N-Jason: Run-Time Norm Compliance in AgentSpeak(L)

JeeHang Lee[1], Julian Padget[1],
Brian Logan[2], Daniela Dybalova[2], and Natasha Alechina[2]

[1] Department of Computer Science,
University of Bath,
Bath, BA2 7AY, UK
{j.lee,j.a.padget}@bath.ac.uk
[2] School of Computer Science,
University of Nottingham,
Nottingham, NG8 1BB, UK
{bsl,dxd,nza}@cs.nott.ac.uk

Abstract. Normative systems offer a means to govern agent behaviour in dynamic open environments. Under the governance, agents themselves must be able to reason about compliance with state- or event-based norms (or both) depending upon the formalism used. This paper describes how norm awareness enables a BDI agent to exhibit norm compliant behaviour at run-time taking into account normative factors. To this end, we propose ***N-Jason***, a run-time norm compliant BDI agent framework supporting norm-aware deliberation as well as run-time norm execution mechanism, through which new unknown norms are recognised and bring about the triggering of plans. To be able to process a norm such as an obligation, the agent architecture must be able to deal with deadlines and priorities, and choose among the plans triggered by a particular norm. Consequently, we extend the syntax and the scheduling algorithm of AgentSpeak(RT) to operate in the context of ***Jason***/AgentSpeak(L) and provide 'real-time agency', which we explain through a detailed examination of the operational semantics of a single reasoning cycle.

Keywords: Norms, BDI, Agent Programming Language, Normative System.

1 Introduction

In conventional development of BDI agents, norm compliance is typically achieved by design. That is, by specifying plans that are triggered by detached norms, because the agent programmer knows which norms the agent shall adopt, and then prioritising those rules so that the supporting norms are chosen over those preferred by the agent's mental attitudes, in order to suppress conflicts between the normative and the agent's existing goals. This creates an undesirable dependence between the agent implementation and the norm implementation, which creates two issues:

1. When an agent encounters new and unknown norms, which were not taken into account at design time, there is typically no plan to deal with those norms in the

F. Dalpiaz et al. (Eds.): EMAS 2014, LNAI 8758, pp. 367–387, 2014.

plan library at run-time. Hence, norm compliant behaviour cannot normally be exhibited because the norms are unavoidably ignored. Yet worse, agents may suffer a punishment from the enforcement of the normative system as a result of a violation caused by their incapacity to process the normative event.
2. The hierarchical prioritisation of normative over ordinary plans deprives an agent of its autonomy, since the norms in effect are treated as hard constraints, whose violation is not possible.

We believe that such tensions can be resolved by the use of an extended model of norm awareness. In the literature on BDI agents, norm awareness, which is a precursor to norm compliance, is typically manifested in two places: (i) at the *perception* level, by taking new unknown norms into account as part of the generic execution mechanism [13,14] and (ii) at the *deliberation* level, by attempts to resolve the conflict between normative factors and agents' mental attitudes [1,9]. We propose to coalesce these approaches into one 'sense–think–act' reasoning cycle informed by the concept of awareness, which Charlton [4] describes as the capacity *"to select and integrate relevant inputs from a complex environment to enable humans or animals to choose between a large repertoire of behavioural responses"*. This definition reminds us that, in order to be norm aware, agents should have knowledge (or understanding) about norms in respect of: (i) what (state) the norms intend to reach or to achieve, (ii) which action plans are appropriate to execute norms and (iii) which behaviour agents should prefer between normative goals and the agent's own interests.

Thus, this paper addresses the convergence of these approaches in the context of the BDI agent architecture, in order to be able to ground the discussion of how the extended model of norm awareness enables a BDI agent to exhibit norm compliant behaviour at run-time. To do so, we propose ***N-Jason***, a run-time norm-compliant BDI agent framework supporting a run-time norm execution mechanism, under which new and unknown norms are recognised and enable the triggering of an appropriate plan (if present), in conjunction with norm-aware deliberation [1]. To be able to process a norm such as an obligation, the agent architecture should be able to deal with deadlines and priorities, and choose among plans triggered by a particular norm. Consequently, we extend the syntax and the scheduling algorithm of AgentSpeak(RT) [15] to operate in the context of ***Jason***/AgentSpeak(L) [3] and provide 'real-time agency', which we explain through a detailed examination of the operational semantics of a single reasoning cycle.

The paper is organised as follows. In §2 an institutional framework and semantics of norms considered in ***N-Jason*** are introduced. It is followed by §3, where we present a run-time norm compliant BDI agent framework including programming language and interpreter. After the operational semantics in §4, related work and the contribution of this work are contrasted in §5. The conclusion and future work are discussed in §6.

2 Institutional Framework

Normative frameworks can be viewed as a kind of external repositories of (normative) knowledge from which (normative) guidance may be delivered to agents. Usually, a normative framework is composed of a set of rules whose purpose is to model the normative

positions established by the actions of agents and hence realise the governance of individual agents in the society. These rules are not hard-coded recipes presenting reactive behaviours, such as those in the static expert systems, but rather describe consequences arising from observations for the purpose of reasoning about the current context, resulting in situation-specific norms. The framework identifies not only correct and incorrect actions but also norms such as obligations, permissions and prohibitions through the institutional trace that records its evolving internal state, subject to observed external events representing actions in the external world.

Depending on the formalism of the normative system, norms can be categorised as state- or event-based. State-based norms usually express higher level norms that impose desirable or required states on the system (or an environment), often as a logical combination of institutional facts, which should be brought about by the actions of agents [8]. In contrast, event-based norms generally represent relatively lower level activities addressing possibly executable events (or actions) at the individual agent level [7]. In this paper, we use Cliffe's institutional model [5] for the purpose of providing detached event-based norms, upon which we develop the run-time norm compliance model presented here.

The institutional framework provides a formal action language Inst*AL* to specify norms, describing coordinations and interactions between agents and (or) environments in the context of an institution. The normative specification is translated to a computational model that utilises Answer Set Programming (ASP) [10], which enables reasoning about the current context described in the institution. The institution is composed of a set of *institutional states*, evolving over time triggered by the occurrence of both internal and external *events*. An institutional state is a set of *fluents* which are present (denoting true) or absent (denoting false) at a given time instant. In addition, such institutional fluents are divided into *domain fluents* and *normative fluents* which are further partitioned into: (i) *power* ($\mathcal{W}$) – indicates events that are empowered to bring about institutional change (ii) *permission* ($\mathcal{P}$) – indicates events that can occur without violation, and (iii) *obligations* ($\mathcal{O}$) – specifies events that are obliged to happen before the occurrence of a deadline (e.g. a timeout), or else a violation occurs.

