue on normative multiagent systems. **Autonomous Agents and Multi-Agent Systems**, Springer, v. 17, n. 1, p. 1–10, 2008.

BOISSIER, Olivier; BORDINI, Rafael H; HÜBNER, Jomi F; RICCI, Alessandro. Dimensions in programming multi-agent systems. **The Knowledge Engineering Review**, Cambridge University Press, v. 34, 2019.

BORDINI, Rafael H; HÜBNER, Jomi Fred; WOOLDRIDGE, Michael. **Programming multi-agent systems in AgentSpeak using Jason**. [S.l.]: John Wiley & Sons, 2007. v. 8.

BRATMAN, Michael E; ISRAEL, David J; POLLACK, Martha E. Plans and resource-bounded practical reasoning. **Computational intelligence**, Wiley Online Library, v. 4, n. 3, p. 349–355, 1988.

BRITO, Maiquel de et al. **A model of institucional reality supporting the regulation in artificial institutions**. 2016. PhD thesis – Universidade Federal de Santa Catarina.

BRITO, Maiquel de; HÜBNER, Jomi F; BORDINI, Rafael H. Programming institutional facts in multi-agent systems. In: SPRINGER. INTERNATIONAL Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems. [S.l.: s.n.], 2012. P. 158–173.

BRITO, Maiquel de; HÜBNER, Jomi F. Institutional Situatedness in Multi-Agent Systems. **wesaac**, p. 12, 2014.

CALVARESI, Davide; CALBIMONTE, Jean-Paul; DUBOSSON, Fabien; NAJJAR, Amro; SCHUMACHER, Michael. Social network chatbots for smoking cessation: agent and multi-agent frameworks. In: IEEE. 2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI). [S.l.: s.n.], 2019. P. 286–292.

CAMPOS, Jordi; SANCHEZ, Maite; AGUILAR, J.a; ESTEVA, Marc. Formalising Situatedness and Adaptation in, p. 126–139, 2009.

CARDOSO, HENRIQUE LOPES; OLIVEIRA, EUGÉNIO. Institutional Reality and Norms: Specifying and Monitoring Agent Organizations. **International Journal of Cooperative Information Systems**, v. 16, n. 01, p. 67–95, 2007. ISSN 0218-8430.

CARDOSO, Henrique Lopes; OLIVEIRA, EugENio. Institutional reality and norms: Specifying and monitoring agent organizations. **International Journal of Cooperative Information Systems**, World Scientific, v. 16, n. 01, p. 67–95, 2007.

CASSANDRAS, Christos G; LAFORTUNE, Stéphane. **Introduction to discrete event systems**. [S.l.]: Springer, 2008.

CLIFFE, Owen; DE VOS, Marina; PADGET, Julian. Specifying and reasoning about multiple institutions. In: SPRINGER. INTERNATIONAL Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems. [S.l.: s.n.], 2006. P. 67–85.

CLIFFE, Owen; VOS, Marina De; PADGET, Julian. Answer set programming for representing and reasoning about virtual institutions. In: SPRINGER. INTERNATIONAL Workshop on Computational Logic in Multi-Agent Systems. [S.l.: s.n.], 2006. P. 60–79.

CRIADO, Natalia; ARGENTE, Estefania; NORIEGA, Pablo; BOTTI, Vicent. Reasoning about constitutive norms in BDI agents. **Logic Journal of the IGPL**, OUP, v. 22, n. 1, p. 66–93, 2014.

CUNHA, Rafhael R; HÜBNER, Jomi F; BRITO, Maiquel de. Instituições em Sistemas Multiagentes a luz da Teoria da Construção da Realidade Social. **Workshop Escola de Sistemas de Agentes, seus Ambientes e Aplicações**, v. XIII, p. 71–81, 2019.

CUNHA, Rafhael R; HÜBNER, Jomi F; BRITO, Maiquel de. Supporting the Reasoning About Environmental Consequences of Institutional Actions. In: SPRINGER. INTERNATIONAL Workshop on Coordination, Organizations, Institutions, Norms, and Ethics for Governance of Multi-Agent Systems. [S.l.: s.n.], 2022. P. 134–147.

DASTANI, Mehdi; GROSSI, Davide; MEYER, John Jules Ch; TINNEMEIER, Nick. Normative multi-agent programs and their logics. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, 5605 LNAI, p. 16–31, 2009. ISSN 03029743.

DASTANI, Mehdi; VAN DER TORRE, Leendert; YORKE-SMITH, Neil. Monitoring interaction in organisations. In: SPRINGER. INTERNATIONAL Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems. [S.l.: s.n.], 2012. P. 17–34.

DE BRITO, Maiquel; HÜBNER, Jomi Fred; BOISSIER, Olivier. Situated artificial institutions: stability, consistency, and flexibility in the regulation of agent societies. **Autonomous Agents and Multi-Agent Systems**, Springer, v. 32, n. 2, p. 219–251, 2018.

DEMOLOMBE, Robert. Relationships between actions performed by institutional agents, human agents or software agents. In: SPRINGER. INTERNATIONAL Conference on Deontic Logic in Computer Science. [S.l.: s.n.], 2010. P. 259–273.

DIGNUM, V; ALDEWERELD, H; DIGNUM, F. On the engineering of multi agent organizations. In: PROCEEDINGS of the 12th International Workshop on Agent-Oriented Software Engineering. [S.l.: s.n.], 2011. P. 53–65.

ESTEVA, Marc; RODRIGUEZ-AGUILAR, Juan A; ARCOS, J LL; SIERRA, Carles; NORIEGA, Pablo; ROSELL, Bruno; CRUZ, David de la. Electronic institutions development environment. In: PROCEEDINGS of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers. [S.l.: s.n.], 2008. P. 1657–1658.

FARJAM, Mike; KIRCHKAMP, Oliver. Bubbles in hybrid markets: How expectations about algorithmic trading affect human trading. **Journal of Economic Behavior & Organization**, Elsevier, v. 146, p. 248–269, 2018.

FELICÍSSIMO, Carolina; LUCENA, Carlos; BRIOT, Jean-Pierre; CHOREN, Ricardo. An approach for contextual regulations in open MAS. In: CITESEER. PROCEEDINGS of the 8th International Bi-Conference Workshop on Agent-oriented Information Systems (AOIS) at AAMAS. [S.l.: s.n.], 2006. v. 6, p. 25–32.

FIKES, Richard; FARQUHAR, Adam. Distributed repositories of highly expressive reusable ontologies. **IEEE Intelligent Systems and their Applications**, IEEE, v. 14, n. 2, p. 73–79, 1999.

FORNARA, Nicoletta. Specifying and monitoring obligations in open multiagent systems using semantic web technology. In: SEMANTIC agent systems. [S.l.]: Springer, 2011. P. 25–45.

FORNARA, Nicoletta; COLOMBETTI, Marco. Ontology and time evolution of obligations and prohibitions using semantic web technology. In: SPRINGER. INTERNATIONAL Workshop on Declarative Agent Languages and Technologies. [S.l.: s.n.], 2009. P. 101–118.

FORNARA, Nicoletta; COLOMBETTI, Marco. Representation and monitoring of commitments and norms using OWL. **AI communications**, IOS Press, v. 23, n. 4, p. 341–356, 2010.

FORNARA, Nicoletta; TAMPITSIKAS, Charalampos. Using OWL Artificial Institutions for dynamically creating Open Spaces of Interaction. In: AT. [S.l.: s.n.], 2012. P. 281–295.

FORNARA, Nicoletta; VIGANO, Francesco; VERDICCHIO, Mario; COLOMBETTI, Marco. Artificial institutions: a model of institutional reality for open multiagent systems. **Artificial intelligence and Law**, Springer, v. 16, n. 1, p. 89–105, 2008.

FORNARA, Nicoletta; VIGANÒ, Francesco; COLOMBETTI, Macro. Agent communication and institutional reality. In: SPRINGER. INTERNATIONAL Workshop on Agent Communication. [S.l.: s.n.], 2004. P. 1–17.

GEIGER, R Stuart. The lives of bots. **arXiv preprint arXiv:1810.09590**, 2018.

GEIGER, R Stuart. The social roles of bots and assisted editing programs. In: PROCEEDINGS of the 5th International Symposium on Wikis and Open Collaboration. [S.l.: s.n.], 2009. P. 1–2.

GONZÁLEZ-BRIONES, Alfonso; DE LA PRIETA, Fernando; MOHAMAD, Mohd Saberi; OMATU, Sigeru; CORCHADO, Juan M. Multi-agent systems applications in energy optimization problems: A state-of-the-art review. **Energies**, MDPI, v. 11, n. 8, p. 1928, 2018.

GOVERNATORI, Guido; DUMAS, Marlon; TER HOFSTEDE, Arthur HM; OAKS, Phillipa. A formal approach to protocols and strategies for (legal) negotiation. In: PROCEEDINGS of the 8th international conference on Artificial intelligence and law. [S.l.: s.n.], 2001. P. 168–177.

GROSSI, Davide; ALDEWERELD, Huib; VÁZQUEZ-SALCEDA, Javier; DIGNUM, Frank. Ontological aspects of the implementation of norms in agent-based electronic institutions. **Computational and Mathematical Organization Theory**, v. 12, 2-3 SPEC. ISS., p. 251–275, 2006. ISSN 1381298X.

GUARINO, Nicola. **Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy**. [S.l.]: IOS press, 1998. v. 46.

HINDRIKS, Frank. Deconstructing Searle's Making the Social World. **Philosophy of the Social Sciences**, SAGE Publications Sage CA: Los Angeles, CA, v. 45, n. 3, p. 363–369, 2015.

