# Specifying and Executing Open Multi-Agent Systems

Alexander Artikis[1,2], Marek Sergot[3], Jeremy Pitt[4], Dídac Busquets[5], and Régis Riveret[5]

[1] University of Piraeus, Greece
[2] NCSR Demokritos, Athens, Greece
[3] Department of Computing, Imperial College London, UK
[4] Electrical & Electronic Engineering Department, Imperial College London, UK
`a.artikis@unipi.gr`,
`{m.sergot, j.pitt, d.busquets, r.riveret}@imperial.ac.uk`

**Abstract.** Multi-agent systems where the agents are developed by parties with competing interests, and where there is no access to an agent's code, are often classified as 'open'. The members of such systems may inadvertently fail to, or even deliberately choose not to, conform to the system specification. Consequently, it is necessary to specify the normative relations that exist between the agents, such as permission, obligation and institutional power. We summarise our work on executable specification of open multi-agent systems. We adopt a bird's eye view of these systems, as opposed to an agent's perspective whereby it reasons about how it should act. Our specifications are 'dynamic' in the sense that they may be modified at run-time by the agents. We encode specifications in the 'Event Calculus for Run-Time reasoning' (RTEC). Therefore, our framework supports the execution of very large multi-agent systems.

## 1 Introduction

An 'open' Multi-Agent System (MAS) is a system in which the member agents are developed by different parties and serve different, often competing interests. In open MAS, the behaviour of an agent cannot be predicted in advance [8]. Furthermore, an agent may choose not to conform to the MAS specification in order to achieve its individual goals, or it may fail to conform to the MAS specification due to, say, a bug in its code [21]. Agents may also fail to behave as intended because of factors beyond their control. This is commonplace when open MAS are deployed on distributed environments with unreliable communication channels. A few examples of this type of MAS are electronic marketplaces, virtual organisations and digital media rights management applications.

We summarise our framework for *executable specification* of open MAS [2, 3, 5]. We adopt a bird's eye view of MAS, as opposed to an agent's own perspective whereby it reasons about how it should act. Moreover, we view open MAS as instances of *normative systems* [9]. A feature of this type of system is that actuality, what is the case, and ideality, what ought to be the case, do not necessarily coincide. Therefore, we specify what is permitted, prohibited and obligatory. We also represent explicitly the *institutional powers* [10,19,20] of the agents, and maintain the standard, long established distinction between institutional power, permission and physical capability. Institutional power refers to the standard feature of any normative system whereby designated agents, when acting in specified roles, are empowered by an institution to create relations or states of affairs of special significance within the institution. Consider, for example, the case where an agent is empowered by an institution to award a contract and thereby create a set of normative relations between the contracting parties.

Several approaches have been proposed in the literature for the specification of open MAS. Most of these approaches offer 'static' MAS specifications, that is, there is no support for run-time specification modification. In some open MAS, however, environmental, social or other conditions may favour specifications that are modifiable during the system execution. Consider, for instance, the case of a malfunction of a large number of sensors in a sensor network, or the case of manipulation of a voting procedure due to strategic voting. To deal with such issues, our framework supports 'dynamic' MAS specifications, that is, specifications that are developed at design-time but may be modified at run-time by the members of a system.

We encode specifications of open MAS in the 'Event Calculus for Run-Time reasoning' (RTEC) [6]. The Event Calculus [12] is a simple and flexible logic programming formalism for representing and reasoning about events and their effects. RTEC includes various optimisation techniques for an important class of computational tasks, specifically those in which given a record of what events have occurred (a 'narrative') and a set of axioms (expressing the specification of a MAS), we compute the values of various facts (denoting institutional powers, permissions, and other normative relations) at specified time points. RTEC thus provides a practical means of informing the decision-making of the agents and their users, and the MAS designers.

**Brief History and Applications** The development of the presented framework started in the context of the EU ALFEBIITE project [16]. In addition to the Event Calculus, we have been using (variants of) the action language $C+$ to specify and execute MAS [4]. In the recent years, however, we focused on highly scalable computational tools and thus produced RTEC, which supports the run-time execution of very large MAS.

Our framework has been used in various applications, such as voting [15], resource-sharing [17], negotiation [3,5] and argumentation [4].

## 2 Meta-model

### 2.1 Assumptions

We view open MAS from an external perspective, that is, we consider only externally observable states of affairs. Our framework supports:

1. The development of formal, declarative specifications of open MAS.

2. The run-time adaptation of a MAS specification in order to meet the dynamic environmental and social conditions.
3. The execution of the specifications of (very large) MAS for the benefit of the agents, the agents' users and the MAS designers. For example, during the run-time activities, our framework executes the MAS specification in order to compute the institutional powers, permissions, obligations and sanctions current at each time.

In Sections 2.2 and 2.3 we discuss the structure of specifications and the techniques for run-time adaptation, while in Section 3 we present the tool for specification execution.

### 2.2 Specification

We maintain the standard and long established distinction between physical capability, institutional power and permission [10, 13]. Accordingly, we present a four-level specification of open MAS that expresses:

– the physical capabilities,
– institutional powers,
– permissions, prohibitions and obligations of the agents;
– the sanctions and enforcement policies that deal with the performance of forbidden actions and non-compliance with obligations.