These normative fluents represent the normative consequences of particular behaviours which should be achieved by agents in a certain context. For example, if an agent *X* is obliged to carry out an action *act* by deadline *deadline* otherwise the violation event *violation* is generated, the form of the normative information is represented as:

```
obl(act, deadline, violation)
```
(*obligation*)

Also if an agent *X* is permitted to perform an action *act*, then the representation is:

```
perm(act)
```
(*permission*)

The determination of those normative consequences is carried out using an answer set solver driven by a rule-based specification (Inst*AL*) which explores all possible outcomes derivable from the institutional state arising from the occurrence of a single

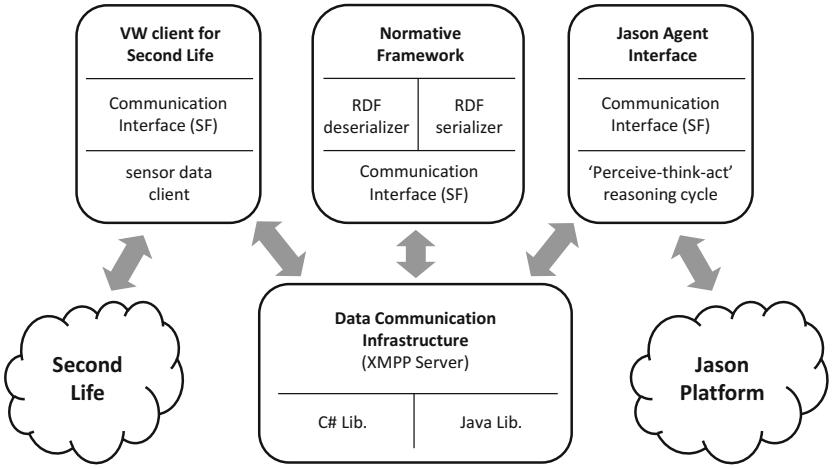

Fig. 1. Governing virtual characters behaviour with Insitutions

event[1] as determined by the generation and consequence rules that comprise the institutional model.

Lee *et al.* [12] demonstrate a governance mechanism using this institutional model that shows how the normative consequences of particular actions can be delivered to agents' minds as percepts (to conventional ***Jason*** agents rather than the variety described here) either on request or by subscription, making them available for the agent reasoning process. The components in this case comprise: (i) the virtual agents (VA) in Second Life, (ii) an institutional model for social norm reasoning, and (iii) BDI agents that are responsible for individual reasoning, as illustrated in Figure 1. The virtual agents (VA) in the virtual world appear as sensors to the rest of the system: as soon as virtual world events are detected in *Second Life* (*SL*), the VA turns them into symbolic representations and publishes them, while both the BDI agent and the institution subscribe to that topic. When the institution receives this information, it triggers the social norm reasoning process, which determines the new normative positions of the actors and identifies appropriate behaviours for the current (social) situation. This information is then published as `perm(act)` or `obl(act, deadline, violation)` for the BDI agent to incorporate into its reasoning process, following the principles set out in [1]. When the decision making process (for norm compliance) is completed, the action plans are published, which are then interpreted by the VA using the atomic actions available in the virtual world – because the virtual agent actions are typically more primitive that those of the intelligent agent.

With regards to the norm compliance in BDI agents, as described in Figure 1, van Riemsdijk *et al.* suggest in [14] that one feasible approach for run-time norm execution is the use of *"pre-existing capabilities"* in the agent program when an agent encounters

[1] Note: the institutional model can also function as a normative oracle for an agent, if presented with a sequence of events, in which case it derives all the possible outcomes from all possible orderings of those events, subject to whatever constraints are specified on the ordering.

new and unknown norms. This assumes that event-based norms can identify the associated necessary actions, since event-based norms typically refer to relatively low-level activities that address possibly executable events (or actions) at the individual agent level [7]. If appropriate information can be extracted from the detached norm, such that it is recognisable to an agent, in this way an agent presumably may execute unknown norms and so exhibit a form of norm compliance at run-time.

For example, the `act` term in an obligation represents a similar level of knowledge to plans or events in a BDI agent program. If an agent can retrieve and recognise what action (or event) is required to be achieved, then it can trigger certain plans and attempt to carry out such behaviour even though the norm is not handled explicitly in the agent specification. With regard to the norm-aware reasoning, an agent may deduce a preference, if it is able to know the relative priorities, and critical impact or the deadline of normative factors by extracting `deadline` and `violation` information. This norm-aware reasoning may allow an agent to pursue its own preferences between its own goals, norms and sanctions by measuring feasibility, as proposed by Alechina *et al.* [1]. In this paper, we only use obligations for such purpose, in order to focus on the essential aspects of the agent's internal reasoning process. Additionally, we consider the handling of prohibitions for the compatibility with other normative systems, however they are not explicit in the institution mechanism employed here.

3 The *N-Jason* BDI Agent Framework

In this section we outline ***N-Jason***, a norm aware BDI agent interpreter and its programming language for run-time norm compliant agent behaviour. In principle, it extends ***Jason***/AgentSpeak(L) syntactically, semantically and in the reasoning process of the interpreter. In practice, ***N-Jason*** is conceptually similar to AgentSpeak(RT) [15], which is capable of dealing with deadlines and priorities and scheduling intentions with the aim of providing real-time agency. ***N-Jason*** is conceptually a superset of AgentSpeak(RT), to which it adds normative concepts (i.e. obligations, permissions, prohibitions, deadlines, priorities and durations) and norm aware deliberation.

We firstly examine work to date with regards to the programming language aspect. This is followed by an informal explanation of the ***N-Jason*** reasoning cycle. Subsequently, we show how the extended model of norm awareness in BDI agents is established by the combination of the run-time norm execution mechanism and norm-aware deliberation.

3.1 The *N-Jason* Agent Programming Language

A ***N-Jason*** agent consists of four main components: beliefs, goals, events and a set of plans. Beliefs and goals are identical to those in standard ***Jason***, while events and plans are extended. We now give a brief summary of the extended features of the basic elements in the agent specification. We take advantage of ***Jason***'s plan annotation mechanism to provide deadline, duration and priority information, so that each feature is simply a term, such as `deadline(X)`, `duration(Y)` or `priority(Z)`, where the parameters are (positive) integer literals. The interpretation of these annotations and examples are covered in the following.

Belief: A *belief* represents agent's information (e.g. initial states of an agent, internal knowledge established through the reasoning cycle) and its knowledge about the environments wherein agents are situated (e.g. percepts observed by agents, messages containing the information about other agents and norms delivered from normative frameworks). Typically, a belief is represented as a grounded atomic formula. The collection of beliefs is referred to as a belief base, which contains belief literals in the form of belief atoms and negations.

Goal: A *goal* is one of two basic types: an achievement goal or a test goal. The former are usually specified as predicates prefixed by the '!' operator. This specifies a certain state of the environment that the agent wants to achieve, which is indicated when the predicate associated with its achievement goal is true. The latter test goal, for which the prefix is the '?' operator, indicates that agents want to know whether the associated predicate is a true belief.

Event: An *event* is the main component for triggering agent's plans. In principle, changes in agent's mental attitudes (i.e. beliefs, goals and intentions) give rise to events. There are two types of events: one is an addition event denoted by '+', which means the addition of a belief or an achievement goal. The other is a deletion event denoted by '–', referring to a recantation of a base belief.

As in ***Jason***, an addition event is categorised by a belief addition event denoted by '+' and a goal addition event jointly denoted by '+' and '!'. All external belief changes bring about belief addition events, so as to initiate the execution of corresponding plans. In contrast, the goal addition event results from both internal and external changes in goals. In other words, explicit goals from the users or other agents result in a goal addition event, but also a goal addition event can be generated by internal operations affecting the agent's mental attitude, such as the execution of subgoals triggered in response to an external event.