HINDRIKS, Koen V; DE BOER, Frank S; VAN DER HOEK, Wiebe; MEYER, John-Jules Ch. Agent programming with declarative goals. In: SPRINGER. INTERNATIONAL Workshop on Agent Theories, Architectures, and Languages. [S.l.: s.n.], 2000. P. 228–243.

HÜBNER, Jomi F; BOISSIER, Olivier; BORDINI, Rafael H. A normative programming language for multi-agent organisations. **Annals of Mathematics and Artificial Intelligence**, Springer, v. 62, n. 1, p. 27–53, 2011.

HÜBNER, Jomi F; BOISSIER, Olivier; KITIO, Rosine; RICCI, Alessandro. Instrumenting multi-agent organisations with organisational artifacts and agents. **Autonomous agents and multi-agent systems**, Springer, v. 20, n. 3, p. 369–400, 2010.

LEMAÎTRE, Christian; EXCELENTE, Cora B. Multi-agent organization approach. In: TOLEDO. PROCEEDINGS of II Iberoamerican Workshop on DAI and MAS. [S.l.: s.n.], 1998. P. 7–16.

LUCK, M; MCBURNEY, P; PREIST, C. A Roadmap for Agent Based Computing. **AgentLink, network of excellence**, 2003.

LUCK, Michael; NORIEGA, Pablo; RODRIGUEZ-AGUILAR, Juan A; SIERRA, Carles, et al. Communicating open systems. **Artificial Intelligence**, Elsevier, v. 186, p. 38–94, 2012.

MAEDCHE, Alexander; STAAB, Steffen. Ontology learning for the semantic web. **IEEE Intelligent systems**, IEEE, v. 16, n. 2, p. 72–79, 2001.

MERABET, Ghezlane Halhoul; ESSAAIDI, Mohammed; TALEI, Hanaa; ABID, Mohamed Riduan; KHALIL, Nacer; MADKOUR, Mohcine; BENHADDOU, Driss. Applications of multi-agent systems in smart grids: A survey. In: IEEE. 2014 International conference on multimedia computing and systems (ICMCS). [S.l.: s.n.], 2014. P. 1088–1094.

MUNTANER-PERICH, Eduard; DE LA ROSA ESTEVA, Josep Lluís. Towards a formalisation of dynamic electronic institutions. In: SPRINGER. INTERNATIONAL Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems. [S.l.: s.n.], 2007. P. 97–109.

NIGAM, Vivek; LEITE, Joao. A dynamic logic programming based system for agents with declarative goals. In: SPRINGER. INTERNATIONAL Workshop on Declarative Agent Languages and Technologies. [S.l.: s.n.], 2006. P. 174–190.

OSSOWSKI, Sascha; JULIÁN INGLADA, Vicente; BAJO PÉREZ, Javier; BILLHARDT, Holger; BOTTI, Vicente; CORCHADO RODRÍGUEZ, Juan Manuel, et al. Open Issues in Open MAS: An abstract architecture proposal. Daniel Borrajo, Luis Castillo y Corchado Rodríguez, Juan M.(Eds ..., 2007.

PÉREZ-MARCOS, Javier; JIMÉNEZ-BRAVO, Diego M; DE PAZ, Juan F; VILLARRUBIA GONZÁLEZ, Gabriel; LÓPEZ, Vivian F; GIL, Ana B. Multi-agent system application for music features extraction, meta-classification and context analysis. **Knowledge and Information Systems**, Springer, v. 62, n. 1, p. 401–422, 2020.

PITT, Jeremy; MAMDANI, Abe; CHARLTON, Patricia. The open agent society and its enemies: a position statement and research programme. **Telematics and Informatics**, Elsevier, v. 18, n. 1, p. 67–87, 2001.

PIUNTI, Michele; BOISSIER, Olivier; HÜBNER, Jomi F; RICCI, Alessandro. Embodied organizations: a unifying perspective in programming agents, organizations and environments. **COIN10@ MALLOW**, Citeseer, p. 98–114, 2010.

RICCI, Alessandro; PIUNTI, Michele; VIROLI, Mirko. Environment programming in multi-agent systems: an artifact-based perspective. **Autonomous Agents and Multi-Agent Systems**, Springer, v. 23, n. 2, p. 158–192, 2011.

RIEMSDIJK, Birna van; HOEK, Wiebe van der; MEYER, John-Jules Ch. Agent programming in Dribble: from beliefs to goals using plans. In: PROCEEDINGS of the second international joint conference on Autonomous agents and multiagent systems. [S.l.: s.n.], 2003. P. 393–400.

ROCHA COSTA, Antônio Carlos da. Proposal for a notion of modularity in multiagent systems. In: PRE-PROCEEDINGS of Engineering Multi-Agent Systems Workshop at AAMAS, Paris. [S.l.: s.n.], 2014. P. 21–40.

RODRIGUEZ-AGUILAR, Juan A.; SIERRA, Carles; ARCOS, Josep Ll; LOPEZ-SANCHEZ, Maite; RODRIGUEZ, Inmaculada. Towards next generation coordination infrastructures. **Knowledge Engineering Review**, v. 30, n. 4, p. 435–453, 2015. ISSN 14698005.

RUSSELL, Peter Norvig. Artificial Intelligence: A Modern Approach by Stuart. **Russell and Peter Norvig contributing writers, Ernest Davis...[et al.]**, 2010.

SARDIS, Manolis; VOUROS, George. Electronic institutions infrastructure for e-chartering. In: SPRINGER. INTERNATIONAL Workshop on Engineering Societies in the Agents World. [S.l.: s.n.], 2007. P. 90–107.

SEARLE, John. **Making the social world: The structure of human civilization**. [S.l.]: Oxford University Press, 2010.

SEARLE, John R. **The construction of social reality**. [S.l.]: Simon and Schuster, 1995.

SMIT, JP; BUEKENS, Filip; DU PLESSIS, Stan. Developing the incentivized action view of institutional reality. **Synthese**, Springer, v. 191, n. 8, p. 1813–1830, 2014.

TESTA, Italo. Dewey's social ontology: A pragmatist alternative to Searle's approach to social reality. **International Journal of Philosophical Studies**, Taylor & Francis, v. 25, n. 1, p. 40–62, 2017.

THÉVIN, Lauren; BADEIG, Fabien; DUGDALE, Julie; BOISSIER, Olivier; GARBAY, Catherine. Un Système Multi-Agent normatif hybride pour l'interaction mixte: application à la gestion de crises. **Revue des Sciences et Technologies de**

l'Information-Série RIA: Revue d'Intelligence Artificielle, v. 29, n. 3-4, p. 453–482, 2015.

VÁZQUEZ-SALCEDA, Javier; ALDEWERELD, Huib; GROSSI, Davide; DIGNUM, Frank. From human regulations to regulated software agents' behavior. **Artificial Intelligence and Law**, Springer, v. 16, n. 1, p. 73–87, 2008.

VIGANÒ, Francesco. A Framework for Model Checking Institutions. **Model Checking and Artificial Intelligence**, p. 129–145, 2007.

VIGANÒ, Francesco; COLOMBETTI, Marco. Model Checking Norms and Sanctions in Institutions. n. 2, p. 316–329, 2008.

WEYNS, Danny; OMICINI, Andrea; ODELL, James. Environment as a first class abstraction in multiagent systems. **Autonomous agents and multi-agent systems**, Springer, v. 14, n. 1, p. 5–30, 2007.

WHITWORTH, Brian. The social requirements of technical systems. In: VIRTUAL Communities: Concepts, Methodologies, Tools and Applications. [S.l.]: IGI Global, 2011. P. 1461–1481.

WINIKOFF, Michael; PADGHAM, Lin; HARLAND, James; THANGARAJAH, John. Declarative and procedural goals in intelligent agent systems. In: MORGAN KAUFMAN. INTERNATIONAL Conference on Principles of Knowledge Representation and Reasoning. [S.l.: s.n.], 2002.

WOOLDRIDGE, Michael. **An introduction to multiagent systems**. [S.l.]: John Wiley & Sons, 2009.

ZAMBONELLI, Franco; JENNINGS, Nicholas R; WOOLDRIDGE, Michael. Organisational abstractions for the analysis and design of multi-agent systems. In: SPRINGER. INTERNATIONAL Workshop on Agent-Oriented Software Engineering. [S.l.: s.n.], 2000. P. 235–251.

