Making Norms Concrete*

Huib Aldewereld
Universiteit Utrecht
The Netherlands
huib@cs.uu.nl

Sergio Álvarez-Napagao
Universitat Politècnica de
Catalunya, Barcelona, Spain
salvarez@lsi.upc.edu

Frank Dignum
Universiteit Utrecht
The Netherlands
dignum@cs.uu.nl

Javier Vázquez-Salceda
Universitat Politècnica de
Catalunya, Barcelona, Spain
jvazquez@lsi.upc.edu

ABSTRACT

In systems based on organisational specifications a reoccurring problem remains to be solved in the disparity between the level of abstractness of the organisational concepts and the concepts used in the implementation. Organisational specifications (deliberately) abstract from general practice, which creates a need to relate the abstract concepts used in the specification to concrete ones used in practice. A solution for this problem is the use of *counts-as* statements, which, by defining the social reality, provide the concrete concepts their institutional and organisational meaning. Continuing work on the implementation of counts-as to relate abstract and concrete concepts in agent-based systems, this paper investigates the implementation of counts-as statements in DROOLS to relate abstract organisational specifications and its norms to concrete situations.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Theory, Algorithms, Design

Keywords

Normative Systems, Social and Organisational Structure, Environments , Organisations and Institutions

1. INTRODUCTION

Norms in organisational specifications tend to abstract from the concrete events and situations that the norm is supposed to cover. The norms of organisations are intentionally specified at a high level of abstraction to range over many different situations and to require little maintenance over time. While this abstraction creates increased stability over time and flexibility of application for the norms, it also creates a problem when using norms as the abstract concepts in norms need to be related to the concrete events/concepts that occur in the system.

A typical way to solve this gap between normative ontologies and concrete systems is by applying refinement techniques as done in requirement engineering to link the abstract model elements to concrete concepts *at design time*. However, this only solves static cases and cannot cope with dynamic domains where the link between the abstract and concrete concepts changes over time (due to unforeseen circumstances). Moreover, using such methods for bridging the gap does not explicitly state how the concrete and abstract concepts relate, and leaves no information for the system to reason about different implementations (or different contexts) at runtime. In critical domains such as crisis management, it is important to be able to reason about different approaches to solve a crisis. For instance, in the Netherlands a nation-wide set of procedures for crisis management exists called GRIP (Coordinated Regional Incident-Management Procedure). These procedures define the appropriate response to a crisis in terms of the required coordination, control, and flow of information based on the severity of the disaster. There are 4 levels, where GRIP-1 is the lowest, e.g., a large car accident or a small local fire, and GRIP-4 is the highest, e.g., a terrorist attack or large-scale flooding. These kinds of contexts require the ability to reason about different links between the abstract organisational concepts and concrete concepts used in practice (e.g., would scaling from one GRIP level to another provide additional means to solve the crisis).

It has already been proposed that an explicit representation of the links can be given by the concept of counts-as [1, 9]. Based on ideas by Searle [14], counts-as statements add institutional meanings to real-world facts. Although being applied more frequently these days, counts-as relations are often still represented as static links between the institutional facts and the brute facts [1, 4]. While giving an explicit representation of the institutional meaning, thus allowing for reasoning about the links, such static relations do not work well in dynamic domains like the previous mentioned crisis management where the institutional meaning of concepts can change over time, due to a change of context.

In [2] it was investigated how to extend the static im-

*This research has been carried out within the FP7-215890 ALIVE project, funded by the European Commission.

Cite as: Making Norms Concrete, Huib Aldewereld, Sergio Álvarez-Napagao, Frank Dignum, and Javier Vázquez-Salceda, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 807-814

plementation of counts-as with the constitutive elements of counts-as, thus allowing for an explicit representation of the different contexts applicable to the institutional domains. In this implementation the counts-as relation is not only defined within a context, but is also used to define its context.

The counts-as relation can thus be used to connect abstract concepts to more concrete ones. However, this does not yet show how abstract norms should be translated to more concrete ones. In [9] it was proposed to use counts-as statements as a kind of operationalisation of norms. This can be done by expressing that a certain state counts-as a violation of a norm. This leads to a form of Anderson's reduction of deontic logic [3]. For example, by stating that "not doing a payment *counts-as* a violation", one basically expresses that payment is obliged. In this way norms are given an explicit meaning by relating facts (institutional facts or agent actions) to the concept of norm violation (often represented as a static proposition).