The first level of specification concerns the externally observable physical capabilities of an agent. For instance, in a virtual soccer field we may express the conditions in which an agent is capable of 'kicking' the ball. The remaining three levels of specification are described next.

**Institutional Power** The term institutional (or 'institutionalised') power refers to the characteristic feature of an institution — legal system, formal organisation, or informal grouping — whereby designated agents, often when acting in specific roles, are empowered to create or modify facts of special significance in that institution — *institutional facts* in the terminology of [20] — usually by performing a specified kind of act, such as when an agent signs a contract, or the chairperson of a formal meeting declares the meeting closed. This concept has received considerable attention within the jurisprudential literature, usually under the headings of 'legal power', 'legal capacity' or 'norms of competence'.

According to the account given by [10], institutional power can be seen as a special case of a more general phenomenon whereby an action, or a state of affairs, $A$ — because of the rules and conventions of an institution — counts, in that institution, as an action or state of affairs $B$ [20]. Consider, for example, the case where sending a letter with a particular form of words counts as making an offer, or raising a hand counts as making a bid at an auction, or banging the table with a wooden mallet counts as declaring a meeting closed.

For the specification of the effects of actions within institutions, it is essential to distinguish between, for example, the act of making an offer and the act by means of which that offer is made (such as sending a letter). Banging the table with a wooden mallet is not the same act as closing a meeting. Indeed, it is only if the table is banged by a person with the institutional power to close the meeting that the meeting is thereby declared closed. The same act performed by an agent without this power has no effect on the status of the meeting (though it may have other effects). In such examples, we say that an agent 'has the institutional power' (or just 'power'), or 'is empowered', to close the meeting by means of banging the table with a wooden mallet.

**Permission** This level of specification provides the definitions of permitted, prohibited and obligatory actions. These definitions are application-specific. In some cases, we might want to associate institutional powers with permissions. For example, in some open MAS an agent is permitted to perform an action if that agent is empowered to perform that action. According to this definition, an agent is always permitted to exercise its institutional powers. In other MAS the relationship is stronger: an agent is permitted to perform an action if *and only if* it is empowered to perform that action. In general, however, there is no standard relationship between powers and permissions. For example, it is sometimes valuable to *forbid* an agent to perform an action even if it is empowered to perform that action. Jones and Sergot [10] cite Makinson [13] to illustrate the distinction between these concepts:

> "[C]onsider the case of a priest of a certain religion who does not have permission, according to instructions issued by the ecclesiastical authorities, to marry two people, only one of whom is of that religion, unless they both promise to bring up the children in that religion. He may nevertheless have the [*institutional*] *power* to marry the couple even in the absence of such a promise, in the sense that if he goes ahead and performs the ceremony, it still counts as a valid act of marriage under the rules of the same church even though the priest may be subject to reprimand or more severe penalty for having performed it." [13, p.409]

Similarly, the specification of obligations is application-specific. Determining what actions are permitted, prohibited or obligatory enables the classification of the behaviour of individual agents and the MAS as a whole into categories such as 'social' or 'anti-social', 'acceptable' or 'unacceptable', and so on. For example, the behaviour of an agent might be considered 'anti-social' or 'unacceptable' if that agent performs (certain) forbidden actions or does not comply with its obligations. Based on the behaviour of the individual agents, it is possible to classify the behaviour of the MAS as a whole. For instance, the state of a MAS may be considered 'unacceptable' if the majority of its members have not complied with their obligations.

**Enforcement Policies** This level of specification expresses the sanctions and enforcement policies that deal with 'anti-social' or 'unacceptable' behaviour. We

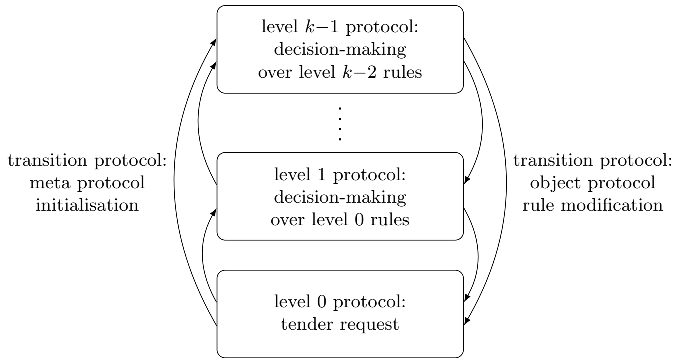
level $k{-}1$ protocol: decision-making over level $k{-}2$ rules
level 1 protocol: decision-making over level 0 rules
level 0 protocol: tender request
transition protocol: meta protocol initialisation
transition protocol: object protocol rule modification

**Fig. 1.** A Framework for Dynamic Specifications.

are concerned with the following issues: (i) when is an agent sanctioned, and (ii) what is the penalty that the agent has to face (in the case that it does get sanctioned). The specification of both of these issues is also application-specific. As far as the first is concerned, agents may be sanctioned for not complying with their obligations, or they may be sanctioned if they perform forbidden actions. As regards the second issue, penalties can come in many different forms. The rules of an auction house, for example, may stipulate that bidders who bid out of turn, and are therefore considered 'sanctioned', are no longer empowered to enter other auctions. In different settings, the same type of misbehaviour might create different penalties, such as *bad reputation*. In any case, a sanction should be proportional to the offence being committed, and the system should provide mechanisms for appealing such sanctions [14].