APPENDIX A – IMPLEMENTATION OF THE ALGORITHMS IN JASON

Algorithm A.1 is essentially a plan that agents can execute to discover which institutional actions (`Actions` argument) can be performed to achieve a goal (`Goal` argument). The algorithm searches in the institutional specification (i) which states pointed out by the purposes match the agent's goal, (ii) which status functions are associated with these purposes and (iii) which concrete actions can constitute these status functions, returning a list of actions.

```
1  +!alg1(Goal, Actions)
2    <-  .queue.create(Actions);
3      getPurposesOfState(Goal, Purposes);
4      for(.member(Purpose,Purposes)){
5          getStatusFunctionsFromPurpose(Purpose, NameSF);
6          for(.member(StatusFunction, NameSF)){
7            ?constitutive_rule(Action,StatusFunction,_,_);
8            .queue.add(Actions,Action);
9          }
10     }.
11
12 +!alg1(Goal, []).
```

Listing A.1 – Implementation of Algorithm 1

Algorithm A.2 is essentially a plan that agents can execute to discover which states (`States` argument) can be reached from the execution of an action (`Action` argument). The algorithm searches in the institutional specification (i) which status functions can be assigned to the action, (ii) which purposes are associated with these status functions and (iii) which states are pointed by these purposes, returning a list of states.

```
1  +!alg2(Action, States) : constitutive_rule(Action,SF,_,_)
2    <- .queue.create(States);
3      getPurposesOfStatusFunctions(SF, Purposes);
4      for(.member(Purpose,Purposes)){
5       getPredicatesOfStatesRelatedToPurpose(Purpose,Predicates);
6       for(.member(Predicate,Predicates)){
7         .queue.add(States, Predicate);
8       }
9      }.
10
11 +!alg2(Action, []).
```

Listing A.2 – Implementation of Algorithm 2

Algorithm A.3 is essentially a plan that agents can execute to discover whether the institutional action to be taken (`Action` argument) produces changes in the environment that

match the agent's goal (`Goal` argument). The algorithm uses `alg2` to search for the states that can be reached if `Action` is executed and checks if `Goal` is in that list of states, returning true or false.

```
1 +!alg3(Goal, Action, R)
2   <- !alg2(Action,States);
3      R = .member(Goal,States).
```

Listing A.3 – Implementation of Algorithm 3

Algorithm A.4 is essentially a plan that agents can execute to discover whether the institutional action to be taken (`Action` argument) produces changes in the environment that match the agent's anti-goal (`AntiGoal` argument). The algorithm uses `alg2` to search for the states that can be reached if `Action` is executed and checks if `AntiGoal` is in that list of states, returning true or false.

```
1 +!alg4(AntiGoal, Action, R)
2   <- !alg2(Action,States);
3      R = .member(AntiGoal,States).
```

Listing A.4 – Implementation of Algorithm 4

APPENDIX B – APPLICATION EXAMPLE 1: BOOK TRADE

B.1 CASE 0 - DEFAULT SETUP

The default Gradle code for all cases in example 1 is shown in Listing B.1. The implementation lines within the dependencies code snippet add dependencies for the JaCaMO framework and the Purpose Model, respectively. These lines direct Gradle to fetch files from these projects independently and add them to the project being coded.

```
1  defaultTasks 'jar'
2
3  apply plugin: 'java'
4  apply plugin: 'eclipse'
5
6  java {
7      toolchain {
8          languageVersion = JavaLanguageVersion.of(15)
9      }
10 }
11
12 repositories {
13     mavenCentral()
14     maven { url "https://raw.githubusercontent.com/jacamo-lang/
           mvn-repo/master" }
15     maven { url "https://repo.gradle.org/gradle/libs-releases-
           local" }
16     maven { url 'https://jitpack.io' }
17 }
18
19 dependencies {
20     implementation ('org.jacamo:jacamo:1.1.1-SNAPSHOT')
21     implementation ('com.github.rafhaelrc:PurposeModel:1.8')
22 }
23
24 sourceSets {
25     main {
26         java {
27             srcDir 'src'
28         }
29         resources {
30             srcDir 'src/resources'
31         }
32     }
```

```
33 }
34
35 jar {
36     baseName 'bookTrade'
37     manifest {
38         attributes 'Main-Class': 'Main',
39                    'Specification-Title': 'Book Trade',
40                    'Specification-Version': project.version,
41                    'Implementation-Version': new Date().toString
                         ()
42     }
43 }
44
45 clean {
46     delete 'bin'
47     delete 'build'
48     delete 'log'
49 }
```

Listing B.1 – Default Gradle file for all book Trade example cases

B.2 CASE 1 - SIMPLE MAS

The artifact code shown in Listing B.2 has implemented the functions `deliver_paper_note` and `deliver_book` which are used by *Bob* to deliver the paper note to *Tom* and by *Tom* to deliver the book to *Bob*, respectively. These functions add beliefs about actions performed within the system, resulting in Tom's reaction and Bob's goal achievement.

```
1  package tools;
2
3  import cartago.Artifact;
4  import cartago.OPERATION;
5  import java.io.IOException;
6  import java.util.logging.Logger;
7
8
9  public class ElectronicMachine extends Artifact{
10   private Logger logger = Logger.getLogger(ElectronicMachine.
        class.getName());
11
12   @OPERATION
13   public void init()  {
14     defineObsProperty("task", 0);
```

```
15
16    }
17
18    @OPERATION
19    public void deliver_paper_note(String seller) {
20      defineObsProperty("deliver_paper_note", "bob");
21      // implement the action..
22    }
23
24    @OPERATION
25    public void deliver_book(String buyer) {
26      defineObsProperty("deliverBook");
27      // implement the action..
28    }
29  }
```

Listing B.2 – Artifact - Example 1 and Case 1

The JCM code shown in Listing B.3 specifies the agents *Bob* and *Tom*, their focus on the artifact `machine`, the workspace `wsp` and the artifact `machine` that is available in that space.

```
1  mas book_trade {
2
3    agent bob {
4        focus: machine
5      }
6      agent tom {
7        focus: machine
8      }
9      workspace wsp {
10       artifact machine: tools.ElectronicMachine()
11   }
12 }
```

Listing B.3 – JCM - Example 1 and case 1

Bob's code is shown in Listing B.4. This code contains the plan that *Bob* should execute to achieve its `holdBook` goal.

```
1  bookOwner(tom).  // agent belief
2
3  !holdBook.  // agent goal
4
5  +!holdBook : bookOwner(Seller)
6      <- deliver_paper_note(Seller);
```

```
7         .wait(deliverBook).
```

Listing B.4 – Bob's Program - Example 1 and Case 1

Tom's code is shown in Listing B.5. This code contains the plan that *Tom* should execute to deliver the book to the agent that performed the `deliver_paper_note` action.

```
1 +deliver_paper_note(Buyer) <- deliver_book(Buyer).
```

Listing B.5 – Tom's Program - Example 1 and Case 1

B.3 CASE 2 - MAS WITH STATUS FUNCTIONS

The artifact code shown in Listing B.6 has implemented the same functions as in Listing B.2. The difference is in the beliefs that these functions add to the system. While in Listing B.2 the beliefs are related to the actions that are performed, in Listing B.6 the beliefs are related to the status functions that have been constituted.

```
1  package tools;
2
3  import cartago.Artifact;
4  import cartago.OPERATION;
5  import java.io.IOException;
6  import java.util.logging.Logger;
7
8
9  public class ElectronicMachine extends Artifact{
10   private Logger logger = Logger.getLogger(ElectronicMachine.
       class.getName());
11
12   @OPERATION
13   public void init()  {
14     defineObsProperty("task", 0);
15   }
16
17   @OPERATION
18   public void deliver_paper_note(String seller) {
19     defineObsProperty("payment", "bob");
20     // implement the action..
21   }
22
23   @OPERATION
24   public void deliver_book(String buyer) {
25     defineObsProperty("deliverBook");
26     // implement the action..
```

```
27   }
28 }
```

Listing B.6 – Artifact - Example 1 and Case 2

The JCM code shown in Listing B.7 is similar to Listing B.3. The difference is the addition of the agents' focus on the `inst_test_art` artifact and the addition of the institutional specification associated with the `wsp` workspace.

```
1  mas book_trade {
2
3    agent bob {
4        focus: machine, inst_test.inst_test_art
5      }
6      agent tom {
7        focus: machine, inst_test.inst_test_art //focus on the
             institutional artifact
8      }
9      workspace wsp {
10     artifact machine: tools.ElectronicMachine()
11   }
12   institution inst_test: src/resources/constitutive-specification
        .sai {
13     workspaces: wsp
14   }
15 }
```

Listing B.7 – JCM - Example 1 and Case 2

The institutional specification code is shown in Listing B.8. It contains the status function and constitutive rule used in this example.

```
1 institution_id : bhInst.
2
3 status_functions:
4
5 events: payment(X).
6
7 constitutive_rules:
8
9 1: deliver_paper_note(X) count-as payment(X).
```

Listing B.8 – Institutional specification Example 1 and Case 2

Bob's code is shown in Listing B.9. The difference between Listing B.4 and this Listing is the action that *Bob* should perform. While in the first the action is hard-coded, *Bob* consults

the institutional specification to identify which concrete action can constitute payment status function in the second.

```
1 bookOwner(tom).  // agent belief
2 statusFunction(payment(tom)).
3
4 !holdBook.  // agent goal
5
6 +!holdBook : statusFunction(SF) & bookOwner(Seller)
7       <- ?constitutive_rule(Action,SF,_,_);
8          Action;
9          .wait(deliverBook).
```

Listing B.9 – Bob's Program - Example 1 and Case 2

Tom's code is shown in Listing B.10. This code contains the plan that *Tom* should execute to deliver the book to the agent that performed the deliver_paper_note action. The difference between Listing B.5 and this Listing is in the trigger that triggers Tom's reaction. While in the first the reaction is activated by a specific action, in the second the reaction occurs because the status function payment has been constituted.

```
1 +payment(Buyer) <-  deliver_book(Buyer).
```

Listing B.10 – Tom's Program - Example 1 and Case 2

B.4 CASE 3 - MAS WITH STATUS FUNCTIONS AND NORMS

The normative specification shown in Listing B.11 describes the norm that obliges *Tom* to deliver the book to the agent that performed the payment constitution when such constitution is performed.

```
1 scope main {
2
3    norm n1: payment(X)
4       -> obligation(tom,n1, deliver_book(X), `now`+`15 seconds`)
           .
5 }
```