Using such a norm representation next to the use of counts-as as a way to relate (system) facts (or events) to agent actions and their effects gives the agent the means to reason about *both* the meaning of their actions in a causal manner (i.e., what their actions bring about in a given context) and about the normative impact of performing those actions in a certain manner (e.g., will doing a specific action make the agent compliant to the normative context). Essentially, it makes explicit the fact that the agent is within two contexts; 1) the normative context, and 2) the institutional context. The normative context determines whether a certain action is normatively correct, while the institutional context determines which specific events (services or basic actions) counts-as performance of the action. E.g. credit-card payment can count-as a payment within the context of 'buying a book at Amazon', but this payment might not be acceptable as fulfillment of an obligation to pay within the context of 'buying a house'.

This work aims to extend the implementation of [2] with respect to normative reasoning. It was shown there that the constitutive part of counts-as adds interesting reasoning possibilities to agent systems; agents could reason about the effects of changing contexts and choose appropriate contexts to achieve their goals. This paper shows that the link from counts-as to the norms adds another interesting reasoning possibility; namely, the ability to determine the normative consequences of actions and the ability to reason about how to act based on the norm specification.

This paper is structured as follows. In the next section we look in more detail at how counts-as is used in agent reasoning. Then in section 3 we briefly discuss the theoretical background of counts-as. In section 4 we discuss how systems with production rules can be used to implement counts-as in general, and in section 5 we discuss how the implementation in DROOLS is done. We end this paper with conclusions in section 6.

2. REASONING WITH COUNTS-AS

In this section we illustrate the benefits to agent reasoning when having explicit links between concrete events and abstract concepts as well as having explicit the relations between these abstract concepts and the norms.

In complex domains like, e.g., crisis management, the meaning of the concrete events can change from context to context. In GRIP-1 scenarios (little severity, for instance, a car crash) the meaning of the institutional fact 'sufficient coordination' means something different than in a GRIP-3 situations (like a flooding). While in the GRIP-1 scenario, one person verbally communicating to coordinate the efforts of the different parties involved (police, medics, firemen) counts as 'sufficient coordination', in more pressing situations a larger coordination structure is required. Moreover, the manner in which the coordination is managed differs between these scenarios; where in the car crash the orders can be done verbally between parties, coordinating in a large scale flooding requires some form of tele-communication (that might be recorded).

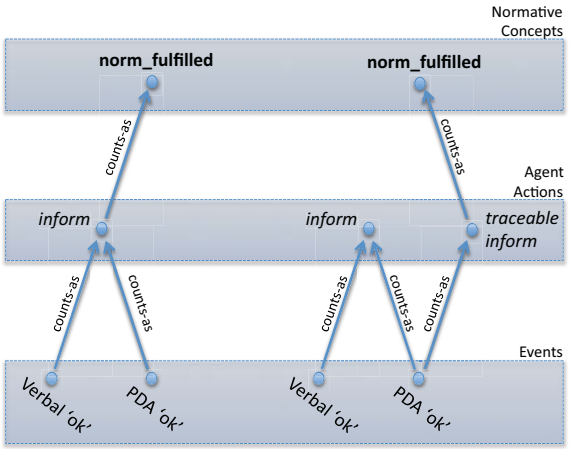

Figure 1: *inform* in a simple scenario (left) and a complex scenario (right).

The above has consequences for the way the people (or agents) have to follow norms dictated by the crisis management procedures. Consider, for instance, the following norm: "Crisis handlers need to inform their superiors through adequate measures". This norm is rather vague in describing that crisis handlers (e.g., police officers or firefighters at the scene of a crisis) need to use the appropriate channels to inform their supervisor. In a simple scenario, like the car accident mentioned earlier, a simple verbal "ok" to the coordinator on site will suffice, but in more complex situations it might be required to keep a log of all communication between the parties involved and a more complex action, such as acknowledging orders from your superior via your PDA might be required. The situation is shown with counts-as relations in figure 1.

In the GRIP-1 scenario, agents can choose between informing their superiors via a verbal acknowledgement or using their PDA to acknowledge the orders given. Either of these events can be classified as doing an *inform* action. Subsequently, it can be derived that doing an *inform* action makes the agent fulfill the norm. In the GRIP-4 scenario, however, this is not the case. While a verbal "ok" to your superior still counts-as an *inform*, the norm specifies that all communication should be traceable, which the verbal inform is not. The normative counts-as relation between the actions and whether the norms are fulfilled has changed in this scenario, and the agents should adapt accordingly.

Note that the classifications on the lower level (between events and actions) can also dynamically change in a do-

main. While in our current example the verbal "ok" does not count as a traceable inform, there are contexts where that event would count as that kind of action, e.g., when the verbal communication is overheard by a (trustable) third-party.