Sanctions are one means by which an open MAS may discourage 'unacceptable' or 'anti-social' behaviour. Another mechanism is to try to devise additional controls that will force agents to comply with their obligations or prevent them from performing forbidden actions. In an automated auction, for example, forbidden bids may be physically blocked, in the sense that their transmission is disabled. The general strategy of designing mechanisms to force compliance and eliminate non-permitted behaviour is termed *regimentation* [9]. It has been argued that regimentation is rarely desirable (it results in a rigid system that may discourage agents from entering it), and not always practical (it may require a highly complex infrastructure). In any case, violations may still occur even when regimenting a computational system (consider, for instance, a faulty regimentation device). For all of these reasons, we cannot rely exclusively on regimentation mechanisms.

## 2.3 Self-Organisation

Our framework supports dynamic specifications, that is, it allows agents to alter the specification of a MAS at run-time [2]. In the terminology of the framework, the tender request protocol, used a running example in the book, is the 'object' protocol. At any point in time during the execution of the object protocol the participants may start a 'meta' protocol in order to potentially modify the object protocol rules — for instance, replace an existing rule-set with a new one. The meta protocol may be any protocol for decision-making over rule modification, such as voting or argumentation. Furthermore, the participants of the meta protocol may initiate a meta-meta protocol to modify the rules of the meta protocol, or they may initiate a meta-meta-meta protocol to modify the rules of the meta-meta protocol, and so on. In a $k$-level framework, level 0 corresponds to the main (tender request, in this paper) protocol, while a protocol of level $n$ ($0{<}n{\leq}k{-}1$) is created, by the protocol participants of a level $m$ ($0{\leq}m{<}n$), in order to decide whether to modify the protocol rules of level $n{-}1$. The framework for dynamic (tender request) specifications is displayed in Figure 1.

Apart from object and meta protocols, the framework includes 'transition' protocols — see Figure 1 — that is, procedures that express, among other things, the conditions in which an agent may successfully initiate a meta protocol, the roles that each meta protocol participant will occupy, and the ways in which an object protocol is modified as a result of the meta protocol interactions. (A role $r$ is associated with a set of constraints expressing the powers, permissions, obligations and sanctions of $r$, and a set of 'conditions' that an agent must satisfy in order to be *eligible* to occupy $r$.)

A protocol specification consists of the 'core' rules that are always part of the specification, and the *Degrees of Freedom* (*DoF*), that is, the specification components that may be modified at run-time. A protocol specification with $l$ DoF creates an $l$-dimensional specification space where each dimension corresponds to a DoF. A point in the $l$-dimensional specification space, or *specification point*, represents a complete protocol specification — a *specification instance* — and is denoted by an $l$-tuple where each element of the tuple expresses a 'value' of a DoF. One way of evaluating a proposal for specification point change, that is, for changing the current specification of a MAS, is by modelling a dynamic protocol specification as a metric space [7]. More precisely, we compute the 'distance' between the proposed specification point and the current point. We constrain the process of specification point change by permitting proposals only if the proposed point is not too 'far' from/'different' to the current point. The motivation for formalising such a constraint is to favour gradual changes of a MAS specification.

## 3 Language and Tool

We encode specifications of open MAS in a dialect of the Event Calculus [12] called 'Event Calculus for Run-Time reasoning' (RTEC) [6]. The Event Calculus is a logic programming formalism for representing and reasoning about events and their effects. We also use RTEC for executing the MAS specifications. For example, given the messages exchanged between the agents, RTEC computes the normative relations current at each time.

**Table 1.** Main predicates of RTEC.

| **Predicate** | **Meaning** |
|---|---|
| $\mathsf{happensAt}(E,\ T)$ | Event $E$ occurs at time $T$ |
| $\mathsf{holdsAt}(F = V,\ T)$ | The value of fluent $F$ is $V$ at time $T$ |
| $\mathsf{holdsFor}(F = V,\ I)$ | $I$ is the list of the maximal intervals for which $F = V$ holds continuously |
| $\mathsf{initiatedAt}(F = V,\ T)$ | At time $T$ a period of time for which $F = V$ is initiated |
| $\mathsf{terminatedAt}(F = V,\ T)$ | At time $T$ a period of time for which $F = V$ is terminated |
| $\mathsf{union\_all}(L,\ I)$ | $I$ is the list of maximal intervals produced by the union of the lists of maximal intervals of list $L$ |
| $\mathsf{intersect\_all}(L,\ I)$ | $I$ is the list of maximal intervals produced by the intersection of the lists of maximal intervals of list $L$ |
| $\mathsf{relative\_complement\_all}(I',\ L,\ I)$ | $I$ is the list of maximal intervals produced by the relative complement of the list of maximal intervals $I'$ with respect to every list of maximal intervals of list $L$ |