Listing B.11 – Norm specification - Example 1 and Case 3

The JCM code shown in Listing B.12 is similar to the specification shown in Listing B.7. The difference is in agent *Tom*. In this specification it focuses on the artifact nplArt and has the goal setup_sai that connects it to the normative specification.

```
1 mas book_trade {
2
3   agent bob {
```

```
4         focus: machine, inst_test.inst_test_art
5       }
6       agent tom {
7         focus: machine, wsp_npl.nplArt,
8         inst_test.inst_test_art //focus on the institutional
              artifact
9
10        goals: setup_sai // connect norms and institution
11      }
12
13      workspace wsp_npl {
14            artifact nplArt: sai.bridges.jacamo.NormativeBoardSai
15      }
16
17      workspace wsp {
18      artifact machine: tools.ElectronicMachine()
19    }
20    institution inst_test: src/resources/constitutive-specification
          .sai {
21      workspaces: wsp
22    }
23  }
```

Listing B.12 – JCM - Example 1 and Case 3

Tom's code is shown in Listing B.13. This code contains the plan that *Tom* should perform when the norm related to it is activated and some plans that connect it to the normative specification.

```
1  +obligation(Ag,R,Goal,Deadline) //the agent perceives the
       obligation following the NPL notation
2      : .my_name(Ag)
3      <- println("I am obliged to see to me that the state ",Goal,"
          holds");
4          Goal.
5
6
7  //connect norms to institution
8  +!setup_sai: focusing(ArtSai,inst_test_art,_,_,inst_test,_) &
       focusing(NplArt,nplArt,_,_,wsp_npl,_) <-
9      getSaiEngine(SE)[artifact_id(ArtSai)];
10     setInstitution(SE)[artifact_id(NplArt)];
11     load("src/org/norms.npl").
12
```

```
13 +!setup_sai<-
14     .wait(focusing(A,_,_,B,inst_test,_)&focusing(ArtSai,
          inst_test_art,_,_,inst_test,_) & focusing(NplArt,nplArt,_,_
          ,wsp_npl,_));
15     !setup_sai.
16
17
18 { include("$jacamoJar/templates/common-cartago.asl") }
19 { include("$jacamoJar/templates/common-moise.asl") }
```

Listing B.13 – Tom's Program - Example 1 and Case 3

B.5 CASE 4 - MAS WITH INSTITUTION, NORMS AND PURPOSES

The JCM code shown in Listing B.14 is similar to the specification shown in Listing B.12 and B.7. The difference is the addition of the `onto` artifact that serves to add the ontology that implements the purpose model.

```
1  mas book_trade {
2
3    agent bob {
4        focus: machine, inst_test.inst_test_art
5      }
6      agent tom {
7        focus: machine, wsp_npl.nplArt,
8        inst_test.inst_test_art //focus on the institutional
             artifact
9
10       goals: setup_sai // connect norms and institution
11     }
12
13     workspace wsp_npl {
14           artifact nplArt: sai.bridges.jacamo.NormativeBoardSai
15     }
16
17     workspace wsp {
18         artifact onto: mas.OntologyArtifact("resources/
              book_trade_ontology.owl")
19     artifact machine: tools.ElectronicMachine()
20   }
21   institution inst_test: src/resources/constitutive-specification
        .sai {
22     workspaces: wsp
```

```
23    }
24 }
```

Listing B.14 – JCM - Example 1 and Case 4

The ontology's XML code is shown in Listing B.15. This code contains the classes (e.g., `status function`, `purpose`, etc.), the connections between the classes (e.g., `hasPurpose`, `hasConsequence`, etc.) and the individuals representing the elements of this scenario (e.g., `payment(x)`, `book_trade_purpose`, etc.).

```
1  <?xml version="1.0"?>
2  <Ontology xmlns="http://www.w3.org/2002/07/owl#"
3       xml:base="http://www.semanticweb.org/rafhaelrc/ontologies
           /2020/8/semantic_sf_5"
4       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5       xmlns:xml="http://www.w3.org/XML/1998/namespace"
6       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
7       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
8       ontologyIRI="http://www.semanticweb.org/rafhaelrc/ontologies
           /2020/8/semantic_sf_5">
9      <Prefix name="" IRI="http://www.semanticweb.org/rafhaelrc/
          ontologies/2020/8/semantic_sf_5"/>
10     <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
11     <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-
          syntax-ns#"/>
12     <Prefix name="sf5" IRI="http://www.semanticweb.org/rafhaelrc/
          ontologies/2020/8/semantic_sf_5"/>
13     <Prefix name="xml" IRI="http://www.w3.org/XML/1998/namespace"
          />
14     <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
15     <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema
          #"/>
16     <Prefix name="swrl" IRI="http://www.w3.org/2003/11/swrl#"/>
17     <Prefix name="swrlb" IRI="http://www.w3.org/2003/11/swrlb#"/>
18     <Declaration>
19         <Class IRI="#Parameter"/>
20     </Declaration>
21     <Declaration>
22         <Class IRI="#Predicate"/>
23     </Declaration>
24     <Declaration>
25         <Class IRI="#Purpose"/>
26     </Declaration>
27     <Declaration>
```

```
28          <Class IRI="#State"/>
29      </Declaration>
30      <Declaration>
31          <Class IRI="#Status-Function"/>
32      </Declaration>
33      <Declaration>
34          <ObjectProperty IRI="#hasConsequence"/>
35      </Declaration>
36      <Declaration>
37          <ObjectProperty IRI="#hasParameter"/>
38      </Declaration>
39      <Declaration>
40          <ObjectProperty IRI="#hasPredicate"/>
41      </Declaration>
42      <Declaration>
43          <ObjectProperty IRI="#hasPurpose"/>
44      </Declaration>
45      <Declaration>
46          <ObjectProperty IRI="#hasStatus"/>
47      </Declaration>
48      <Declaration>
49          <ObjectProperty IRI="#isConsequenceOf"/>
50      </Declaration>
51      <Declaration>
52          <ObjectProperty IRI="#isParameterOf"/>
53      </Declaration>
54      <Declaration>
55          <ObjectProperty IRI="#isPredicateOf"/>
56      </Declaration>
57      <Declaration>
58          <ObjectProperty IRI="#isPurposeOf"/>
59      </Declaration>
60      <Declaration>
61          <ObjectProperty IRI="#isStatusOf"/>
62      </Declaration>
63      <Declaration>
64          <DataProperty IRI="#hasPrice"/>
65      </Declaration>
66      <Declaration>
67          <DataProperty IRI="#hasTitle"/>
68      </Declaration>
69      <Declaration>
```

```
70          <DataProperty IRI="#hasValue"/>
71      </Declaration>
72      <Declaration>
73          <NamedIndividual IRI="#book_trade_purpose"/>
74      </Declaration>
75      <Declaration>
76          <NamedIndividual IRI="#holdBook"/>
77      </Declaration>
78      <Declaration>
79          <NamedIndividual IRI="#state_book"/>
80      </Declaration>
81      <Declaration>
82          <NamedIndividual IRI="#payment(X)"/>
83      </Declaration>
84      <Declaration>
85          <AnnotationProperty abbreviatedIRI="owl:position"/>
86      </Declaration>
87      <ClassAssertion>
88          <Class IRI="#Purpose"/>
89          <NamedIndividual IRI="#book_trade_purpose"/>
90      </ClassAssertion>
91      <ClassAssertion>
92          <Class IRI="#Predicate"/>
93          <NamedIndividual IRI="#holdBook"/>
94      </ClassAssertion>
95      <ClassAssertion>
96          <Class IRI="#State"/>
97          <NamedIndividual IRI="#state_book"/>
98      </ClassAssertion>
99      <ClassAssertion>
100         <Class IRI="#Status-Function"/>
101         <NamedIndividual IRI="#payment(X)"/>
102     </ClassAssertion>
103     <ObjectPropertyAssertion>
104         <ObjectProperty IRI="#hasConsequence"/>
105         <NamedIndividual IRI="#book_trade_purpose"/>
106         <NamedIndividual IRI="#state_book"/>
107     </ObjectPropertyAssertion>
108     <ObjectPropertyAssertion>
109         <ObjectProperty IRI="#hasPredicate"/>
110         <NamedIndividual IRI="#state_book"/>
111         <NamedIndividual IRI="#holdBook"/>
```

```
112     </ObjectPropertyAssertion>
113     <ObjectPropertyAssertion>
114         <ObjectProperty IRI="#hasPurpose"/>
115         <NamedIndividual IRI="#payment(X)"/>
116         <NamedIndividual IRI="#book_trade_purpose"/>
117     </ObjectPropertyAssertion>
118     <InverseObjectProperties>
119         <ObjectProperty IRI="#hasConsequence"/>
120         <ObjectProperty IRI="#isConsequenceOf"/>
121     </InverseObjectProperties>
122     <InverseObjectProperties>
123         <ObjectProperty IRI="#hasParameter"/>
124         <ObjectProperty IRI="#isParameterOf"/>
125     </InverseObjectProperties>
126     <InverseObjectProperties>
127         <ObjectProperty IRI="#hasPredicate"/>
128         <ObjectProperty IRI="#isPredicateOf"/>
129     </InverseObjectProperties>
130     <InverseObjectProperties>
131         <ObjectProperty IRI="#hasPurpose"/>
132         <ObjectProperty IRI="#isPurposeOf"/>
133     </InverseObjectProperties>
134     <InverseObjectProperties>
135         <ObjectProperty IRI="#hasStatus"/>
136         <ObjectProperty IRI="#isStatusOf"/>
137     </InverseObjectProperties>
138     <IrreflexiveObjectProperty>
139         <ObjectProperty IRI="#hasStatus"/>
140     </IrreflexiveObjectProperty>
141     <IrreflexiveObjectProperty>
142         <ObjectProperty IRI="#isStatusOf"/>
143     </IrreflexiveObjectProperty>
144     <ObjectPropertyDomain>
145         <ObjectProperty IRI="#hasConsequence"/>
146         <Class IRI="#Purpose"/>
147     </ObjectPropertyDomain>
148     <ObjectPropertyDomain>
149         <ObjectProperty IRI="#hasPredicate"/>
150         <Class IRI="#State"/>
151     </ObjectPropertyDomain>
152     <ObjectPropertyDomain>
153         <ObjectProperty IRI="#hasPurpose"/>
```

```
154         <Class IRI="#Status-Function"/>
155     </ObjectPropertyDomain>
156     <ObjectPropertyDomain>
157         <ObjectProperty IRI="#hasStatus"/>
158         <Class IRI="#Status-Function"/>
159     </ObjectPropertyDomain>
160     <ObjectPropertyDomain>
161         <ObjectProperty IRI="#isConsequenceOf"/>
162         <Class IRI="#State"/>
163     </ObjectPropertyDomain>
164     <ObjectPropertyDomain>
165         <ObjectProperty IRI="#isPredicateOf"/>
166         <Class IRI="#Predicate"/>
167     </ObjectPropertyDomain>
168     <ObjectPropertyRange>
169         <ObjectProperty IRI="#hasConsequence"/>
170         <Class IRI="#State"/>
171     </ObjectPropertyRange>
172     <ObjectPropertyRange>
173         <ObjectProperty IRI="#hasPredicate"/>
174         <Class IRI="#Predicate"/>
175     </ObjectPropertyRange>
176     <ObjectPropertyRange>
177         <ObjectProperty IRI="#hasPurpose"/>
178         <Class IRI="#Purpose"/>
179     </ObjectPropertyRange>
180     <ObjectPropertyRange>
181         <ObjectProperty IRI="#hasStatus"/>
182         <Class IRI="#Status-Function"/>
183     </ObjectPropertyRange>
184     <ObjectPropertyRange>
185         <ObjectProperty IRI="#isConsequenceOf"/>
186         <Class IRI="#Purpose"/>
187     </ObjectPropertyRange>
188     <ObjectPropertyRange>
189         <ObjectProperty IRI="#isPredicateOf"/>
190         <Class IRI="#State"/>
191     </ObjectPropertyRange>
192     <DataPropertyRange>
193         <DataProperty IRI="#hasPrice"/>
194         <Datatype abbreviatedIRI="xsd:double"/>
195     </DataPropertyRange>
```

```
196     <DataPropertyRange>
197         <DataProperty IRI="#hasTitle"/>
198         <Datatype abbreviatedIRI="xsd:string"/>
199     </DataPropertyRange>
200     <DataPropertyRange>
201         <DataProperty IRI="#hasValue"/>
202         <Datatype abbreviatedIRI="xsd:double"/>
203     </DataPropertyRange>
204 </Ontology>
```