Support for normative concepts is provided by an extension of the syntax for an event by the addition of *deadline* and *priority* information. The deadline is a real time value indicating a deadline by which an intention should be achieved. It is expressed in a some adequate unit of real world time. When the deadline is passed, it is no longer feasible to achieve an intention or to give a response with a belief change. The priority is a positive integer value that expresses the relative importance between the achievement of an intention and responding to changes in a belief. A larger value reflects a higher priority. Both can optionally be specified in the annotation (a list of terms in between square brackets "`[`" and "`]`") at the end of an event. For example the event:

```
|+!at(X, Y)[deadline(900), priority(10)]
```

specifies the goal adoption that an agent moves to the coordinate (X, Y), by the deadline 900, with priority 10. By default, the deadline is taken as infinity and the priority as zero. Note that the deadline and priority annotations do not play a part in unification at plan selection stage.

Plan: A *plan* is a sequence of actions (and subgoals) which is a means to achieve a (main) goal or a means to respond to changes in beliefs by agents. The plan typically consists of a head and a body, but sometimes an optional plan label, which defines an index, a name and other information, can be specified. The head is

composed of a triggering event, which specifies an event for which the plan is to be used and a context specifying the condition which must be true for the plan to be a candidate for execution. The body is a series of actions and subgoals to achieve a main goal.

The plan is extended to support normative concepts. Given the three main elements, a duration is proposed in ***N-Jason***, specifically in order to enable assessment of the *feasibility* of the plan associated with the deadline (see §3.4). The duration is a non-negative integer value representing a required time to execute the plan. In principle, the duration may be determined by the summation of an execution time of each external action in the plan body. For simplicity, we follow the assumption described in [1], that the estimated time for each external action is fixed and already known. Like deadline and priority, a duration can be optionally specified in the plan label in the form of an annotation (a list of terms in between square brackets "[" and "]"). For example, the plan:

```
@plan[duration(50)]
+!at(X, Y) : req(ag)
<- move_toward(X, Y); !ack(ag).
```

is triggered by the request from the agent *ag* to move to the coordinate (X, Y), and then to send back an acknowledgement to *ag*. The required (or estimated) execution time of the plan is 50.

3.2 The *N-Jason* Interpreter

The interpreter plays an important role in the operationalisation of agent programs. The agent's belief base, intentions and events are manipulated by the interpreter, and practical reasoning consisting of deliberation and means-ends reasoning is performed to achieve a goal or to respond to environmental changes.

During a single reasoning cycle, run-time norm compliance is accomplished by an extended model of norm awareness that has three steps:

1. *Event Reconsideration*, to find out what the norm is intended to achieve or to reach,
2. *Option Reconsideration*, to identify which plan is the most appropriate in response to the norm,
3. *Intention Scheduling*, to confirm the decision about which behaviour agent would prefer between goals, norms and sanctions.

The interpreter code of ***N-Jason*** is shown in Algorithm 1. B is the belief base, E is the event base, G is a set of goals and I is a set of intentions of an agent. The function *create-tevent* encodes a percept as a triggering event and returns it. The function *add-event* updates the agent's event base with an event which is a pair of a triggering event and an intention. The function *update-belief* updates the agent's belief base with a percept p. The function *type* returns a type of p, either *obligation* or *prohibition*, if p is a norm. The function *edp* constructs a triggering event using the terms in the event-based norm, if the type of p is a norm (e.g. obligations). The functions EVENT- and OPTION-RECONSIDERATION accomplish the run-time norm execution mechanism described in §3.3. The main algorithm of the SCHEDULE function which carries

Algorithm 1. ***N-Jason*** Interpreter Reasoning Cycle

1: $B := B_0$ /* B_0 are initial beliefs */
2: $G := G_0$ /* G_0 are initial goals */
3: $E := E \cup G$
4: $P := P \cup N$ /* P are percepts and N are norms */
5: **for all** $p \in P$ **and** $p \notin B$ **do**
6: te_p = *create-tevent(p)*
7: $R_{te_p} := \{\pi\theta \mid \theta$ is a mgu for te_p and plan $\pi\}$
8: **if** $R_{te_p} \neq \varnothing$ **then**
9: $E :=$*add-event*$(E,\ te_p)$
10: **else if** $R_{te_p} = \varnothing$ **and** $type(p) = (obl \mid proh)$ **then**
11: $E :=$EVENT-RECONSIDERATION(p)
12: **end if**
13: $B :=$*update-belief*$(B,\ p)$
14: **end for**
15: **for all** $\langle te, \tau\rangle \in E$ **do**
16: $O_{te} := \{\pi\theta \mid \theta$ is an applicable unifier for te and plan $\pi\}$
17: $\pi\theta\theta' := S_O(O_{te})$ where θ' is a context unifier for te and plan π
18: **if** $\pi\theta\theta' = nil$ **then**
19: $\pi\theta\theta' :=$ OPTION-RECONSIDERATION(te)
20: **end if**
21: **if** $\pi\theta\theta' \neq nil$ **and** $\tau \notin I$ **then**
22: $I := I \cup \pi\theta\theta'$
23: **else if** $\pi\theta\theta' \neq nil$ **and** $\tau \in I$ **then**
24: $I := (I \backslash \tau) \cup$ push $(\pi\theta\theta'\sigma, \tau)$ where σ is an mgu for $\pi\theta\theta'$ and τ
25: **else if** $\pi\theta\theta' = nil$ **and** $\tau \in I$ **then**
26: $I := (I \backslash \tau)$
27: **end if**
28: $I :=$SCHEDULE(I)
29: **if** $I \neq \varnothing$ **then**
30: $I :=$EXECUTE(I)
31: **end if**
32: **end for**

out norm-aware intention scheduling is shown in §3.4. The internal operation of the ***N-Jason*** interpreter is extended from [15]. We use the same notations as in [15] for consistency and comparability.

We now give an informal explanation of one reasoning cycle in the interpreter. At the start (lines 1–4), we assume that an agent perceives knowledge (P) from its environment and about its normative positions (N) (e.g. obligations) from one or more institutional frameworks. N is treated just like P, that is a form of percept at this stage, by the interpreter (line 4).

The belief base (B) and the event base (E) are updated by P in the belief update process (*belief-update-function (buf)* more precisely) (see lines 6–13). This belief update involves the creation/addition of events in response to each new percept. Once a percept (p) is encoded as a triggering event (te_p) by the function *create-tevent*, the interpreter

checks whether te_p has a set of relevant plans R_{te_p}[2] in the plan library Π. If R_{te_p} is retrieved, then E is updated with the event, a pair of te_p and its intention, by the function *add-event*. If no relevant plan is retrieved, te_p is ignored but B is updated in any case with p by the function *update-belief*. The same approach is taken for norms when the norms and its relevant plans are already specified in the agent program. Otherwise, the event reconsideration process (line 11) starts to find out what the norms are intended to achieve, as the first step in run-time norm execution.

Next, the interpreter starts the reasoning process in order to determine an applicable plan[3] in the selected set of applicable plans (O_{te}). The selection function S_O chooses a single option from O_{te} as a result of the unification of event and context. If S_O retrieves nothing (denoted by nil), then the interpreter follows exactly the same path as described above. The option reconsideration process (line 19) tries to find out which action plans are appropriate to execute unknown norms, as the second step in run-time norm execution. See lines 17–20.