## 3.1 Representation

The time model of RTEC is linear and includes integer time-points. Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. *Fluents* express properties that are allowed to have different values at different points in time. The term $F = V$ denotes that fluent $F$ has value $V$ — Boolean fluents are a special case in which the possible values are true and false. $\mathsf{holdsAt}(F = V, T)$ represents that fluent $F$ has value $V$ at a particular time-point $T$. $\mathsf{holdsFor}(F = V, I)$ states that $I$ is the list of the maximal intervals for which $F = V$ holds continuously. holdsAt and holdsFor are defined in such a way that, for any fluent $F$, $\mathsf{holdsAt}(F = V, T)$ if and only if $T$ belongs to one of the maximal intervals of $I$ for which $\mathsf{holdsFor}(F = V, I)$.

An *event description* in RTEC includes rules that define the event instances with the use of the happensAt predicate, the effects of events with the use of the initiatedAt and terminatedAt predicates, and the values of the fluents with the use of the holdsAt and holdsFor predicates. Table 1 summarises the RTEC predicates available to the MAS specification developer.

We represent the actions of the agents and the environment by means of the happensAt predicate, while the state of the agents and the environment are represented as fluents. In MAS execution, therefore, the task is to compute the maximal intervals for which a fluent representing an agent or environment variable (such as the institutional powers of an agent) has a particular value continuously.

Fluents in RTEC are of two kinds: *simple* and *statically determined*. For a simple fluent $F$, $F = V$ holds at a particular time-point $T$ if $F = V$ has been initiated by an event that has occurred at some time-point earlier than $T$, and has not been terminated at some other time-point in the meantime. This is an implementation of the *law of inertia*. To compute the *intervals* $I$ for which $F = V$, that is, $\mathsf{holdsFor}(F = V, I)$, we find all time-points $T_s$ at which $F = V$ is initiated, and then, for each $T_s$, we compute the first time-point $T_f$ after $T_s$ at which $F = V$ is terminated. The time-points at which $F = V$ is initiated are computed by means of domain-specific initiatedAt rules. In the tender request protocol, for example, we are interested in identifying the conditions in which a contract is established between two parties:

$$\begin{aligned}&\mathsf{initiatedAt}(\mathit{contract}(C, B, \mathit{Tdr}) = \mathsf{true},\ T)\ \mathsf{iff}\\ &\qquad \mathsf{happensAt}(\mathit{award}(C, B, \mathit{Tdr}),\ T),\\ &\qquad \mathsf{holdsAt}(\mathit{pow}(\mathit{award}(C, B, \mathit{Tdr})) = \mathsf{true},\ T)\end{aligned} \tag{1}$$

$\mathit{contract}(C, B, \mathit{Tdr}) = \mathsf{true}$ expresses a contact between agents $C$ and $B$ on tender $\mathit{Tdr}$. $\mathit{award}(C, B, \mathit{Tdr})$ is an action stating that agent $C$ awards agent $B$ tender $\mathit{Tdr}$. The *pow* fluent expresses institutional power. $\mathit{pow}(\mathit{award}(C, B, \mathit{Tdr})) = \mathsf{true}$ states that agent $C$ has the institutional power to award agent $B$ tender $\mathit{Tdr}$. According to rule (1), a contract is established between agents $C$ and $B$ on tender $\mathit{Tdr}$ if and only if $C$ exercises its institutional power to award $B$ tender $\mathit{Tdr}$. The conditions in which an agent has the institutional power (or is empowered) to award a tender are presented in Section 4.

Similarly, we define rules terminating $\mathit{contract}(C, B, \mathit{Tdr}) = \mathsf{true}$. The maximal intervals during which this fluent holds continuously are computed using the built-in RTEC predicate holdsFor from rule (1) and the rules terminating the *contract* fluent.

In addition to the domain-independent definition of holdsFor, an event description may include domain-specific holdsFor rules, used to define the values of a fluent $F$ in terms of the values of other fluents. Such a fluent $F$ is called *statically determined*. Examples will be given presently. holdsFor rules of this kind make use of interval manipulation constructs — see the last three items of Table 1. The interval manipulation constructs of RTEC support the following type of definition: for all time-points $T$, $F = V$ holds at $T$ if and only if some Boolean combination of fluent-value pairs holds at $T$. For a wide range of fluents, this is a much more concise definition than the traditional style of Event Calculus representation, that is, identifying the various conditions under which the fluent is initiated and terminated so that maximal intervals can then be computed using the domain-independent holdsFor. The interval manipulation constructs of RTEC can also lead to much more efficient computation [6].