Listing B.15 – OWL Ontology - Book Trade

Bob's code is shown in Listing B.16. This code contains the plan that *Bob* should execute to achieve its `holdBook` goal. In this plan, *Bob* uses `alg1` to find available institutional actions that can help it achieve its goal. The difference between this implementation and the previous ones is that *Bob* can query this information at runtime and does not need to previously know neither the action nor the status function that should be constituted.

```
1 +!holdBook :  <- !alg1(holdBook, Actions);
2                  .queue.head(Actions, Action);
3                  Action.
```

Listing B.16 – Bob's Program - Example 1 and Case 4

It is important to be clear that the Listings used in the previous examples that are not replaced in the following examples remain valid and therefore are not repeated. The complete code is available in a git repository at `https://github.com/rafhaelrc/PurposeModel/tree/main/demo/book_trade`.

APPENDIX C – APPLICATION EXAMPLE 2: CONQUER TERRITORY

C.1 CASE 1 - MAS WITH IMPLICIT STATUS FUNCTIONS AND PURPOSES

The artifact code shown in Listing C.1 has implemented the functions `broadcast_a_message` and `posting_on_a_webservice`.

```
1  package tools;
2
3  import cartago.Artifact;
4  import cartago.OPERATION;
5  import java.io.IOException;
6  import java.util.logging.Logger;
7
8
9  public class ElectronicMachine extends Artifact{
10   private Logger logger = Logger.getLogger(ElectronicMachine.
       class.getName());
11
12   @OPERATION
13   public void init()  {
14     defineObsProperty("task", 0);
15   }
16
17   @OPERATION
18   public void broadcast_a_message() {
19     // implement the action..
20   }
21
22   @OPERATION
23   public void posting_on_a_webservice() {
24     // implement the action..
25   }
26 }
```

Listing C.1 – Artifact - Example 2 and Case 1

The JCM code shown in Listing C.2 specifies the agent *Bob*, its focus on the artifact `machine`, the workspace `wsp` and the artifact `machine` that is available in that space.

```
1  mas conquer_territory {
2
3      agent bob {
4        focus: machine
5      }
```

```
6       workspace wsp {
7         artifact machine: tools.ElectronicMachine()
8       }
9   }
```

Listing C.2 – JCM - Example 2 and case 1

Bob's code is shown in Listing B.4. This code contains the plan that *Bob* should execute to achieve its `territory_conquered` goal. Furthermore, this code has specified through beliefs the institutional and environmental consequences of performing `broadcast_a_message`. Based on this information, *Bob* checks whether the action to be performed has inflicted its anti-goal. If is the case, *Bob* avoids the action. If not, *Bob* executes it.

```
1  anti_goal(soldier_killed_from_allied_base).
2  cause(forcing_an_attack, territory_conquered).
3  cause(forcing_an_attack, soldier_killed_from_allied_base).
4  constitutive_rule(broadcast_a_message,forcing_an_attack,_,_).
5
6  !territory_conquered. // agent goal
7
8  +!territory_conquered : cause (SF,territory_conquered)
9       <- if (anti_goal(AG) & cause(SF,AG)){
10              .fail;
11          } else {
12              ?constitutive_rule(Action,SF,_,_);
13              Action;
14          }.
15
16 { include("$jacamoJar/templates/common-cartago.asl") }
17 { include("$jacamoJar/templates/common-moise.asl") }
```

Listing C.3 – Bob's Program - Example 2 and Case 1

C.2 CASE 2 - MAS WITH EXPLICIT STATUS FUNCTIONS AND PURPOSES

The JCM code shown in Listing C.4 is similar to Listing C.2. The difference is the addition of the agent's focus on the `inst_test_art` artifact, the addition of the institutional specification associated with the `wsp` workspace, and the inclusion of the `onto` artifact that serves to add the ontology that implements the purpose model.

```
1  mas conquer_territory {
2
3    agent bob {
4        focus: machine, inst_test.inst_test_art
```

```
5       }
6
7       workspace wsp {
8       artifact onto: mas.OntologyArtifact("src/resources/
            conquer_territory_ontology.owl")
9       artifact machine: tools.ElectronicMachine()
10    }
11
12    institution inst_test: src/resources/constitutive-specification
          .sai {
13      workspaces: wsp
14    }
15  }
```

Listing C.4 – JCM - Example 2 and case 2

The institutional specification code is shown in Listing C.5. It contains the status function and constitutive rule used in this example.

```
1  institution_id : bhInst.
2
3  status_functions:
4
5  events: forcing_an_attack, authorizing_an_attack.
6
7  constitutive_rules:
8
9  1: broadcast_a_message count-as forcing_an_attack.
10 2: posting_on_a_webservice count-as authorizing_an_attack.
```