In [2] we have shown that making the links between the abstract and concrete concepts explicit allows the agents to reason about different contexts and changing contexts. By changing contexts the organisational capabilities of the agents are changed, which allows them to use other means to solve the crises; allowing agents to reason about these possibilities adds a dimension. The representation of the ontological links between the institutional facts and brute facts is thus not just for solving the ontological gap between the two levels, it also gives meaning to the different contexts and explains what happens when contexts change. In our scenario of the crisis management an agent might change the context from being a local incident to a regional disaster by starting to communicate through a central control unit. This choice of communication implicitly involves other parties in the disaster which might constitute different counts-as rules and possibly different norms.

Making the normative links explicit adds a similar reasoning capability, as the concepts of norm violation and norm fulfillment get a proper meaning in the system. Agents can use the counts-as rules to derive which actions would or would not violate the norms and thus be able to reason *on forehand* whether a particular action will get the required result (that is, with respect to being norm compliant).

3. ABOUT COUNTS-AS

As mentioned before there is a need to explicitly represent the relations between the abstract and concrete concepts which can be used by agents in their reasoning. Moreover, since these relations between concepts is dependent on the context in which that relation is evaluated, the definition of the context of those relations needs to be explicit as well. This can be done with counts-as statements, which have three different readings. The different readings of counts-as can be summarised in the example seen in table 1 presented below (based on [10]).

"In normative system Γ, happenings with severe consequences to the general safety *count as* disasters"	**Constitutive**
"It is always the case that large scale fires *count as* happenings with severe consequences to the general safety"	**Classificatory**
"In normative system Γ large scale fires *count as* disasters"	**Proper Class.**

Table 1: Three notions of counts-as.

In the example, the counts-as locution occurs three times. However, the three locutions are each of a different nature. The second premise is a (generally acknowledged) contextual classification concerning an universal context. The conclusion is a 'new', proper contextual classification which is considered to hold with respect to the given system. The first premise, however, has a semantic ingredient that is not shared with the other two locutions of counts-as. The first premise is what Searle referred to as the ability to "constitute social reality" [14]. The counts-as defines a context in which that counts-as relation holds. Counts-as has the ability to change the world, not in the sense that it affects physical reality; it makes no sense to express that "children of the age of 3 *counts-as* writers", since 3-year old children are physically unable to write. Stating it does not make them able to. Instead, counts-as adds institutional/organisational semantics to real-world events and concepts (that is, the events and concepts are given meaning in the context of the institution/organisation). Doing so can, however, change the institutional/organisational capability of people. That is, counts-as does not change what people can or cannot do physically, but it does change what people are allowed/entitled to do institutionally.

The rules and norms related to concepts in the social world change with the changes made by counts-as. It is this kind of change that counts-as brings to the world. It creates social facts that determine how situations should happen or how situations should be handled. Thus, the counts-as does not directly influence the world, but it influences the capabilities/rights of roles, and the way these roles interact with each other. Counts-as defines the social (normative) meaning of things; it defines the applicability of the norms on brute (real-world) concepts.

Continuing ideas from [9], one can also apply counts-as locutions for the operationalisation of norms of an organisation. Introducing two primitive normative facts, *norm violation* and *norm fulfillment*, one can represent any arbitrary norm (expressed in a deontic logic) as a counts-as locution. For example:

$$\begin{aligned} Obliged\ A &\Rightarrow \neg A\ counts\text{-}as\ norm\_violation \\ Forbidden\ A &\Rightarrow A\ counts\text{-}as\ norm\_violation \end{aligned}$$

In similar manner, properties of norms (how to achieve norm compliance) can be expressed as counts-as statements as well. For example

$$Obliged\ A \Rightarrow A\ counts\text{-}as\ norm\_fulfillment, \text{ etc.}$$

Naturally, these counts-as expressions can be given either of the three meanings of counts-as as specified above, thereby expressing norm constitution, general norms (which hold in every context), and context-specific norms.

Combining both uses of counts-as, as explicit link between institutional and brute facts and as explicit link between institutional facts and normative facts, gives a double application of counts-as when determining the normative meaning of events:

1. low level events are related to institutional facts, e.g., $Verbal$ 'ok' $counts\text{-}as\ inform$.
2. normative meaning is added, e.g., $inform$ *counts-as* norm_fulfillment

The need for both these counts-as locutions becomes evident when realising that counts-as relations are *not transitive* in general (see also [10]). Because of having to take into account two separate contexts which can change independently of each other, the relations between low-level events and normative states cannot be simplified to just one counts-as. If, however, both counts-as relations would be classificatory, instead of proper classificatory as used in the example presented before, or if one would only reason about one single context, the transitive closure of counts-as relations can be used just as well. In dynamic domains, where

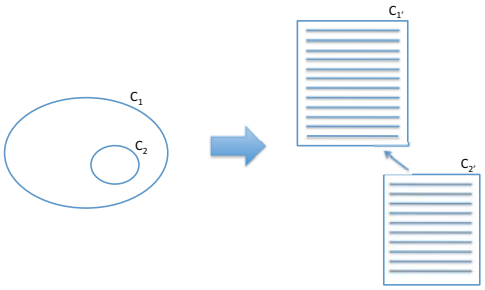

Figure 2: Context subsumption.