## 3.2 Reasoning

RTEC executes MAS specifications by computing and storing the maximal intervals of fluents expressing the institutional powers, permissions, obligations, etc of each agent. MAS execution takes place at specified query times $Q_1, Q_2, \ldots$. At each $Q_i$ the input events, that is, the actions of the agents and the environment, that fall within a specified interval — the 'working memory' ($\mathit{WM}$) or 'window' — are taken into consideration. All input events that took place before or at $Q_i - \mathit{WM}$ are discarded. This is to make the cost of MAS execution dependent only on the $\mathit{WM}$ size and not on the complete event history. The $\mathit{WM}$ size, and the temporal distance between two consecutive query times — the inter-query 'step' ($Q_i - Q_{i-1}$) — are set by the user (for instance, the MAS designers).

At $Q_i$, the maximal fluent intervals computed by RTEC are those that can be derived from the input events that occurred in the interval $(Q_i - \mathit{WM}, Q_i]$, as recorded at time $Q_i$. When $\mathit{WM}$ is longer than the inter-query step, that is, when $Q_i - \mathit{WM} < Q_{i-1} < Q_i$, it is possible that an input event occurs in the interval $(Q_i - \mathit{WM}, Q_{i-1}]$ but arrives at RTEC only after $Q_{i-1}$; its effects are taken into account at query time $Q_i$. In the common case that input events arrive at RTEC with delays (due to communication network issues), it is preferable therefore to make $\mathit{WM}$ longer than the inter-query step. Note that information may still be lost. Any input events arriving between $Q_{i-1}$ and $Q_i$ are discarded at $Q_i$ if they took place before or at $Q_i - \mathit{WM}$. To reduce the possibility of losing information, one may increase the $\mathit{WM}$ size. Doing so, however, decreases the efficiency of MAS execution. Further details about RTEC may be found in [6] while the source code is available from `https://github.com/aartikis/RTEC`.

## 4 The Framework in Use

To illustrate our framework for executable specification of open MAS, we present a specification and execution of the tender request protocol. Illustrations of self-organisation may be found in [2].

### 4.1 Specification

Section 2.2 presented the main constructs of MAS specifications. In this section, we focus on institutional power and permission. Example formalisations of physical capability, obligation, sanction and social role may be found in [3, 5].

**Institutional Power** The conditions in which an agent has the institutional power to bid for a tender may be expressed as follows:

$$\begin{aligned}
&\mathsf{holdsFor}(\mathit{pow}(\mathit{bid}(B,\ \mathit{Tdr})) = \mathsf{true},\ I)\ \mathsf{iff}\\
&\quad \mathsf{holdsFor}(\mathit{role\_of}(B,\ \mathit{bidder}) = \mathsf{true},\ I_1),\\
&\quad \mathsf{holdsFor}(\mathit{status}(\mathit{Tdr}) = \mathit{bidding\_open},\ I_2),\\
&\quad \mathsf{intersect\_all}([I_1, I_2],\ I)
\end{aligned} \tag{2}$$

Institutional power is expressed by means of the statically determined $\mathit{pow}$ fluent. The $\mathit{role\_of}$ fluent expresses the roles that each agent occupies. The $\mathit{status}(\mathit{Tdr})$ fluent expresses the status of tender $\mathit{Tdr}$. $\mathit{status}(\mathit{Tdr}) = \mathit{bidding\_open}$ states that bidding is open for $Tdr$, $status(Tdr) = bidding\_closed$ denotes that bidding is closed for $Tdr$ and decisions about the winner must be made, and $status(Tdr) = null$ states that the deliberation over $Tdr$ has ended. $\mathsf{intersect\_all}(L, I)$ computes the list $I$ of maximal intervals such that $I$ represents the intersection of maximal intervals of the lists of list $L$, as, for example:

$$\mathsf{intersect\_all}([[(26,31)],[(21,26),(30,40)]],\ [(30,31)])$$

$I$ in $\mathsf{intersect\_all}(L, I)$ is a list of maximal intervals that includes each time-point that is part of all lists of $L$.

According to rule (2), the list of maximal intervals $I$ during which an agent $B$ is empowered to place a bid for tender $Tdr$ is computed by calculating the intersection of the list of maximal intervals $I_1$ during which $B$ occupies the role of bidder and the list of maximal intervals $I_2$ during which $Tdr$ is open for bidding. Put simply, $B$ is empowered to bid for $Tdr$ if and only if $B$ is a bidder and $Tdr$ is open for bidding.

Exercising the power to place a bid initiates the *submittedBid* fluent:

$$\begin{aligned}
&\mathsf{initiatedAt}(submittedBid(B,\ Tdr) = P,\ \ T)\ \mathsf{iff}\\
&\quad \mathsf{happensAt}(bid(B,\ Tdr, P),\ \ T),\\
&\quad P \neq null,\\
&\quad \mathsf{holdsAt}(pow(bid(B,\ Tdr)) = \mathsf{true},\ \ T)
\end{aligned} \tag{3}$$

$submittedBid(B,\ Tdr) = P$ states that agent $B$ has exercised its power to bid $P$ for tender $Tdr$. The maximal intervals during which $submittedBid(B,\ Tdr) = P$ holds continuously are computed using the built-in RTEC predicate $\mathsf{holdsFor}$ from rule (3) and the rule (not shown here) terminating $submittedBid(B,\ Tdr) = P$.