Listing C.5 – Institutional specification Example 2 and Case 2

The ontology's XML code is shown in Listing C.6. This code contains the classes (e.g., status function, purpose, etc.), the connections between the classes (e.g., hasPurpose, hasConsequence, etc.) and the individuals representing the elements of this scenario (e.g., authorize_an_attack, force_attack, etc.).

```
1  <?xml version="1.0"?>
2  <Ontology xmlns="http://www.w3.org/2002/07/owl#"
3       xml:base="http://www.semanticweb.org/rafhaelrc/ontologies
             /2020/8/semantic_sf_5"
4       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5       xmlns:xml="http://www.w3.org/XML/1998/namespace"
6       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
7       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
```

```
8       ontologyIRI="http://www.semanticweb.org/rafhaelrc/ontologies
          /2020/8/semantic_sf_5">
9       <Prefix name="" IRI="http://www.semanticweb.org/rafhaelrc/
          ontologies/2020/8/semantic_sf_5"/>
10      <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
11      <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-
          syntax-ns#"/>
12      <Prefix name="sf5" IRI="http://www.semanticweb.org/rafhaelrc/
          ontologies/2020/8/semantic_sf_5"/>
13      <Prefix name="xml" IRI="http://www.w3.org/XML/1998/namespace"
          />
14      <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
15      <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema
          #"/>
16      <Prefix name="swrl" IRI="http://www.w3.org/2003/11/swrl#"/>
17      <Prefix name="swrlb" IRI="http://www.w3.org/2003/11/swrlb#"/>
18      <Declaration>
19          <Class IRI="#Parameter"/>
20      </Declaration>
21      <Declaration>
22          <Class IRI="#Predicate"/>
23      </Declaration>
24      <Declaration>
25          <Class IRI="#Purpose"/>
26      </Declaration>
27      <Declaration>
28          <Class IRI="#State"/>
29      </Declaration>
30      <Declaration>
31          <Class IRI="#Status-Function"/>
32      </Declaration>
33      <Declaration>
34          <ObjectProperty IRI="#hasConsequence"/>
35      </Declaration>
36      <Declaration>
37          <ObjectProperty IRI="#hasParameter"/>
38      </Declaration>
39      <Declaration>
40          <ObjectProperty IRI="#hasPredicate"/>
41      </Declaration>
42      <Declaration>
43          <ObjectProperty IRI="#hasPurpose"/>
```

```
44      </Declaration>
45      <Declaration>
46          <ObjectProperty IRI="#hasStatus"/>
47      </Declaration>
48      <Declaration>
49          <ObjectProperty IRI="#isConsequenceOf"/>
50      </Declaration>
51      <Declaration>
52          <ObjectProperty IRI="#isParameterOf"/>
53      </Declaration>
54      <Declaration>
55          <ObjectProperty IRI="#isPredicateOf"/>
56      </Declaration>
57      <Declaration>
58          <ObjectProperty IRI="#isPurposeOf"/>
59      </Declaration>
60      <Declaration>
61          <ObjectProperty IRI="#isStatusOf"/>
62      </Declaration>
63      <Declaration>
64          <DataProperty IRI="#hasPrice"/>
65      </Declaration>
66      <Declaration>
67          <DataProperty IRI="#hasTitle"/>
68      </Declaration>
69      <Declaration>
70          <DataProperty IRI="#hasValue"/>
71      </Declaration>
72      <Declaration>
73          <NamedIndividual IRI="#authorize_attack"/>
74      </Declaration>
75      <Declaration>
76          <NamedIndividual IRI="#authorizing_an_attack"/>
77      </Declaration>
78      <Declaration>
79          <NamedIndividual IRI="#force_attack"/>
80      </Declaration>
81      <Declaration>
82          <NamedIndividual IRI="#forcing_an_attack"/>
83      </Declaration>
84      <Declaration>
85          <NamedIndividual IRI="#soldier_killed_from_allied_base"/>
```

```
86      </Declaration>
87      <Declaration>
88          <NamedIndividual IRI="#state1"/>
89      </Declaration>
90      <Declaration>
91          <NamedIndividual IRI="#state2"/>
92      </Declaration>
93      <Declaration>
94          <NamedIndividual IRI="#territory_conquered"/>
95      </Declaration>
96      <Declaration>
97          <AnnotationProperty abbreviatedIRI="owl:position"/>
98      </Declaration>
99      <ClassAssertion>
100         <Class IRI="#Purpose"/>
101         <NamedIndividual IRI="#authorize_attack"/>
102     </ClassAssertion>
103     <ClassAssertion>
104         <Class IRI="#Status-Function"/>
105         <NamedIndividual IRI="#authorizing_an_attack"/>
106     </ClassAssertion>
107     <ClassAssertion>
108         <Class IRI="#Purpose"/>
109         <NamedIndividual IRI="#force_attack"/>
110     </ClassAssertion>
111     <ClassAssertion>
112         <Class IRI="#Status-Function"/>
113         <NamedIndividual IRI="#forcing_an_attack"/>
114     </ClassAssertion>
115     <ClassAssertion>
116         <Class IRI="#Predicate"/>
117         <NamedIndividual IRI="#soldier_killed_from_allied_base"/>
118     </ClassAssertion>
119     <ClassAssertion>
120         <Class IRI="#State"/>
121         <NamedIndividual IRI="#state1"/>
122     </ClassAssertion>
123     <ClassAssertion>
124         <Class IRI="#State"/>
125         <NamedIndividual IRI="#state2"/>
126     </ClassAssertion>
127     <ClassAssertion>
```

```
128         <Class IRI="#Predicate"/>
129         <NamedIndividual IRI="#territory_conquered"/>
130     </ClassAssertion>
131     <ObjectPropertyAssertion>
132         <ObjectProperty IRI="#hasConsequence"/>
133         <NamedIndividual IRI="#authorize_attack"/>
134         <NamedIndividual IRI="#state2"/>
135     </ObjectPropertyAssertion>
136     <ObjectPropertyAssertion>
137         <ObjectProperty IRI="#hasPurpose"/>
138         <NamedIndividual IRI="#authorizing_an_attack"/>
139         <NamedIndividual IRI="#authorize_attack"/>
140     </ObjectPropertyAssertion>
141     <ObjectPropertyAssertion>
142         <ObjectProperty IRI="#hasConsequence"/>
143         <NamedIndividual IRI="#force_attack"/>
144         <NamedIndividual IRI="#state1"/>
145     </ObjectPropertyAssertion>
146     <ObjectPropertyAssertion>
147         <ObjectProperty IRI="#hasPurpose"/>
148         <NamedIndividual IRI="#forcing_an_attack"/>
149         <NamedIndividual IRI="#force_attack"/>
150     </ObjectPropertyAssertion>
151     <ObjectPropertyAssertion>
152         <ObjectProperty IRI="#hasPredicate"/>
153         <NamedIndividual IRI="#state1"/>
154         <NamedIndividual IRI="#soldier_killed_from_allied_base"/>
155     </ObjectPropertyAssertion>
156     <ObjectPropertyAssertion>
157         <ObjectProperty IRI="#hasPredicate"/>
158         <NamedIndividual IRI="#state1"/>
159         <NamedIndividual IRI="#territory_conquered"/>
160     </ObjectPropertyAssertion>
161     <ObjectPropertyAssertion>
162         <ObjectProperty IRI="#hasPredicate"/>
163         <NamedIndividual IRI="#state2"/>
164         <NamedIndividual IRI="#territory_conquered"/>
165     </ObjectPropertyAssertion>
166     <InverseObjectProperties>
167         <ObjectProperty IRI="#hasConsequence"/>
168         <ObjectProperty IRI="#isConsequenceOf"/>
169     </InverseObjectProperties>
```

```
170     <InverseObjectProperties>
171         <ObjectProperty IRI="#hasParameter"/>
172         <ObjectProperty IRI="#isParameterOf"/>
173     </InverseObjectProperties>
174     <InverseObjectProperties>
175         <ObjectProperty IRI="#hasPredicate"/>
176         <ObjectProperty IRI="#isPredicateOf"/>
177     </InverseObjectProperties>
178     <InverseObjectProperties>
179         <ObjectProperty IRI="#hasPurpose"/>
180         <ObjectProperty IRI="#isPurposeOf"/>
181     </InverseObjectProperties>
182     <InverseObjectProperties>
183         <ObjectProperty IRI="#hasStatus"/>
184         <ObjectProperty IRI="#isStatusOf"/>
185     </InverseObjectProperties>
186     <IrreflexiveObjectProperty>
187         <ObjectProperty IRI="#hasStatus"/>
188     </IrreflexiveObjectProperty>
189     <IrreflexiveObjectProperty>
190         <ObjectProperty IRI="#isStatusOf"/>
191     </IrreflexiveObjectProperty>
192     <ObjectPropertyDomain>
193         <ObjectProperty IRI="#hasConsequence"/>
194         <Class IRI="#Purpose"/>
195     </ObjectPropertyDomain>
196     <ObjectPropertyDomain>
197         <ObjectProperty IRI="#hasPredicate"/>
198         <Class IRI="#State"/>
199     </ObjectPropertyDomain>
200     <ObjectPropertyDomain>
201         <ObjectProperty IRI="#hasPurpose"/>
202         <Class IRI="#Status-Function"/>
203     </ObjectPropertyDomain>
204     <ObjectPropertyDomain>
205         <ObjectProperty IRI="#hasStatus"/>
206         <Class IRI="#Status-Function"/>
207     </ObjectPropertyDomain>
208     <ObjectPropertyDomain>
209         <ObjectProperty IRI="#isConsequenceOf"/>
210         <Class IRI="#State"/>
211     </ObjectPropertyDomain>
```

```
212     <ObjectPropertyDomain>
213         <ObjectProperty IRI="#isPredicateOf"/>
214         <Class IRI="#Predicate"/>
215     </ObjectPropertyDomain>
216     <ObjectPropertyRange>
217         <ObjectProperty IRI="#hasConsequence"/>
218         <Class IRI="#State"/>
219     </ObjectPropertyRange>
220     <ObjectPropertyRange>
221         <ObjectProperty IRI="#hasPredicate"/>
222         <Class IRI="#Predicate"/>
223     </ObjectPropertyRange>
224     <ObjectPropertyRange>
225         <ObjectProperty IRI="#hasPurpose"/>
226         <Class IRI="#Purpose"/>
227     </ObjectPropertyRange>
228     <ObjectPropertyRange>
229         <ObjectProperty IRI="#hasStatus"/>
230         <Class IRI="#Status-Function"/>
231     </ObjectPropertyRange>
232     <ObjectPropertyRange>
233         <ObjectProperty IRI="#isConsequenceOf"/>
234         <Class IRI="#Purpose"/>
235     </ObjectPropertyRange>
236     <ObjectPropertyRange>
237         <ObjectProperty IRI="#isPredicateOf"/>
238         <Class IRI="#State"/>
239     </ObjectPropertyRange>
240     <DataPropertyRange>
241         <DataProperty IRI="#hasPrice"/>
242         <Datatype abbreviatedIRI="xsd:double"/>
243     </DataPropertyRange>
244     <DataPropertyRange>
245         <DataProperty IRI="#hasTitle"/>
246         <Datatype abbreviatedIRI="xsd:string"/>
247     </DataPropertyRange>
248     <DataPropertyRange>
249         <DataProperty IRI="#hasValue"/>
250         <Datatype abbreviatedIRI="xsd:double"/>
251     </DataPropertyRange>
252 </Ontology>
```

Listing C.6 – OWL Ontology - Conquer Territory

Bob's code is shown in Listing C.7. This code contains the plan that *Bob* should execute to achieve its `territory_conquered` goal. In this plan, *Bob* uses `alg1` to find available institutional actions that can help it achieve its goal. In addition, *Bob* uses `try_actions` to try to perform one of the actions found in `alg1` that would satisfy its goal without inflicting its anti-goal.

```
1  anti_goal(soldier_killed_from_allied_base).
2
3  !territory_conquered. // agent goal
4
5  +!territory_conquered
6    <- !alg1(territory_conquered,Actions);
7       !try_actions(Actions).
8
9  +!try_actions(Actions) <-
10   if(.length(Actions, Size) & Size < 1){
11     .fail;
12   }
13   .queue.head(Actions,Action);
14   !alg2(Action, States);
15     if(anti_goal(AG) & .member(AG, States)){
16       .queue.remove(Actions, Action);
17         !try_actions(Actions);
18   }
19     else{
20       Action;
21   }.
22
23 { include("$jacamoJar/templates/common-cartago.asl") }
24 { include("$jacamoJar/templates/common-moise.asl") }
```

Listing C.7 – Bob's Program - Example 2 and Case 2

It is important to be clear that the Listings used in the previous examples that are not replaced in the following examples remain valid and therefore are not repeated. The complete code is available in a git repository at `https://github.com/rafhaelrc/PurposeModel/tree/main/demo/conquer_territory`.