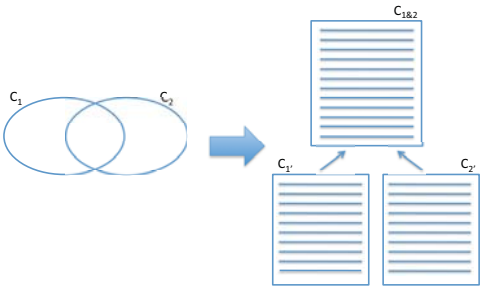

Figure 3: Context overlap.

both the organisational context and the normative context can change over time, an explicit representation of both relations is required.

3.1 Dealing with Sub-contexts and Overlap

In order to be able to implement the different kinds of contextual reasoning in practice, we have implemented all the afore-mentioned aspects of counts-as into DROOLS rules. As we will see in section 5, this allows for ease of use and efficient reasoning by the agents in the system. However there were some issues to be tackled, mainly related with the handling of overlapping and subsuming contexts.

As described above, the constitutive counts-as rules define the social context in which the counts-as holds. This could, in practice, mean a lot of different rules defining a single social context, which could make it problematic (or rather inefficient) to use when comparing different contexts (e.g., in case when an agent wants to decide whether a scale-up from GRIP-2 to GRIP-3 is required). To deal with this inefficiency at runtime, we consider contexts to only have their unique counts-as rules (the rules that are not part of any other context). But this requires a proper handling of the occurrence of context subsumptions (i.e. context A being sub-context of context B) and context overlap (i.e. a non-empty intersection between the scopes of context A and B).

Any domain contains a number of social contexts defined by constitutive counts-as rules as mentioned above. These constitutive counts-as rules define the classifications that *only hold for that context*. Global classifications are considered to be part of the universal context, which subsumes each defined social context (i.e., all defined contexts are a *sub-context* of the universal context). To deal with subsumed contexts and to allow for quick reasoning about what makes contexts unique, we limit the counts-as rules in a context to only those rules which are not contained in any of its *parent-contexts*. Then, by using context inheritance relations we specify that all the counts-as rules that hold in a context are those contained in its specification *and* any contained in the specification of its parents. Figure 2 shows the subsumption of context C_2 by context C_1; for instance, the social context of the GRIP procedures (C_2) being a sub-context of the social context of crisis management organisations (C_1). This basically means that the worlds in the context of GRIP are a 'refinement' of the worlds in the context of crisis management organisation; that is to say, these worlds adhere to both the classifications made by the parent context as well as to the classifications specified by the specific GRIP scenarios. Therefore, in a world in the social context of crisis management organisation, all counts-as rules of $C_{1'}$ apply, but in worlds in the social context of GRIP apply both the counts-as rules from $C_{1'}$ and $C_{2'}$. It is then easy to see that what makes the GRIP context different from the global context by looking at just the rules specified in C'_2.

Similarly, we can deal with overlapping contexts. Take, for example, the different GRIP-levels; each are a specification of the crisis management situation at a different level of severity, but they all contain elements that remain the same between them; e.g., ambulances counts-as means of evacuation in both GRIP-2 and GRIP-3. There are, however, distinctions between the separate levels as well; e.g., army trucks count-as means of evacuation only in GRIP-3, not in GRIP-2. In figure 3 it is visualised how contextual descriptions for C_1 and C_2 are split: a) a new shared, parent context (shown as $C_{1\&2}$ in the figure) that contains all counts-as rules that are shared between contexts C_1 and C_2, b) two distinct sub-contexts (shown as $C_{1'}$ and $C_{2'}$ in the figure) which contain the counts-as rules that make each original context distinct.

By this split it now becomes fairly easy to determine the differences between contexts C_1 and C_2: one looks at the specific rules for each in $C_{1'}$ and $C_{2'}$, respectively. Similarly, it is easy to determine the similarities between contexts C_1 and C_2 by looking at their shared parent-context $C_{1\&2}$.

Using this manner of reasoning with context overlap and context subsumption, we implemented the aspects of counts-as which is described in section 5. First, however, we look at how to implement norms with a production system in general in the next section.