Exercising the power to bid for a tender indirectly empowers the chair of the procedure to award the submitted bid:

$$\begin{aligned}
&\mathsf{holdsFor}(pow(award(C, B,\ Tdr)) = \mathsf{true},\ \ I)\ \mathsf{iff}\\
&\quad \mathsf{holdsFor}(role\_of(C,\ chair) = \mathsf{true},\ \ I_1),\\
&\quad \mathsf{holdsFor}(status(Tdr) = null,\ \ I_2),\\
&\quad \mathsf{holdsFor}(submittedBid(B,\ Tdr) = null,\ \ I_3),\\
&\quad \mathsf{relative\_complement\_all}(I_1,\ \ [I_2, I_3],\ \ I)
\end{aligned} \tag{4}$$

$status(Tdr) = null$ states that the deliberation over tender $Tdr$ has ended. $submittedBid(B,\ Tdr) = null$ states that agent $B$ has not exercised its power to bid for $Tdr$. $\mathsf{relative\_complement\_all}(I', L, I)$ computes the list $I$ of maximal intervals such that $I$ represents the relative complements of the list of maximal intervals $I'$ with respect to the maximal intervals of the lists of list $L$. Below is an example of $\mathsf{relative\_complement\_all}$:

$$\begin{aligned}
&\mathsf{relative\_complement\_all}([(5,20),(26,50),(60,70)],\\
&\quad [[(1,4),(55,65)],[(52,80)]],\ \ [(5,20),(26,50)])
\end{aligned}$$

$I$ in $\mathsf{relative\_complement\_all}(I', L, I)$ is a list of maximal intervals that includes each time-point of $I'$ that is not part of any list of $L$.

Rule (4) states that an agent $C$ is empowered to award a tender $Tdr$ to an agent $B$ if and only if $C$ occupies the role of chair, the deliberation over $Tdr$ has not ended, and $B$ has exercised its power to bid for $Tdr$.

Exercising the power to award a tender creates a contract between the chair of the procedure and the awarded bidder (see rule (1)). A contract creates a set of further powers of the contracting parties. These are expressed similarly to the ones presented above. Additionally, a contract creates a bundle of permissions and obligations — examples of permission are given below.

**Permission** This level of specification provides the definitions of permitted, prohibited and obligatory actions. Consider the formulation below:

$$\begin{aligned}
&\mathsf{holdsFor}(\mathit{per}(\mathit{bid}(B,\ \mathit{Tdr})) = \mathsf{true},\ I)\ \mathsf{iff}\\
&\quad \mathsf{holdsFor}(\mathit{pow}(\mathit{bid}(B,\ \mathit{Tdr})) = \mathsf{true},\ I_1),\\
&\quad \mathsf{holdsFor}(\mathit{submittedBid}(B,\ \mathit{Tdr}) = \mathit{null},\ I_2),\\
&\quad \mathsf{intersect\_all}([I_1, I_2],\ I)
\end{aligned} \tag{5}$$

The fluent *per* expresses the permission to perform an action. According to rule (5), an agent is permitted to submit a bid if and only if the agent is empowered to submit a bid and has not already exercised this power.

Below is another example of permission:

$$\begin{aligned}
&\mathsf{holdsFor}(\mathit{per}(\mathit{award}(C, B, \mathit{Tdr})) = \mathsf{true},\ I)\ \mathsf{iff}\\
&\quad \mathsf{holdsFor}(\mathit{status}(\mathit{Tdr}) = \mathit{bidding\_closed},\ I_1),\\
&\quad \mathsf{holdsFor}(\mathit{atLeastZBids}(\mathit{Tdr}) = \mathsf{true},\ I_2),\\
&\quad \mathsf{holdsFor}(\mathit{bestBidder}(\mathit{Tdr}) = B,\ I_3),\\
&\quad \mathsf{intersect\_all}([I_1, I_2, I_3],\ I)
\end{aligned} \tag{6}$$

$\mathit{status}(\mathit{Tdr}) = \mathit{bidding\_closed}$ states that bidding is closed for $Tdr$. $\mathit{atLeastZBids}(\mathit{Tdr})$ becomes true when at least $Z$ agents have exercised their institutional power to place a bid for $Tdr$. $\mathit{bestBidder}(\mathit{Tdr}) = B$ states that, according to some set of rules, the bid of agent $B$ is the most suitable for $Tdr$. According to rule (6), therefore, agent $C$ is permitted to award agent $B$ the tender $Tdr$ if and only if bidding is closed, there are at least $Z$ bids for $Tdr$ and $B$ is the best bidder.

In this formalisation, an agent $C$ may be empowered to award a bidder $B$ but forbidden to exercise this power — if for example $B$ is not the best bidder. Exercising this power will create a contract between $C$ and $B$, but $C$ may be subject to penalty for performing this forbidden action (in some cases $C$ may lose the role of chair and therefore its associated institutional powers).

In rule (6) we chose to formalise *atLeastZBids* and *bestBidder* as fluents as opposed to atemporal predicates. This way, the specifications of these constraints may change at run-time by means of a meta-protocol (see Section 2.3).