APPENDIX D – APPLICATION EXAMPLE 3: POSTING INFORMATION ON SOCIAL NETWORKS

D.1 CASE 1 - SOCIAL NETWORKS WITHOUT STATUS FUNCTIONS AND PURPOSES

The artifact code shown in Listing D.1 has implemented the functions `sendMessageByTwitter`, `talkWithBot`, `uploadAPicture`, and `uploadAMessage` that refer to the social networks *Twitter*, *Telegram*, *Instagram* and *Facebook* respectively. For the sake of simplicity, it was decided to encode all these functions in the same artifact even if they simulate the use of different social networks.

```
1  package tools;
2
3  import cartago.Artifact;
4  import cartago.OPERATION;
5  import java.io.IOException;
6  import java.util.logging.Logger;
7
8
9  public class ElectronicMachine extends Artifact{
10   private Logger logger = Logger.getLogger(ElectronicMachine.
       class.getName());
11
12   @OPERATION
13   public void init()  {
14     defineObsProperty("task", 0);
15   }
16
17   @OPERATION
18   public void sendMessageByTwitter() {
19     // implement the action..
20   }
21
22   @OPERATION
23   public void talkWithBot() {
24     // implement the action..
25   }
26
27   @OPERATION
28   public void uploadAPicture() {
29     // implement the action..
30   }
```

```
31
32    @OPERATION
33    public void uploadAMessage() {
34      // implement the action..
35    }
36  }
```

Listing D.1 – Artifact - Example 3 and Case 1

The JCM code shown in Listing D.2 specifies the agent *Bob*, its focus on the artifact `machine`, the workspace `wsp` and the artifact `machine` that is available in that space.

```
1  mas social_networks {
2
3    agent bob {
4        focus: machine
5      }
6
7      workspace wsp {
8      artifact machine: tools.ElectronicMachine()
9    }
10 }
```

Listing D.2 – JCM - Example 3 and case 1

Bob's code is shown in Listing D.3. This code contains the plan that *Bob* should execute to achieve its `published_info` goal. Furthermore, this code has specified through beliefs the environmental consequences of performing `sendMessageByTwitter`, `talkWithBot`, `uploadAPicture`, and `uploadAMessage`. Based on this information, *Bob* checks whether the action to be performed has inflicted its anti-goal. If is the case, *Bob* avoids the action. If not, *Bob* executes it.

```
1  anti_goal(fake_news_spread).
2  cause(sendMessageByTwitter, fake_news_spread).
3  cause(talkWithBot, fake_news_spread).
4  cause(uploadAPicture, fake_news_spread).
5  cause(uploadAMessage, fake_news_spread).
6
7  !published_info. // agent goal
8
9  +!published_info : knet(twitter)
10     <- if(anti_goal(AG) & cause(sendMessageByTwitter,AG)){
11          .fail;
12        } else{
13           sendMessageByTwitter;
```

```
14        }.
15
16 +!published_info : knet(telegram)
17     <- if(anti_goal(AG) & cause(talkWithBot,AG)){
18         .fail;
19        } else{
20           talkWithBot;
21        }.
22
23 +!published_info : knet(instagram)
24     <- if(anti_goal(AG) & cause(uploadAPicture,AG)){
25         .fail;
26        } else{
27           uploadAPicture;
28        }.
29
30 +!published_info : knet(facebook)
31     <- if(anti_goal(AG) & cause(uploadAMessage,AG)){
32         .fail;
33        } else{
34           uploadAMessage;
35        }.
36
37 { include("$jacamoJar/templates/common-cartago.asl") }
38 { include("$jacamoJar/templates/common-moise.asl") }
```

Listing D.3 – Bob's Program - Example 3 and Case 1

D.2 CASE 2 - SOCIAL NETWORKS WITH STATUS FUNCTIONS AND PURPOSES

The JCM code shown in Listing D.4 is similar to Listing D.2. The difference is the addition of the agent's focus on the `inst_test_art` artifact, the addition of the institutional specification associated with the `wsp` workspace, and the inclusion of the `onto` artifact that serves to add the ontology that implements the purpose model. Again, for the sake of simplicity, it was decided to implement all the functions to interact with social networks in the same artifact. The example works without any harm if each function for interacting with each social network is implemented in a separate workspace and artifact.

```
1 mas social_networks {
2
3   agent bob {
4       focus: machine, inst_test.inst_test_art
5     }
```

```
6
7       workspace wsp {
8       artifact onto: mas.OntologyArtifact("src/resources/
            conquer_territory_ontology.owl")
9       artifact machine: tools.ElectronicMachine()
10    }
11
12    institution inst_test: src/resources/constitutive-specification
          .sai {
13      workspaces: wsp
14    }
15  }
```

Listing D.4 – JCM - Example 2 and case 2

The institutional specification code is shown in Listing D.5. It contains the status function and constitutive rule used in this example.

```
1  institution_id : bhInst.
2
3  status_functions:
4
5  events: tweet, messageByTelegram, postByInstagram, postByFacebook
       .
6
7
8  constitutive_rules:
9
10 1: sendMessageByTwitter count-as tweet.
11 2: talkWithBot count-as messageByTelegram.
12 3: uploadAPicture count-as postByInstagram.
13 4: uploadAMessage count-as postByFacebook.
```