4. GROUNDING NORMS WITH PRODUCTION SYSTEMS

In our approach, practical normative reasoning is based on a production system with an initial set of rules implementing the operational semantics described in [11]. Production systems are composed of a set of rules, a working memory, and a rule interpreter or engine [5]. Rules are simple conditional statements, usually of the form *IF a THEN b*, where *a* is usually called left-hand side (*LHS*) and *b* is usually called right-hand side (*RHS*). Our basic idea is that an agent can configure the production system by adding abstract organisational specifications and sets of counts-as rules.

The implementation of rule-based norm operationalisation has already been explored in previous research [7, 13, 16], but these proposals are either not yet implemented in a real system or not using a high enough level of norm abstraction. Some recent approaches [8] define specific norm-oriented programming languages that treat norms as rules of a production system. However, such an approach requires

for an special production system.

We solve this issue by combining a normative language [12] with a reduction to a representation with clear operational semantics based on the framework in [11]. This framework uses logic conditions that determine the state of a norm (active, fulfilled, violated). These conditions can be expressed in first-order logic and can be directly translated into *LHS* parts of rules, with no special adaptation needed. The implementation of the operational semantics in a production system to get a practical normative reasoner is thus straightforward. This allows agents for dynamically changing its organisational context at any moment, by *feeding* the production system with a new abstract organisational specification.

The detection of normative states is a passive procedure consisting in monitoring past events and checking them against a set of active norms. This type of reasoning is already covered by the declarative aspect of production systems, so no additional implementation in an imperative language is needed. Using a forward-chaining rule engine, events will automatically trigger the normative state - based on the operational semantics - without requiring a design on *how* to do it.

Having 1) a direct syntactic translation from norms to rules and 2) a logic implemented in an engine consistent with the process we want to accomplish, allows us to decouple normative state monitoring from the agent reasoning. The initial set of rules we have defined is the same for each type of agent and each type of organisation, and the agent will be able to transparently query the current normative state at any moment and reason upon it. Also this decoupling helps building third party agents such as observers and/or enforcers.

There are several production system implementations available, some widely used by the industry, such as JESS, DROOLS, SOAR or PROVA. In most of these systems rules are syntactically and semantically similar, so switching from one to the other would be quite simple. As production systems dynamically compile rules to efficient structures, they can be used as well to validate and verify the consistency of the norms.

A prototype of our counts-as normative reasoner has been implemented as a DROOLS program. DROOLS is an open-source Object-Oriented rule engine for declarative reasoning in Java [15], supported by the JBoss Community. Its rule engine is an implementation of the forward chaining inference Rete algorithm [6]. Concepts are imported from standardised Description Logic OWL-DL ontologies into Java objects [17]. The use of Java objects inside the rule engine allows for an easier communication of concepts with the agent reasoning, the core of which is also implemented in Java.

5. DROOLS IMPLEMENTATION

In DROOLS we can represent facts by adding them to the knowledge base as objects of the class *Predicate*. The following shows an example of the insertion of $Mayor(a)$ into the knowledge base to express that a (represented as object a of the domain) is in fact a mayor.

```
ksession.insert(new Mayor(a));
```

The class *Predicate* is designed specifically for our implementation and is the superclass of every predicate in the system. We use this abstraction as a basis to reason about norms with DROOLS. The following sections describes briefly how we deal with norms, and how we have extended our practical reasoner to include counts-as.

5.1 Implementing counts-as

We implement the concept of *Context* as a subclass of *Predicate*, asserting its instances into the knowledge base:

```
ksession.insert(Context.CAR_CRASH);
ksession.insert(Context.FLOODING);
ksession.insert(Context.UNIVERSAL);
```

Defining contexts as concepts in the knowledge base allows us to also refer to them explicitly and reason about them. This is an important advantage over implementations where contexts are mere labels on the counts-as relations between concepts.

In order to define the proper classificatory counts-as in a specific context, the predicate *ClassificatoryCountsAs* is introduced. This predicate allows for the expression of classificatory relations *between classes* with respect to a context.

```
ksession.insert(
new CountsAs(
   VerbalOK.class,
   Inform.class));
new CountsAs(
   PdaOK.class,
   Inform.class));
new CountsAs(
   PdaOK.class,
   TraceableInform.class));
```

Figure 4: Definition of classificatory counts-as rules.

The expressions of Figure 4 show two examples of the classificatory counts-as, where the statements respectively describe that, in the *universal* context, a verbal OK *counts-as* an inform, and a PDA OK *counts as* both an inform and a traceable inform.