In a similar way, we may formalise the permissions and obligations that stem from a contract. Note that there are many possible specifications of permitted actions (as well as obligatory actions and enforcement policies). The ones presented here were chosen for the sake of providing a concrete illustration.

**Table 2.** Example run of the tender request protocol.

| happensAt | | $b_2$ | | | $c$ | |
|---|---|---|---|---|---|---|
| | pow | per | obl | pow | per | obl |
| | | | | open | open | |
| $open(tdr)$ at 1 | | | | | | |
| | **bid** | **bid** | | **close** | | |
| $bid(b_1, tdr)$ at 2 | | | | | | |
| | bid | bid | | close, **award($b_1$)** | | |
| $bid(b_2, tdr)$ at 3 | | | | | | |
| | bid | | | close, award($b_1$), **award($b_2$)** | | |
| $bid(b_3, tdr)$ at 4 | | | | | | |
| | bid | | | close, award($b_1$), award($b_2$), **award($b_3$)** | | |
| $deadline(tdr)$ at 5 | | | | | | |
| | | | | award($b_1$), award($b_2$), award($b_3$) | **award($b_2$)** | **award($b_2$)** |
| $award(c, b_2, tdr)$ at 7 | | | | | | |
| | **service**(c) | **service**(c) | **service**(c) | | | |

## 4.2 Execution

In order to illustrate our framework for executable specification, in this section we present an example run of the tender request protocol. In this example, there are three bidders, $b_1, b_2, b_3$, and one chair, $c$. Table 2 presents the stream of events of the run (see the first column) and the powers, permissions and obligations of $b_2$ and $c$. To avoid clutter we omit the powers, permissions and obligations of $b_1$ and $b_3$. We also simplify the descriptions of events and fluents by omitting some of their parameters. The bold font indicates that the power, permission or obligation has become available as a result of the most recent event of the stream.

Given a (not necessarily temporally sorted) event stream, we may query RTEC to determine the system state current at each time. For example, to find out the maximal intervals $I$ for which agent $b_2$ is permitted to bid for tender $tdr$, we instruct RTEC to compute the following query:

$$?- \mathsf{holdsFor}(per(bid(b_2, tdr)) = \mathsf{true},\ I)$$

Similarly, to find out whether $b_2$ is permitted to bid for tender $tdr$ at some specific time-point, say time-point 3, we compute the following query:

$$?- \mathsf{holdsAt}(per(bid(b_2, tdr)) = \mathsf{true},\ 3)$$

We discuss next the system states of the run displayed in Table 2.

Initially bidders do not have any power or permission since there is no open tender. On the other hand, the chair has both the power and permission to open a tender for bidding. Once the chair opens the tender (time 1), the bidders have the power and permission to bid (see rules (2) and (5)). Moreover, the chair has the power to close the bidding procedure. Note, however, that the chair does not have the permission to exercise this power, since a tender must be open for bidding at least for some specified interval (4 time-points in this example).

At time 2, bidder $b_1$ submits its bid, and, while it maintains the power to bid, it loses the permission to exercise this power since it is permitted to bid only once (see rule (5)). The submission of the bid empowers the chair to award that bid (see rule (4)), though it does not give the chair the permission to exercise this power, since the tender is still open for bidding (see rule (6)). The action $bid(b_1, tdr)$ at time 2 has no effect on the powers and permissions of $b_2$ and $b_3$.

At times 3 and 4, $b_2$ and $b_3$ submit their bids — consequently, they lose the permission to bid, while $c$ gains the power to award the submitted bids.

At time 5, the deadline for receiving bids is reached. Due to the inability of RTEC to express future fluent termination, we model deadlines by means of external events emitted by a clock. (Note that, even in the presence of *deadline* events, a *close* action is necessary in order to deal with faulty clocks.) At this point (time 5), the chair gains the permission to award bidder $b_2$ (see rule (6)), since it is the best bidder (we omit the details of calculating the best bidder), and there have been enough submitted bids (in this example, the number of bids must be no less than the number of bidders). For the same reasons, $c$ is obliged to award $b_2$.

By awarding the bid of $b_2$ (time 7), a contract is established between $c$ and $b_2$ (see rule (1)). Consequently, $c$ and $b_2$ acquire a bundle of powers, permissions and obligations. For example, $b_2$ is obliged to offer a service to $c$. The award also finalises the deliberation over the tender, removing from the chair the power and permission to award some other agent.

The state of a MAS, including the powers, permissions, obligations, sanctions and roles that are associated with each agent at each time, computed by RTEC, may be publicised at run-time to (a subset of) the members of a MAS, or their designers. Such run-time services may be provided by a central server or in various distributed configurations. Each agent, for example, may have available an RTEC module in order to compute its powers, permissions, obligations, and so on. In another setting, some agents may have direct access to the MAS state by means of a local RTEC module, whereas other agents may rely on trusted third parties for such information. A further discussion of the possible run-time configurations, including the issues that arise in each configuration (such as policies for the disclosure of the MAS state), will be presented elsewhere.