If one single applicable plan is successfully retrieved by S_O, then the means-ends reasoning adds the applicable plan (π) as an intended means (IM) on top of an intention (I). If te of π is an internal event then π added in the existing I, otherwise a new I is created with π to be added in there (line 21–27). This is followed by the intention scheduling process which returns a *preference maximal set* of intentions in deadline order (line 28). Afterwards, one intention selected by the intention selection function S_I is finally executed (line 30). The details of the remainder are exactly the same as in [3] or [15].

3.3 Run-Time Norm Execution

In §3.2, we explained that run-time norm execution is realised by two steps: (i) event reconsideration and (ii) option reconsideration. Prior to defining those reconsideration processes, we firstly define a property of the *executability* of norms at run-time. We say that a norm such as `obl(evt, deadline, violation)`, is *executable* at run-time iff:

1. $p \in P$ and $type(p) = (obligation \mid prohibition)$, where p is a percept, formed from a list of terms such as $term($“,” $term)^*$, in a set of newly observed percepts P at run-time;
2. $te_p \notin E$, where te_p is a triggering event generated from the percept p, and E is an event base, which is a set of events $\{(te, \tau), (te', \tau'), \ldots\}$, where an event is a pair of a triggering event and an intention (te, τ);
3. $edp(p) \neq nil$ and $\{(te_{edp(p)}, \tau_{edp(p)})\} \cap E \neq \varnothing$, where $edp(p)$ is a function extracting the obliged event together with its deadline and priority from p, $te_{edp(p)}$ is a triggering event of the $edp(p)$, an event term in the norm, and $\tau_{edp(p)}$ is an intention of $te_{edp(p)}$ and
4. $R_{te_{edp(p)}} \neq \varnothing$, where $R_{te_{edp(p)}}$ is a set of relevant plans.

[2] A relevant plan for a particular event is a plan whose triggering event matches the particular event. There can be many relevant plans for each triggering event in general [3].

[3] An applicable plan is a candidate plan for execution, which has a context that evaluates to true given the agent's current beliefs [3].

Algorithm 2. Event Reconsideration

Require: $P \coloneqq P \cup N$
Require: $te_p = \textit{create-tevent}(p)$
1: **if** $p \in P$ **and** $type(p) = obligation$ **then**
2: $te_{edp(p)}$ = *create-tevent*($edp(p)$)
3: $R_{te_{edp(p)}} \coloneqq \{\pi\theta \mid \theta$ is a mgu for $te_{edp(p)}$ and plan $\pi\}$
4: **if** $R_{te_{edp(p)}} \neq \varnothing$ **then**
5: $E \coloneqq$*add-event*(E, te_p)
6: **end if**
7: **else if** $p \in P$ **and** $type(p) = prohibition$ **then**
8: $\Xi \coloneqq$*add-prohibition*$(\Xi, edp(p))$
9: **end if**

The executability determines the necessity of further reconsideration for the new and unknown norms. If those norms are judged executable at the perception stage, the event-reconsideration process starts for the addition of such norms to the event base as triggering events. Similarly, the executability also enables the option-reconsideration in order to execute an applicable plan in relation to the triggering events derived from the norms.

Event Reconsideration aims to verify that a norm perceived at run-time is executable although no corresponding plan exists in the agent program. If an event extracted from a detached norm has a relevance to a certain set of plans, it thus has potential to trigger specific ones, and it is then concluded that the norm is executable. If the norm is proven to be executable, the interpreter adds the norm to the event base E as an achievement goal addition event. The procedure for event reconsideration is as follows (see Algorithm 2):

1. Extract the terms representing an obliged event, a deadline and its priority[4] from the obligation by the function *edp*, whose practical implementation may vary, depending on norm representations in various systems (line 2),
2. Construct a new triggering event (an achievement goal addition event in this case) from the combination of extracted terms (line 2),
3. Query the existence of a set of relevant plans with such a constructed triggering event (line 3),
4. Add such triggering event to E, if relevant plans are successfully retrieved (line 5) and
5. If the norm is a prohibition, then the extracted event is added into the prohibition base (Ξ) (line 7 - 8) and will be revisited at the norm deliberation stage [5].

For example, suppose there is a detached obligation `obl(at(X, Y), 1030, 10)`. If relevant plans are not found in the agent program (plan library of an agent, to be

[4] In principle, the last term is an event which arises when a violation occurs. This value normally indicates the criticality of such a violation. Higher values represents a higher priority.

[5] ***N-Jason*** supports prohibitions as described above, and is therefore compatible with normative systems supporting prohibitions, but we note that the institutional model described in §2 does not have an explicit representation of prohibition, but only the absence of permission.

precise) in response to the obligation, the function *edp* firstly extracts the event (`at(X, Y)`), deadline (`1030`) and priority (`10`) from the obligation. Next, the interpreter constructs a new triggering event (an achievement goal addition event as described above) such as `+!at(X,Y)[deadline(1030), priority(10)]` using the extracted information. Subsequently, the interpreter queries the existence of relevant plans to S_R once again with a new triggering event, `+!at(X,Y)[deadline(1030), priority(10)]`. If the retrieval of relevant plans is successful, then the original event, `+!obl(at(X, Y), 1030, 10)`, is added to E.

One exceptional aspect in event-reconsideration is the addition of a deontic event te_p (which is a detached norm) instead of a normal event $te_{edp(p)}$ (which is a newly constructed triggering event) into the event base E. In so doing, we intend to distinguish norm-triggered intentions from ordinary intentions that normal events trigger, so as to facilitate norm-aware deliberation (see §3.4) in ***N-Jason***. In principle, ***Jason*** creates different intentions in response to different triggering events. Given this characteristic, both a deontic and a normal event create a deontic and a normal intention in ***N-Jason***, respectively. The intended means included in both intentions are identical since a deontic and a normal event trigger exactly the same plan in an agent program. However, the properties (e.g. deadline and priority) of each intention are different. The normal intention follows the original deadline and priority specified in the plan. In contrast, the deontic intention has different deadline and priority, which are inherited from those in the detached norm. As a result, these intentions are the main source of norm-aware deliberation. An agent is able to deliberate on norms and agent's private goals through the evaluation of the relative importance and urgency using norm-triggered (i.e. deontic) intentions and ordinary event-triggered (i.e. normal) intentions.

Suppose a plan whose label is `example`, is specified in an agent program:

```
@example[duration(50)]
+!at(X, Y)[deadline(1000), priority(5)]
<- move_toward(X, Y); !ack(ag).
```

Assuming that a normal event triggering `example` is added to event base E. Then it creates a normal intention using a pair of normal event and its associated plan `plan_example`, whose deadline and priority are 1000 and 5, respectively. Later, a detached obligation `obl(at(X, Y), 1030, 10)` is received. Following Algorithm 2, the deontic event `+!obl(at(X, Y), 1030, 10)` is added to E, since a relevant plan `example` is found. Consequently a deontic intention is created using a pair of a deontic event and its associated plan `example`. Its deadline and priority are 1030 and 10, respectively, which are different from those in the normal intention. Obviously, we have two intentions whose properties are different, although the intended means are absolutely same. Hence, ***N-Jason*** is able to carry out norm-aware deliberation on norms and the agent's own goals using those intentions. If ***N-Jason*** simply adds a normal event instead of a deontic event when an obligation is detached, then norm-aware deliberation may not be feasible since there must be only one normal intention.