Listing D.5 – Institutional specification Example 3 and Case 2

The ontology's XML code is shown in Listing D.6. This code contains the classes (e.g., status function, purpose, etc.), the connections between the classes (e.g., hasPurpose, hasConsequence, etc.) and the individuals representing the elements of this scenario (e.g., messageByTwitter, fake_news_spread, etc.).

```
1  <?xml version="1.0"?>
2  <Ontology xmlns="http://www.w3.org/2002/07/owl#"
3       xml:base="http://www.semanticweb.org/rafhaelrc/ontologies
             /2020/8/semantic_sf_5"
4       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5       xmlns:xml="http://www.w3.org/XML/1998/namespace"
```

```
6       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
7       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
8       ontologyIRI="http://www.semanticweb.org/rafhaelrc/ontologies
           /2020/8/semantic_sf_5">
9      <Prefix name="" IRI="http://www.semanticweb.org/rafhaelrc/
          ontologies/2020/8/semantic_sf_5"/>
10     <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
11     <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-
          syntax-ns#"/>
12     <Prefix name="sf5" IRI="http://www.semanticweb.org/rafhaelrc/
          ontologies/2020/8/semantic_sf_5"/>
13     <Prefix name="xml" IRI="http://www.w3.org/XML/1998/namespace"
          />
14     <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
15     <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema
          #"/>
16     <Prefix name="swrl" IRI="http://www.w3.org/2003/11/swrl#"/>
17     <Prefix name="swrlb" IRI="http://www.w3.org/2003/11/swrlb#"/>
18     <Declaration>
19         <Class IRI="#Parameter"/>
20     </Declaration>
21     <Declaration>
22         <Class IRI="#Predicate"/>
23     </Declaration>
24     <Declaration>
25         <Class IRI="#Purpose"/>
26     </Declaration>
27     <Declaration>
28         <Class IRI="#State"/>
29     </Declaration>
30     <Declaration>
31         <Class IRI="#Status-Function"/>
32     </Declaration>
33     <Declaration>
34         <ObjectProperty IRI="#hasConsequence"/>
35     </Declaration>
36     <Declaration>
37         <ObjectProperty IRI="#hasParameter"/>
38     </Declaration>
39     <Declaration>
40         <ObjectProperty IRI="#hasPredicate"/>
41     </Declaration>
```

```
42      <Declaration>
43          <ObjectProperty IRI="#hasPurpose"/>
44      </Declaration>
45      <Declaration>
46          <ObjectProperty IRI="#hasStatus"/>
47      </Declaration>
48      <Declaration>
49          <ObjectProperty IRI="#isConsequenceOf"/>
50      </Declaration>
51      <Declaration>
52          <ObjectProperty IRI="#isParameterOf"/>
53      </Declaration>
54      <Declaration>
55          <ObjectProperty IRI="#isPredicateOf"/>
56      </Declaration>
57      <Declaration>
58          <ObjectProperty IRI="#isPurposeOf"/>
59      </Declaration>
60      <Declaration>
61          <ObjectProperty IRI="#isStatusOf"/>
62      </Declaration>
63      <Declaration>
64          <DataProperty IRI="#hasPrice"/>
65      </Declaration>
66      <Declaration>
67          <DataProperty IRI="#hasTitle"/>
68      </Declaration>
69      <Declaration>
70          <DataProperty IRI="#hasValue"/>
71      </Declaration>
72      <Declaration>
73          <NamedIndividual IRI="#fake_news_spread"/>
74      </Declaration>
75      <Declaration>
76          <NamedIndividual IRI="#messageByTelegram"/>
77      </Declaration>
78      <Declaration>
79          <NamedIndividual IRI="#postByFacebook"/>
80      </Declaration>
81      <Declaration>
82          <NamedIndividual IRI="#postByInstagram"/>
83      </Declaration>
```

```
84      <Declaration>
85          <NamedIndividual IRI="#published_info"/>
86      </Declaration>
87      <Declaration>
88          <NamedIndividual IRI="#stateS1"/>
89      </Declaration>
90      <Declaration>
91          <NamedIndividual IRI="#transmit_information"/>
92      </Declaration>
93      <Declaration>
94          <NamedIndividual IRI="#tweet"/>
95      </Declaration>
96      <Declaration>
97          <AnnotationProperty abbreviatedIRI="owl:position"/>
98      </Declaration>
99      <ClassAssertion>
100         <Class IRI="#Predicate"/>
101         <NamedIndividual IRI="#fake_news_spread"/>
102     </ClassAssertion>
103     <ClassAssertion>
104         <Class IRI="#Status-Function"/>
105         <NamedIndividual IRI="#messageByTelegram"/>
106     </ClassAssertion>
107     <ClassAssertion>
108         <Class IRI="#Status-Function"/>
109         <NamedIndividual IRI="#postByFacebook"/>
110     </ClassAssertion>
111     <ClassAssertion>
112         <Class IRI="#Status-Function"/>
113         <NamedIndividual IRI="#postByInstagram"/>
114     </ClassAssertion>
115     <ClassAssertion>
116         <Class IRI="#Predicate"/>
117         <NamedIndividual IRI="#published_info"/>
118     </ClassAssertion>
119     <ClassAssertion>
120         <Class IRI="#State"/>
121         <NamedIndividual IRI="#stateS1"/>
122     </ClassAssertion>
123     <ClassAssertion>
124         <Class IRI="#Purpose"/>
125         <NamedIndividual IRI="#transmit_information"/>
```

```
126     </ClassAssertion>
127     <ClassAssertion>
128         <Class IRI="#Status-Function"/>
129         <NamedIndividual IRI="#tweet"/>
130     </ClassAssertion>
131     <ObjectPropertyAssertion>
132         <ObjectProperty IRI="#hasPurpose"/>
133         <NamedIndividual IRI="#messageByTelegram"/>
134         <NamedIndividual IRI="#transmit_information"/>
135     </ObjectPropertyAssertion>
136     <ObjectPropertyAssertion>
137         <ObjectProperty IRI="#hasPurpose"/>
138         <NamedIndividual IRI="#postByFacebook"/>
139         <NamedIndividual IRI="#transmit_information"/>
140     </ObjectPropertyAssertion>
141     <ObjectPropertyAssertion>
142         <ObjectProperty IRI="#hasPurpose"/>
143         <NamedIndividual IRI="#postByInstagram"/>
144         <NamedIndividual IRI="#transmit_information"/>
145     </ObjectPropertyAssertion>
146     <ObjectPropertyAssertion>
147         <ObjectProperty IRI="#hasPredicate"/>
148         <NamedIndividual IRI="#stateS1"/>
149         <NamedIndividual IRI="#fake_news_spread"/>
150     </ObjectPropertyAssertion>
151     <ObjectPropertyAssertion>
152         <ObjectProperty IRI="#hasPredicate"/>
153         <NamedIndividual IRI="#stateS1"/>
154         <NamedIndividual IRI="#published_info"/>
155     </ObjectPropertyAssertion>
156     <ObjectPropertyAssertion>
157         <ObjectProperty IRI="#hasConsequence"/>
158         <NamedIndividual IRI="#transmit_information"/>
159         <NamedIndividual IRI="#stateS1"/>
160     </ObjectPropertyAssertion>
161     <ObjectPropertyAssertion>
162         <ObjectProperty IRI="#hasPurpose"/>
163         <NamedIndividual IRI="#tweet"/>
164         <NamedIndividual IRI="#transmit_information"/>
165     </ObjectPropertyAssertion>
166     <InverseObjectProperties>
167         <ObjectProperty IRI="#hasConsequence"/>
```

```
168         <ObjectProperty IRI="#isConsequenceOf"/>
169     </InverseObjectProperties>
170     <InverseObjectProperties>
171         <ObjectProperty IRI="#hasParameter"/>
172         <ObjectProperty IRI="#isParameterOf"/>
173     </InverseObjectProperties>
174     <InverseObjectProperties>
175         <ObjectProperty IRI="#hasPredicate"/>
176         <ObjectProperty IRI="#isPredicateOf"/>
177     </InverseObjectProperties>
178     <InverseObjectProperties>
179         <ObjectProperty IRI="#hasPurpose"/>
180         <ObjectProperty IRI="#isPurposeOf"/>
181     </InverseObjectProperties>
182     <InverseObjectProperties>
183         <ObjectProperty IRI="#hasStatus"/>
184         <ObjectProperty IRI="#isStatusOf"/>
185     </InverseObjectProperties>
186     <IrreflexiveObjectProperty>
187         <ObjectProperty IRI="#hasStatus"/>
188     </IrreflexiveObjectProperty>
189     <IrreflexiveObjectProperty>
190         <ObjectProperty IRI="#isStatusOf"/>
191     </IrreflexiveObjectProperty>
192     <ObjectPropertyDomain>
193         <ObjectProperty IRI="#hasConsequence"/>
194         <Class IRI="#Purpose"/>
195     </ObjectPropertyDomain>
196     <ObjectPropertyDomain>
197         <ObjectProperty IRI="#hasPredicate"/>
198         <Class IRI="#State"/>
199     </ObjectPropertyDomain>
200     <ObjectPropertyDomain>
201         <ObjectProperty IRI="#hasPurpose"/>
202         <Class IRI="#Status-Function"/>
203     </ObjectPropertyDomain>
204     <ObjectPropertyDomain>
205         <ObjectProperty IRI="#hasStatus"/>
206         <Class IRI="#Status-Function"/>
207     </ObjectPropertyDomain>
208     <ObjectPropertyDomain>
209         <ObjectProperty IRI="#isConsequenceOf"/>
```

```
210         <Class IRI="#State"/>
211     </ObjectPropertyDomain>
212     <ObjectPropertyDomain>
213         <ObjectProperty IRI="#isPredicateOf"/>
214         <Class IRI="#Predicate"/>
215     </ObjectPropertyDomain>
216     <ObjectPropertyRange>
217         <ObjectProperty IRI="#hasConsequence"/>
218         <Class IRI="#State"/>
219     </ObjectPropertyRange>
220     <ObjectPropertyRange>
221         <ObjectProperty IRI="#hasPredicate"/>
222         <Class IRI="#Predicate"/>
223     </ObjectPropertyRange>
224     <ObjectPropertyRange>
225         <ObjectProperty IRI="#hasPurpose"/>
226         <Class IRI="#Purpose"/>
227     </ObjectPropertyRange>
228     <ObjectPropertyRange>
229         <ObjectProperty IRI="#hasStatus"/>
230         <Class IRI="#Status-Function"/>
231     </ObjectPropertyRange>
232     <ObjectPropertyRange>
233         <ObjectProperty IRI="#isConsequenceOf"/>
234         <Class IRI="#Purpose"/>
235     </ObjectPropertyRange>
236     <ObjectPropertyRange>
237         <ObjectProperty IRI="#isPredicateOf"/>
238         <Class IRI="#State"/>
239     </ObjectPropertyRange>
240     <DataPropertyRange>
241         <DataProperty IRI="#hasPrice"/>
242         <Datatype abbreviatedIRI="xsd:double"/>
243     </DataPropertyRange>
244     <DataPropertyRange>
245         <DataProperty IRI="#hasTitle"/>
246         <Datatype abbreviatedIRI="xsd:string"/>
247     </DataPropertyRange>
248     <DataPropertyRange>
249         <DataProperty IRI="#hasValue"/>
250         <Datatype abbreviatedIRI="xsd:double"/>
251     </DataPropertyRange>
```

```
252 </Ontology>
```

Listing D.6 – OWL Ontology - Social Networks

Bob's code is shown in Listing D.7. This code contains the plan that *Bob* should execute to achieve its `published_info` goal. In this plan, *Bob* uses `alg1` to find available institutional actions that can help it achieve its goal. In addition, *Bob* uses `try_actions` to try to perform one of the actions found in `alg1` that would satisfy its goal without inflicting its anti-goal.

```
1  anti_goal(fake_news_spread).
2
3  !published_info. // agent goal
4
5  +!published_info
6    <- !alg1(published_info,Actions);
7       !try_actions(Actions).
8
9  +!try_actions(Actions) <-
10   if(.length(Actions, Size) & Size < 1){
11     .fail;
12   }
13   .queue.head(Actions,Action);
14   !alg2(Action, States);
15     if(anti_goal(AG) & .member(AG, States)){
16       .queue.remove(Actions, Action);
17         !try_actions(Actions);
18   }
19     else{
20       Action;
21   }.
22
23 { include("$jacamoJar/templates/common-cartago.asl") }
24 { include("$jacamoJar/templates/common-moise.asl") }
```

Listing D.7 – Bob's Program - Example 3 and Case 2

It is important to be clear that the Listings used in the previous examples that are not replaced in the following examples remain valid and therefore are not repeated. The complete code is available in a git repository at `https://github.com/rafhaelrc/PurposeModel/tree/main/demo/social_networks`.