## 5 Critical Assessment

We summarised our framework for executable specification of open MAS [2,3,5]. The algorithms of RTEC have proven efficient and scalable for Big Data applications [1,6] and are therefore suitable for executing very large MAS. Our approach is declarative and thus further optimisations of the reasoning algorithms do not require changes in the representation language.

The MAS specifications developed with our framework exhibit a formal semantics — event descriptions in RTEC are (locally) stratified logic programs [18]. Validation and traceability of the effects of events are therefore supported. The availability of logic programming additionally allows MAS specifications to include both temporal and atemporal constraints. Moreover, RTEC includes the formalisation of inertia, which facilitates considerably the development of succinct MAS specifications, and consequently code maintenance.

Our framework supports self-organisation via dynamic specifications. We distinguish between successful and unsuccessful attempts to change a MAS specification. We evaluate an agent's proposal for specification change by modelling a dynamic specification as a metric space, and constrain the enactment of proposals that do not meet the evaluation criteria.

RTEC has direct routes to machine learning via abductive-inductive logic programming [11]. Consequently, we can employ methods for generating and refining MAS specifications in an automated way. Furthermore, RTEC has routes to reasoning under uncertainty via probabilistic logic programming [22] and Markov Logic [23]. This way, we can deal with various types of noise such as that introduced by unreliable communication channels.

RTEC does not have a mechanism for changing the value of a fluent in the future. In MAS specifications, it is often necessary to express that a property is terminated at some future time. For instance, in the tender request protocol, the power to bid expires after a certain period. Moreover, RTEC does not have mechanisms for dealing with asynchronous event sources. Addressing these issues are topics of current research.

## Acknowledgements

Dídac Busquets was supported by the UK EPSRC Grand Challenge Project Autonomic Power System, and Régis Riveret was supported by the Marie Curie Intra-European Fellowship PIEF-GA-2012-331472.

## References

1. E. Alevizos, A. Artikis, K. Patroumpas, M. Vodas, Y. Theodoridis, and N. Pelekis. How not to drown in a sea of information: An event recognition approach. In *IEEE International Conference on Big Data*, pages 984–990, 2015.
2. A. Artikis. Dynamic specification of open agent systems. *Journal of Logic and Computation*, 22(6):1301–1334, 2012.
3. A. Artikis and M. Sergot. Executable specification of open multi-agent systems. *Logic Journal of the IGPL*, 18(1):31–65, 2010.
4. A. Artikis, M. Sergot, and J. Pitt. An executable specification of a formal argumentation protocol. *Artificial Intelligence*, 171(10–15):776–804, 2007.

5. A. Artikis, M. Sergot, and J. Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, 10(1), 2009.
6. A. Artikis, M. J. Sergot, and G. Paliouras. An event calculus for event recognition. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 27(4):895–908, 2015.
7. V. Bryant. *Metric Spaces*. Cambridge University Press, 1985.
8. C. Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47:79–106, 1991.
9. A. Jones and M. Sergot. On the characterisation of law and computer systems: the normative systems perspective. In *Deontic Logic in Computer Science: Normative System Specification*, pages 275–307. J. Wiley and Sons, 1993.
10. A. Jones and M. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, 1996.
11. N. Katzouris, A. Artikis, and G. Paliouras. Incremental learning of event definitions with inductive logic programming. *Machine Learning*, 100(2-3):555–585, 2015.
12. R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–96, 1986.
13. D. Makinson. On the formal representation of rights relations. *Journal of Philosophical Logic*, 15:403–425, 1986.
14. E. Ostrom. *Governing the commons: The evolution of institutions for collective actions*. Political economy of institutions and decisions, 1990.
15. J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Voting in multi-agent systems. *Computer Journal*, 49(2):156–170, 2006.
16. J. Pitt, A. Mamdani, and P. Charlton. The open agent society and its enemies: a position statement and research programme. *Telematics and Informatics*, 18(1):67–87, 2001.
17. J. Pitt, J. Schaumeier, and A. Artikis. Axiomatisation of socio-economic principles for self-organising institutions: Concepts, experiments and challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 7(4), 2013.
18. T. Przymusinski. On the declarative semantics of stratified deductive databases and logic programs. In *Foundations of Deductive Databases and Logic Programming*. Morgan, 1987.
19. J. Rawls. Two concepts of rules. *Philosophical Review*, 64(1):3–32, 1955.
20. J. Searle. *Speech Acts*. Cambridge University Press, 1969.
21. M. Sergot. Modelling unreliable and untrustworthy agent behaviour. In B. Dunin-Keplicz, A. Jankowski, A. Skowron, and M. Szczuka, editors, *Proceedings of Workshop on Monitoring, Security, and Rescue Techniques in Multiagent Systems (MSRAS)*, Advances in Soft Computing, pages 161–178. Springer-Verlag, 2004.
22. A. Skarlatidis, A. Artikis, J. Filipou, and G.s Paliouras. A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming (TPLP)*, 15(2):213–245, 2015.
23. A. Skarlatidis, G. Paliouras, A. Artikis, and G. A. Vouros. Probabilistic event calculus for event recognition. *ACM Transactions on Computational Logic*, 16(2):11:1–11:37, 2015.