Option-Reconsideration is a central element in the practical reasoning process whereas the event reconsideration happens at the perception stage. The main objective of option reconsideration is the determination of an applicable plan corresponding to the new and unknown norm – whose executability is already verified – and is thus

Algorithm 3. Option Reconsideration

Require: $\langle te_p, \tau \rangle \in E$ where te_p is an event and τ is an intention
Ensure: $\pi\theta\theta'$ where θ' is a context unifier for $te_{edp(p)}$ and plan π
1: **if** $type(p) = obligation$ **then**
2: $te_{edp(p)} =$ *create-tevent*$(edp(p))$
3: $R_{te_{edp(p)}} \coloneqq \{\pi\theta \mid \theta$ is a mgu for $te_{edp(p)}$ and plan $\pi\}$
4: **if** $R_{te_{edp(p)}} \neq \varnothing$ **then**
5: $O_{te_{edp(p)}} \coloneqq \{\pi\theta \mid \theta$ is an applicable unifier for $te_{edp(p)}$ and plan $\pi\}$
6: $\pi\theta\theta' \coloneqq S_O(O_{te_{edp(p)}})$ where θ' is a context unifier for $te_{edp(p)}$ and plan π
7: **end if**
8: **end if**

added to E as an achievement goal addition event. If the applicable plan is chosen, then it will probably be used to enact a norm-compliant behaviour, unless it is infeasible as judged by intention scheduling (described in §3.4). The procedure is shown in Algorithm 3.

Like *Event-Reconsideration*, te_p is generated by a new and unknown norm that does not have any relevant plans R_{te_p} at this moment. Thus at the beginning of the option reconsideration, the interpreter carries out the same process for event reconsideration:

1. Extract the event term $edp(p)$ of the norm in order to retrieve relevant plans $R_{te_{edp(p)}}$ (as before), if the type of p is a norm (i.e. an obligation) (line 1 - 2),
2. Retrieve the relevant plans corresponding to the $te_{edp(p)}$ by the unification of an atomic-formula in a triggering event and each plan in an agent (line 3),
3. Determine a set of applicable plans with the constructed triggering event (line 5) and
4. Select a single applicable plan as an intended means to which to commit, through the extended unification of a triggering event, a plan and a context (line 6).

3.4 Norm Awareness in Deliberation

Norm awareness in the deliberation process is achieved by the scheduling of intentions with deadlines and priorities. We extend the algorithm proposed in [15] with the consideration of prohibitions in order to establish a conflict-free *preference maximal set* of intentions. In effect, this is like [1] who proposes a scheduling algorithm that brings about a *preference maximal set* of intentions, but that depends upon (N-)2APL's parallel execution of plans, whereas here the scheduling algorithm for (***N-***)***Jason*** has to take account of the single-threaded plan execution model in ***Jason***.

The scheduling algorithm is introduced in Algorithm 4. A set of candidate intentions $I_C = \{\tau, \tau', \dots\}$, which is sorted in descending order of a priority, is inserted into a scheduling process. If each intention is *feasible*, i.e. a plan on top of the intention can be executed before the deadline and is not prohibited by a set of prohibition $\Xi = \{\xi, \xi', \dots\}$, then the intention is added to the *preference maximal set* (Γ) whose criteria are defined as follows:

Algorithm 4. Scheduling of Intentions

```
1: Γ := ∅, Ξ' := ∅
2: for all τ ∈ I in descending order of priority do
3:     if {τ} ∪ Γ is feasible then
4:         if τ ∉ Ξ then
5:             Γ := {τ} ∪ Γ
6:         else
7:             for all ξ ∈ Ξ do
8:                 Ξ' := {τθ | θ is a mgu for ξ and intention τ}
9:             end for
10:            if priority(τ) > max{priority(ξ), ∀ξ ∈ Ξ'} then
11:                Γ := {τ} ∪ Γ
12:            end if
13:        end if
14:    end if
15: end for
16: sort Γ in order of increasing deadline
17: return Γ
```

1. An intention is feasible *iff* the execution of the intention is completed before its deadline, that is, for τ,

   $$ne(\tau) + et(\tau) - ex(\tau) \leq dl(\tau)$$

   where τ denotes an intention, $ne(\tau)$ is the time at which τ will next execute, $et(\tau)$ is the time required to execute τ, denoted in the plan label, $ex(\tau)$ is the elapsed time to execute τ to this point, and $dl(\tau)$ is the deadline for τ specified in the plan [1].

2. The intention should not be prohibited, that is, for τ
   - $\tau \notin \Xi$ or
   - $\tau \in \Xi$, then $\forall \xi \in \Xi$, $\tau = \xi$ and $priority(\tau) > max\{priority(\xi), \forall \xi \in \Xi\}$

   where τ is an intention, ξ is a prohibited event in the prohibition base Ξ and *priority* is a priority retrieval function.

Scheduling in ***N-Jason*** is also pre-emptive in that the adoption of a new intention τ may prevent scheduled intentions with lower priority than τ (including currently executing intentions) being added to the new schedule just as in N-2APL and AgentSpeak(RT). Intentions that cannot meet their deadline are dropped.

3.5 Implementation

We have implemented ***N-Jason*** on top of the existing code base for ***Jason*** version 1.3.6. The latest prototype[6] of ***N-Jason*** implements the core language extensions (i.e. syntax, semantics) described in §3.1 and the extensions (e.g. run-time norm execution, norm-aware deliberation) described in §3.3 and §3.4. In addition, we implement a norm

[6] ***N-Jason*** is available via `http://bsf.googlecode.com/svn/tags/njason-0.0.1/`

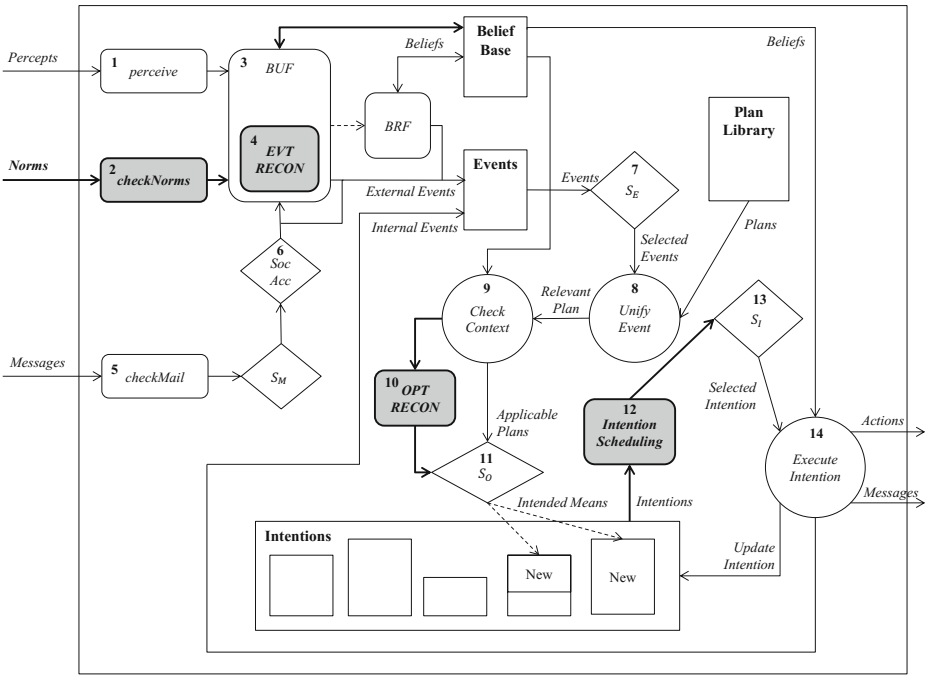

Fig. 2. Extended Features of ***N-Jason*** on ***Jason***/AgentSpeak(L)

adoption mechanism in ***N-Jason***, so that an agent under the governance of institutional frameworks is able to receive situationally appropriate norms and subsequently add them as percepts for processing by the reasoning cycle.