Figure 5, on the other side, shows the proper classificatory counts-as for the example. In this case, each counts-as refers to different contexts: an inform *counts-as* a proper inform in a car crash scenario, but in a flooding scenario a traceable inform *counts-as* a proper inform.

```
ksession.insert(
new ClassificatoryCountsAs(
  Inform.class,
  ProperInform.class,
  Context.CAR_CRASH));
new ClassificatoryCountsAs(
  TraceableInform.class,
  ProperInform.class,
  Context.FLOODING));
```

Figure 5: Definition of proper classificatory counts-as rules.

To implement the uniqueness criterium specified in subsection 3.1, which allows for more efficient runtime use of the counts-as rules, we implemented a DROOLS program to create internal parallel sets of contexts (based on the intuitions expressed in figures 2 and 3 in section 3.1). The first rule of figure 6 shows how this splitting is done, while the second rule of figure 6 gives an example of how one can identify in which (original) context a counts-as rule was formulated.

```
rule "creation of running contexts"
  when
    ClassificatoryCountsAs(a : c1, b : c2)
    and
    lc : TreeSet() from collect(
      ClassificatoryCountsAs(c1 == a, c2 == b))
  then
    RunningContext rc;
    rc = new RunningContext(lc);
    insertLogical(rc);
end

rule "identify running contexts"
  when
    cca : ClassificatoryCountsAs(c : context)
    and
    rc : RunningContext(countsas contains cca)
  then
    insertLogical(
      new RunningContextIdentifier(rc, c));
end
```

Figure 6: Context splitting.

Figure 7 then shows an example of the context splitting. From three counts-as rules, of which two of them are the same for two different contexts, the result will be two contexts.

```
ksession.insert(
  new ClassificatoryCountsAs(
    Ambulance.class,
    MeansOfEvacuation.class,
    Context.GRIP2));
ksession.insert(
  new ClassificatoryCountsAs(
    Ambulance.class,
    MeansOfEvacuation.class,
    Context.GRIP3));
ksession.insert(
  new ClassificatoryCountsAs(
    ArmyTruck.class,
    MeansOfEvacuation.class,
    Context.GRIP3));

(after kession.fireAllRules())

[GRIP2GRIP3, GRIP3]
```

Figure 7: Example of context splitting.

The first rule of the example expresses that ambulances count as a means of evacuation in the context of GRIP-2; the second expresses that ambulances also count as a means of evacuation in the context of GRIP-3; the third rule expresses that in the context of GRIP-3 army trucks also count as a means of evacuation. After the splitting of contexts GRIP-2 and GRIP-3 containing just these three rules we end up with two contexts, namely the context which contains the rules that are present in both GRIP-2 and GRIP-3 (ambulances count as means of evacuation), and the context which gives the refinement of being in context GRIP-3, namely that army trucks also count as means of evacuation. The result being the `GRIP2GRIP3` context containing the rule about ambulances being evacuation means (now unique, as there is no need to specify it twice) and the `GRIP3` context containing only the rule specifying that army trucks are evacuation means. As explained in subsection 3.1, this split allows for an easy and efficient means to check the similarities and differences between the contexts GRIP-2 and GRIP-3[1].

```
rule "activate running contexts"
  when
    ContextActive(c : context)
    and
    RunningContextIdentifier(
      rc : runningContext, context == c)
  then
    insertLogical(new RunningContextActive(rc));
end

rule "classificatory counts-as"
  when
    rc : RunningContextActive(
      ca : ClassificatoryCountsAs(
        y1 : c1, y2 : c2))
    and
    obj : Predicate(class == y1)
  then
    insertLogical(new CountsAs(y1, y2));
end

rule "counts-as"
  when
    c : CountsAs(y1 : c1, y2 : c2)
    and
    obj : Predicate(class == y1)
  then
    Predicate instance;

    instance = (Predicate)(
      ((Class)y2).newInstance());
    instance.setObject(obj.getObject());
    insertLogical(instance);
end
```

Figure 8: Activation of counts-as rules.

The internal effect of a context activation is the activation of all its shared contexts (see Figure 8). With the contexts active, their counts-as rules will be instantiated as active counts-as rules in the rule engine. The counts-as rules are fired whenever there is a matching predicate. The effect of a fired counts-as rule is that for each instance of the first predicate of the rule, a new instance of the second predicate of the rule is created.

Closure is provided in the program by automatically detecting which context should be active based on the active counts-as rules. Figure 9 shows the rules implemented for this purpose. The first rule detects if all the proper classificatory counts-as rules for a certain shared context are instantiated, in which case that shared context will be activated automatically. The second rule checks if all the shared contexts that belong to a user defined context are active, in which case the context will be activated.