Figure 2 shows the extended features[7] and how they fit into the ***Jason*** interpreter. The language extensions, run-time norm execution and norm-aware deliberation are implemented as part of the reasoning cycle of the ***Jason*** interpreter. The norm adoption mechanism is implemented as an extension of the `AgArch` class.

In brief, one reasoning cycle of ***Jason*** is modelled as a transition system over states. The configuration of a ***Jason*** agent [3], contains the current state, denoted s, where $s \in$ {ProcMsg, SelEv, RelPl, ApplPl, AddIM, SelInt, ExecInt, ClrInt}. Each state has a corresponding procedure – `applyProcMsg()`, `applySelEv()`, `applyRelPl()`, `applySelAppl()`, `applyFindOp()`, `applyAddIM()`, `applyProcAct()`, `applySelInt()`, `applyExecInt()` – which are internal to the `reasoningCycle()` method in the transition system. To this transition system, we add the states RcvNorm and `schInt` and customise the procedures `reasoningCycle()`, `applyRelPl()`, `applyFindOp()`, `applySelInt()`. The complete states are detailed in §4.1.

We now sketch some details of the implementation. To begin with, we extend the ***Jason*** agent reasoning architecture class (`AgArch`) by subclassing in order to facilitate the norm adoption mechanism ($^2checkNorms$). Run-time norm execution is achieved

[7] Grey boxes with numbers 2, 4, 10, 12 in Figure 2 are new features. For more explanation about other white boxes see Chapter 4 of [3].

by *Event-* and *Option-Reconsideration* as described in Algorithms 2 and 3 above. The former, ${}^{4}EVTRECON$ in Figure 2, is implemented by customising the native belief update function (`buf()`) which is a subroutine of `reasoningCycle()` to implement Algorithm 2. For the latter, ${}^{10}OPTRECON$ in Figure 2, we customise the native option selection function (`applyFindOp()`) to implement Algorithm 3. Norm-aware deliberation, ${}^{12}IntentionScheduling$ in Figure 2, is accomplished by an intention scheduler with deadlines and priorities implemented in a newly created `IntentionScheduler` class. The scheduling (`schedule()`) method in this class is inserted just before the intention selection function (`selectIntention()`) which is a part of the `applySelInt()` procedure.

Apart from the above changes to the interpreter, the language syntax extensions are implemented by the customisation of the annotation processing routine (`setLabel()` and `setTEvent()`) in the `Plan` class.

3.6 Example

As an example, we consider robots serving beer in a pub, whose main role is to get an order and to deliver a beer to the customer. We assume the existence of some institutions delivering desirable social norms, subject to the observations of participants, and that all agents are governed by such systems. A part of the agent program is shown below:

```
@P1[duration(5)]
+!at(X, Y) : not at(X, Y) <- moveToward(X, Y).

@P2[duration(10)]
+!order(X, Y) <- get(beer); moveToward(X, Y).

// A request from customer seated at (X, Y).
// The deadline is D and the priority is P.
+request(X, Y)[deadline(D), priority(P)]
<- !order(X, Y)[deadline(D), priority(P)].
```

At time 100, the robot receives the following events:

E1: `+!request(2, 3)[deadline(130), priority(20)]`
A request from customer seated at (2, 3).
The deadline is 130 and the customer is important so the priority is 20.
E2: `+!request(1, 1)[deadline(115), priority(10)]`
A request from customer seated at (1, 1).
The deadline is 115 and the the priority is 10.
E3: `+!request(3, 3)[deadline(130), priority(10)]`
A request from customer seated at (3, 3).
The deadline is 130 and the the priority is 10.

These three events trigger the plan P2, and give rise to three possible intentions τ_1 (P2 triggered by (2, 3)), τ_2 (P2 triggered by (1, 1)) and τ_3 (P2 triggered by (3, 3)). τ_2 is not feasible, thus it is dropped, whereas τ_1 and τ_3 are feasible, so scheduled in deadline order: τ_1 is scheduled first between 100 and 110 since it has an earlier deadline followed by τ_3 between 110 and 120. Now the agent starts the execution of τ_1.

Let consider an announcement of a fire alarm by one of the normative frameworks. It broadcasts an obligation containing the coordinates of an exit to all participants so they may escape from the building. Suppose the norm is `obl(at(0, 0), 115, 100)`. Although the obligation is not stated in the agent's program, it is executable since the agent has a *pre-existing* moving ability `!at(X, Y)`, which is enough to satisfy the obligation. With the event- and option-reconsideration, the event :

E4: `+!at(0, 0)[deadline(115), priority(100)]` is generated from the obligation, thus adoption the plan P1, bringing about an intention τ_4 (P1 triggered by (0, 0)). During the execution of τ_1, τ_3 and τ_4 are inserted into a new schedule in deadline order: since the priority of τ_4 is greater than τ_3 and τ_4 has a more urgent deadline, the agent starts to execute τ_4, triggered by the obligation, before the execution of τ_3.

Notwithstanding, that this example is extremely simple, it provides a useful in-principle illustration of norm-aware deliberation – as performed by intention scheduling – as well as the run-time norm execution mechanism in ***N-Jason***.

4 Operational Semantics

In this section, we present a theoretical foundation for the ***N-Jason*** programming language with semantics based upon an extension of the operational semantics for ***Jason***/AgentSpeak(L). Given the formal semantics of ***Jason*** we extend the transition rules which transform one extended configuration into another. To begin with, we show a configuration of individual ***N-Jason*** agents which is almost unchanged except for norm configuration. In the following section, we describe the transition rules that give rise to a configuration change at each state in a single reasoning cycle. For consistency and comparability, we follow exactly the same notations as those in published ***Jason*** descriptions excepting the normative aspects.

4.1 *N-Jason* Configuration

The configuration of ***N-Jason*** is a tuple $\langle ag, C, N, T, s\rangle$ where:

- ag is an agent program consisting of a set of beliefs bs and a set of plans ps, as defined by the EBNF in [3].
- An agent's circumstance C is a tuple $\langle I, E, A\rangle$, where I is a set of *intention* $\{i, i', \ldots\}$, E is a set of *events* $\{(te, i), (te', i'), \ldots\}$, in which event is a pair of a triggering event and an intention (te, i) and A is a set of actions an agent performs in the external environment.
- N is a tuple $\langle \Gamma, \Xi\rangle$ denoting normative consequences delivered from normative systems, where Γ is a set of obligations $\{\gamma, \gamma', \ldots\}$ and Ξ is a set of prohibition $\{\xi, \xi', \ldots\}$.
- T is a tuple $\langle R, A_p, \iota, \varepsilon, \rho\ \rangle$ defining a trace of provisional information required for subsequent steps within a single reasoning cycle, where R is the set of *relevant plans*, A_p the sets of applicable plans, and ι, ε and ρ record an intention, event, and applicable plan (respectively) at a specific moment under consideration within the execution of a single reasoning cycle.

– The current state s within an agent's reasoning cycle is denoted by $s \in \{\mathsf{RcvNorm}, \mathsf{ProcMsg}, \mathsf{SelEv}, \mathsf{RelPl}, \mathsf{ApplPl}, \mathsf{AddIM}, \mathsf{SchInt}, \mathsf{SelInt}, \mathsf{ExecInt}, \mathsf{ClrInt}\}$.

4.2 Transition Rules

The execution of the ***N-Jason*** program leads the modification of the initial configuration of an agent via transition rules given below. For the sake of brevity, we do not repeat the communication semantics, since these are unaffected by the changes in relation to norms.

In general, the transition would normally start from the state ProcMsg, but we propose a preceding step RcvNorm, as described in §2 because this provides the hook for the consideration of the norm as part of the reasoning cycle. Thus, note that the initial configuration of this model is $\langle ag, C, N, T, \mathsf{RcvNorm}\rangle$, where ag is specified by the agent program and other all components are empty, and the reasoning cycle starts from RcvNorm with the transition rules given below.

Receiving Detached Norms: As described in §2, institutional frameworks may distribute norms via broadcasting when a norm is activated by the fulfilment of institutional states triggered by external events in the environment. As soon as the event-based norms are received, the norms effectively act like an ordinary event thus trigger the transition of the agent's mental state. Rule **RcvNorm** (see Figure 3) updates the agent belief base and an event base component C_E associated with adding new norms, specifically in case of obligations in an obligation base N_Γ. Otherwise, only a prohibition is added into the prohibition base and there are no updates to other components.

Relevant Plans: (see Figure 4) If the transition of states ($\mathsf{RcvNorm} \mapsto \mathsf{SelEv}$) is successful after RcvNorm and the state SelEv selects one event from the component E of which event is either $\langle te, i\rangle$ or $\langle \gamma, i\rangle$, rule **Rel_1** starts to assign the set of relevant plans to component T_R in the state RelPl. Rule **Rel_2** indicates the reconsideration situation where a new triggering event extracted from the obligation is assigned to the component C_E, where $\mathsf{Evt}(\gamma)$ is a function constructing a triggering event by the retrieval of information from γ. Rule **Rel_3** assigns a set of relevant plans to T_R in respect of the reconsidered event. Rule **Rel_4** and **Rel_5** cope with the situation where no relevant plan is retrieved. In those cases, events (both ordinary event and reconsidered event) are simply ignored and the state returns to SelEv.

Since transition rules between ($\mathsf{AppPl} \mapsto \mathsf{AddIM}$) are almost same as those in ***Jason*** we give a brief description of each rule at each state from here. If T'_R is successfully assigned then it is followed by: (i) AppPl which assigns a set of applicable plans to T_{A_P} by retrieving those relevant plans whose contexts are believed to be true, (ii) SelAppl which assigns a particular intended means selected by an option selection function S_O to T_ρ, and (iii) AddIM which adds a selected intended means to C_I which is an existing intention or a newly created one. If transitions fail between ($\mathsf{AppPl} \mapsto \mathsf{AddIM}$), then the state SelInt becomes the next step. For more information, see [3].

Scheduling of Intentions: Rule **SchInt** (see Figure 5) updates the component C'_I by the function SCHEDULE(C_I). Note that the scheduling function, SCHEDULE(C_I), sorts

$$\frac{N \neq \{\}}{\langle ag, C, N, T, \mathsf{RcvNorm}\rangle \rightarrow \langle ag', C, N', T, \mathsf{SelEv}\rangle} \quad \textbf{(RcvNorm)}$$

where: $ag'_{bs} = ag_{bs} \cup \{\gamma\}$
$N'_\Gamma = N_\Gamma \cup \{\gamma\} \vee N'_\Xi = N_\Xi \cup \{\xi\}$

Fig. 3. Transition Rule for Receiving a Norm

$$\frac{T_\varepsilon = \langle te, i\rangle \quad \mathsf{RelPlans}(ag_{ps}, te) \neq \{\}}{\langle ag, C, N, T, \mathsf{RelPl}\rangle \rightarrow \langle ag, C, N, T', \mathsf{ApplPl}\rangle} \quad \textbf{(Rel\_1)}$$

where: $T'_R = \mathsf{RelPlans}(ag_{ps}, te)$

$$\frac{T_\varepsilon = \langle \gamma, i\rangle \quad \mathsf{RelPlans}(ag_{ps}, \gamma) = \{\}}{\langle ag, C, N, T, \mathsf{RelPl}\rangle \rightarrow \langle ag, C', N, T, \mathsf{RelPl}\rangle} \quad \textbf{(Rel\_2)}$$

where: $C'_E = \{\langle \mathsf{Evt}(\gamma), i\rangle\}$

$$\frac{T_\varepsilon = \langle \mathsf{Evt}(\gamma), i\rangle \mathsf{RelPlans}(ag_{ps}, \mathsf{Evt}(\gamma)) \neq \{\}}{\langle ag, C, N, T, \mathsf{RelPl}\rangle \rightarrow \langle ag, C, N, T', \mathsf{ApplPl}\rangle} \quad \textbf{(Rel\_3)}$$

where: $T'_R = \mathsf{RelPlans}(ag_{ps}, \mathsf{Evt}(\gamma))$

$$\frac{\mathsf{RelPlans}(ag_{ps}, te) = \{\}}{\langle ag, C, N, T, \mathsf{RelPl}\rangle \rightarrow \langle ag, C, N, T, \mathsf{SelEv}\rangle} \quad \textbf{(Rel\_4)}$$

$$\frac{\mathsf{RelPlans}(ag_{ps}, \mathsf{Evt}(\gamma)) = \{\}}{\langle ag, C, N, T, \mathsf{RelPl}\rangle \rightarrow \langle ag, C, N, T, \mathsf{SelEv}\rangle} \quad \textbf{(Rel\_5)}$$

Fig. 4. Transition Rules for Relevant Plans

$$\frac{T_\rho = \{\}}{\langle ag, C, N, T, \mathsf{SchInt}\rangle \rightarrow \langle ag, C', N, T, \mathsf{SelInt}\rangle} \quad \textbf{(SchInt)}$$

where: $C'_I = \text{SCHEDULE}(C_I)$

Fig. 5. Transition Rule for Scheduling Intentions

intentions in order of priority and deadline so as to determine the *preference maximal set* of intentions discussed in §3.4.

After this step, the transition system follows the same rules as presented in [3] in order to execute an intended means in an particular intention selected by S_I in between SelInt, ExecInt and ClrInt.

5 Related Works

There has been much research over a number of years on the matter of norm compliance through the combination of normative frameworks and classical (BDI-type) cognitive agents [2,11]. However, research on compliance of norms at the individual agent level has received less attention. As discussed in §1, this problem can be decomposed into two perspectives: to facilitate a generic norm execution mechanism at run-time, and to focus on the rational decision making between norms and existing goals.