By using these rules we can identify the concept of a context (like GRIP-2) with the counts-as rules related to that

[1]Note that in this example GRIP-3 ended up as a subcontext of GRIP-2 because of the limited scope of the example. In reality, there are other differences between these contexts which would show that they instead overlap.

```
rule "activate running context"
  when
    rc : RunningContext(cal : countsas)
    and
    forall(
      ca : ClassificatoryCountsAs(a : c1, b : c2)
      from cal
        CountsAs(c1 == a, c2 == b))
  then
    insertLogical(new RunningContextActive(rc));
end

rule "activate context by its running contexts"
  when
    c : Context()
    and
    forall(
      RunningContextIdentifier(
      rc : runningContext, context == c)
        RunningContextActive(runningContext == rc)
    )
  then
    insertLogical(new ContextActive(c));
end
```

Figure 9: Automatic activation of contexts.

context. Having this constitutive relation between a context and the counts-as rules available we can now also handle the following scenario of the crisis management.

Suppose the hospital has to be evacuated due to a flood. There are not enough ambulances available to evacuate all people in time. The commander (chief medic at the location) checks to see what can be done. He can use (special) army trucks. However, army trucks do not (in general) count-as ambulances. The commander can check (with the DROOLS implementation) that army trucks count-as ambulances in the context of GRIP-3. (They are part of constituting GRIP-3). So, the commander decides to move to the context of GRIP-3. Now he has to check what other rules constitute GRIP-3. One of them states that in GRIP-3 the mayor counts-as commander. This means that the commander has to transfer his command to the mayor. Moreover, in a flooding scenario, as stated previously, only a traceable inform counts as an appropiate inform. That means that all agents should be aware of the new context and act accordingly, being forced to adapt to a traceable informing mechanism if they were not using it.

The scenario shows that we need the context as an explicit concept and also we need the constitutive aspect of the counts-as rules that define the context in order for the commander to be able to define a switch to another context (GRIP level) and realizing the consequences of this switch. The DROOLS implementation presented above enables us to do this.

5.2 From norms to rules

Norms in our abstract organisational model use a formalism similar to the one described in [12]. Each norm is represented as a dyadic deontic statement with multiple conditions: $O_{a,b,c,x}(\alpha)$ for obligations, and
$F_{a,b,c,x}(\alpha)$ for prohibitions where a, b, c are the three conditions, respectively: activation condition, maintenance condition and expiration condition; x is the role or actor abiding to the norm; and α is the norm target.

The norm used as an example in Section 2 would be formalised as: $a : emergencyCondition(location)$
$b : emergencyCondition(location)$
$c : done(x, properInform(emergencyCondition(location)))$
$\alpha : properInform(emergencyCondition(location))$
$x : Agent$

As described in Section 4, norms are parsed from the abstract organisational model. The conditions of the deontic statements of these norms undergo a syntactic translation to *LHS* parts of rules, while the *RHS* parts are based on the operational semantics [11]. For example:

```
rule "Norm12_expiration"
when
  Done(X : player, action == I)
  and
  I : ProperInform(content == E)
  and
  E : EmergencyCondition(L : location)
then
  Substitution s = new Substitution();
  s.add(new Value("X", X));
  s.add(new Value("I", I));
  s.add(new Value("E", E));
  s.add(new Value("L", L));
  insertLogical(new HoldsExpiration("Norm12", s));
end
```

The expiration condition of the norm is translated to the *LHS* part, while at the *RHS* the predicate *HoldsExpiration*, which is part of the operational semantics, is added to the knowledge base with a grounding of the variables s. The type *Substitution* is a subclass of *Set<Value>* overriding some methods so that when comparing two instances of *Substitution* we can evaluate if they pertain to the same instantiation of the grounding.

These rules are consistent with our definitions, but are not grounded to low level events representing brute facts. In order to solve this, we have extended this implementation with the counts-as operator. The following rules express whether a norm is fulfilled or violated according to the definitions in Section 3:

```
rule "obligation counts-as"
when
  n : Norm(type == Norm.OBLIGATION_TYPE)
  and
  ni : NormInstance(norm == n)
then
  Context c = new Context(ni.hashCode());
  insertLogical(c);
  insertLogical(
    new ClassificatoryCountsAs(
      IsTrue.class, Fulfilled.class, c));
  insertLogical(
    new ClassificatoryCountsAs(
      IsFalse.class, Violation.class, c));
end

rule "prohibition counts-as"
when
  n : Norm(type == Norm.PROHIBITION_TYPE)
  and
  ni : NormInstance(norm == n)
then
  Context c = new Context(ni.hashCode());
```

```
  insertLogical(c);
  insertLogical(
    new ClassificatoryCountsAs(
      IsTrue.class, Violation.class, c));
  insertLogical(
    new ClassificatoryCountsAs(
      IsFalse.class, Fulfilled.class, c));
end
```

where $IsTrue()$ and $IsFalse()$ are implementations of the predicates defined in [11] that express whether the target of the norm associated with a certain norm instance holds true or false. The statements produced by these rules are thus read as *'In the context of a certain instantiation of a norm, the fact of its norm target being true/false counts as a norm fulfillment/violation'.*