Alechina *et al.* [1] introduce N-2APL, a norm-aware BDI agent architecture and its programming language. It is able to carry out norm-aware deliberation, which aims to permit agents to resolve the conflicts between an agent's own goals, normative goals and sanctions. This is accomplished by a deadline- and priority-based intention scheduling algorithm, which weighs the feasibility for all intentions that may bring about conflicts. The (potential) sanctions may affect agent decision making, but violations are possible in this approach. Given N-2APL, Dybalova *et al.* [9] demonstrate norm-compliant agents in location-based gaming environments in conjunction with the organisational framework, 2OPL [6]. There, once organisations have broadcast state-based norms to all participants, the individual agents achieve a state of the environment described in the norms using a design-based approach. ***N-Jason*** is also able to support norm-aware deliberation in conjunction with an institutional model, which is similar to the combination of N-2APL and 2OPL, but extends the concept of norm awareness to the whole reasoning cycle. As a result, it supports agents in being design-based norm compliant, but can additionally deliver run-time compliance through norm execution.

Meneguzzi *et al.* [13] focuses on norm awareness at the perception level, by extending the AgentSpeak(L) BDI architecture with a run-time plan modification technique. It enables agents to behave appropriately in response to newly accepted norms at run-time. However, it assumes that the norms are non-conflicting, so it does not consider scheduling of plans with regards to their deadlines or possible sanctions in accordance with existing goals in agents. Whereas [13] takes a rather practical perspective, van Riemsdijk *et al.* [14] introduce a formal framework for generic norm execution, which allows agents to be norm compliant by triggering or preventing actions in new and unknown norms at design time. However the agent in [14] works at the level of individual actions (its decision mechanism chooses actions rather than plans) and the norms are specified in terms of actions, making in effect a norm-reactive agent, and it is unclear how the decision mechanism can combine actions to achieve goals and thereby the objective of a norm-deliberative agent. In ***N-Jason***, run-time norm execution is in practice accomplished at the level of plans to achieve goals, and norms indicate a sort of event that triggers plans. Moreover, in ***N-Jason*** run-time norm compliance is achieved on top of the norm aware decision making and in conjunction with the execution mechanism.

Notwithstanding the benefits of ***N-Jason***, there are some issues to highlight in respect of the mechanism for run-time norms. The norm compliance strategy is hard-coded in the semantics of the language, leaving only a capacity for configuration via the plan annotations, whereas the strategy is programmable through agent plans (i.e. supporting the design of strategy by an agent programmer) in JaCaMo [2] and N-2APL [1]. Thus, the proposal presented here provides a pre-packaged approach to normative reasoning, since it deprives the agent of the scope to change plans dynamically or mis-behave

intentionally, based on rules the agent programmer designs. However, the mechanism put forward here does enable legacy agents, which have no compliance rule or strategy in their specification, to become norm-aware automatically. Thus, those agents' behaviour can be coordinated through the governance of normative frameworks without further engineering effort.

Another issue lies in the simple mechanism for the operationalisation of norms in run-time norm execution. The approach described here means the ontology and syntax of norms that can be executed are limited to those present in the plan library of an agent. In consequence, some detached norms, that may correspond semantically to one of an agent's plans, but which are ontologically different from the plan, will be ignored or violated. We are considering how to generalise the execution mechanism with the analysis of semantics of norms, following [14], in conjunction with plan synthesis.

6 Conclusion and Future Works

In this paper, we have presented a design for a norm-aware BDI agent, ***N-Jason***, that enables the exhibition of norm compliance at run-time. Basically ***N-Jason*** offers a generic norm execution mechanism on top of norm-aware deliberation to contribute to the exploitation of run-time norm compliance. Run-time norm execution specifically focuses on the operationalisation of new and unknown (event-based) norms not stated in the agent program at run-time. By judging the executability of them, ***N-Jason*** agents executes those norms following an extended model of norm awareness consisting of: (i) *event reconsideration*, to find out what the norm is intended to achieve or to reach, and (ii) *option reconsideration*, to identify which plan is the most appropriate in response to the norm. The selection of norm compliant behaviour is achieved in the norm-aware deliberation process by *intention scheduling* with deadlines, priorities and prohibitions which confirms the decision about which behaviour agent would prefer between goals, norms and sanctions. It brings about a *preference maximal set* of intentions in order to realise the norm compliance. ***N-Jason*** is implemented in ***Jason***/AgentSpeak(L) and extends its syntax and semantics to create ***N-Jason***.

We believe that run-time norm compliance model is beneficial for the enhancement of both a norm compliance capability and agent autonomy from the agent's perspective. However, we note that the behaviour triggered by run-time norm execution may look like unpredictable/unwanted behaviour from the agent programmer's perspective.

Although this paper particularly considers the execution of event-based norms at run-time in conjunction with the institutional model, the extension to support state-based norms and its normative systems can easily be incorporated into ***N-Jason*** agents and will be as future work. We also plan to detect violations which are generated in the norm aware deliberation, particularly when the normative goals are dropped during scheduling. This offers a potentially useful link for enforcement in the context of normative system implementation. In addition, both empirical and analytical evaluation of the performance of ***N-Jason*** requires proper investigation.

References

1. Alechina, N., Dastani, M., Logan, B.: Programming norm-aware agents. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, Richland, SC, pp. 1057–1064 (2012)
2. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with jacamo. Sci. Comput. Program. 78(6), 747–761 (2013)
3. Bordini, R., Hübner, J., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley Series in Agent Technology. John Wiley & Sons (2007)
4. Charlton, B.: Evolution and the cognitive neuroscience of awareness, consciousness and language. Cognition 50, 7–15 (2000)
5. Cliffe, O., De Vos, M., Padget, J.: Specifying and reasoning about multiple institutions. In: Noriega, P., Vázquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., Matson, E. (eds.) COIN 2006 Workshops. LNCS (LNAI), vol. 4386, pp. 67–85. Springer, Heidelberg (2007)
6. Dastani, M., Tinnemeier, N.A., Meyer, J.-J.C.: A programming language for normative multi-agent systems. In: Multi-Agent Systems: Semantics and Dynamics of Organizational Models, pp. 397–417 (2009)
7. De Vos, M., Balke, T., Satoh, K.: Combining event-and state-based norms. In: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, AAMAS 2013, pp. 1157–1158. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2013)
8. Dignum, V.: A Model for Organizational Interaction. PhD thesis, Utrecht University (2004)
9. Dybalova, D., Testerink, B., Dastani, M., Logan, B.: A framework for programming norm-aware multi-agent systems. In: Dignum, F., Chopra, A. (eds.) Proceedings of the 15th International Workshop on Coordination, Organisations, Institutions and Norms, COIN 2013 (2013)
10. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
11. Hubner, J.F., Sichman, J.S., Boissier, O.: Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. Int. J. Agent-Oriented Softw. Eng. 1(3/4), 370–395 (2007)
12. Lee, J., Li, T., Padget, J.: Towards polite virtual agents using social reasoning techniques. Computer Animation and Virtual Worlds 24(3-4), 335–343 (2013)
13. Meneguzzi, F., Luck, M.: Norm-based behaviour modification in bdi agents. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2009, vol. 1, pp. 177–184. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2009)
14. van Riemsdijk, M.B., Dennis, L.A., Fisher, M., Hindriks, K.V.: Agent reasoning for norm compliance: A semantic approach. In: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, AAMAS 2013, pp. 499–506. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2013)
15. Vikhorev, K., Alechina, N., Logan, B.: Agent programming with priorities and deadlines. In: The 10th International Conference on Autonomous Agents and Multiagent Systems, Richland, SC, pp. 397–404 (2011)