6. CONCLUSIONS AND FUTURE WORK

In this paper we have shown a concrete implementation of abstract norms that can be used by agents to reason about norm compliance. One of the key issues of connecting norms to concrete actions to be taken is that we need (at least) two uses of the counts-as relation. One to connect brute facts (or events) to institutional states and actions. The second one to connect these institutional facts and actions to their normative interpretation. We have also shown that the context of the norms and the counts-as relation is important in the type of applications that we are using this framework for. Thus we cannot suffice with a pure classificatory implementation of the counts-as relation, which would be straightforward. Instead we have to use an implementation that takes the context of the norms and the counts-as relation into account. One of the consequences of using this proper classificatory version of the counts-as relation is that it is no longer transitive. We have to check whether contexts change between the rules in order to know whether they can be combined.

We have shown how the above requirements have been met by the implementation in DROOLS. In DROOLS we can explicitly connect the context to the counts-as rules and use it as a constraint on its use. Agents can make use of the DROOLS engine to reason on what specific course of action they should pursue in order to comply to the norms that are imposed by the context they are in. However, they can do more than just that. They can also reason about the consequences of changing the context. In the example, the commander has to the ability to change the context by informing the mayor about the disaster. Of course this action of the commander itself can be regulated by norms that state that the mayor can only be involved if the disaster gets too big (according to some criteria).

As future work we will look at the connection of this implementation of norms and counts-as to the reasoning agents implemented on the AgentScape platform. Already some work has been done on the integration of normative reasoning in the agent planning on this platform. However, this work assumes the norms to be expressed in terms of concrete actions that can be performed directly on the platform. We will extend this framework to make use of the counts-as relation and the reasoning about the context.

7. REFERENCES

[1] H. Aldewereld. *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols.* PhD thesis, Universiteit Utrecht, June 2007.

[2] H. Aldewereld, S. Álvarez-Napagao, F. Dignum, and J. Vázquez-Salceda. Engineering social reality with inheritence relations. In *Engineering Societies in the Agents World X (ESAW 2009)*, LNAI 5881, 2009.

[3] A. Anderson. A reduction of deontic logic to alethic modal logic. *Mind*, 67:100–103, 1958.

[4] M. Dastani. Normative multi-agent organizations. In *Engineering Societies in the Agents World X (ESAW 2009)*, LNAI 5881, 2009.

[5] R. Davis and J. King. An overview of production systems. *Stanford Artificial Intelligence Laboratory, Report No. STAN-CS-75-524*, Jan 1975.

[6] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[7] A. García-Camino, P. Noriega, and J. A. Rodríguez-Aguilar. Implementing norms in electronic institutions. *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, Jan 2005.

[8] A. García-Camino and J. Rodríguez-Aguilar. Constraint rule-based programming of norms for electronic institutions. *Autonomous Agents and Multi-Agent Systems*, Jan 2009.

[9] D. Grossi, H. Aldewereld, J. Vázquez-Salceda, and F. Dignum. Ontological aspects of the implementation of norms in agent-based electronic institutions. *Computational and Mathematical Organization Theory*, 12(2-3):251–275, 2006.

[10] D. Grossi, J.-J. Ch. Meyer, and F. Dignum. Counts-as. classification or constitution? an answer using modal logic. In L. Goble and J.-J. Ch. Meyer, editors, *Proceedings of the Eight International Workshop on Deontic Logic in Computer Science (DEON'06).* Springer-Verlag, 2006.

[11] N. Oren, S. Panagiotidi, J. Vázquez-Salceda, and S. Modgil. Towards a formalisation of electronic contracting environments. *Proceedings of the Workshop on Coordination, Organizations, Institutions and Norms -COIN@AAAI08-, at AAAI 08, Chicago, USA*, 2008.

[12] S. Panagiotidi, J. Vázquez-Salceda, S. Alvarez-Napagao, S. Willmott, and R. Confalonieri. Intelligent contracting agents language. *Proc. of the Symposium on Behaviour Regulation in Multi-Agent Systems -BRMAS'08-, Aberdeen, UK*, Jan 2008.

[13] A. Paschke, J. Dietrich, and K. Kuhla. A logic based sla management framework. *4th Semantic Web Conference (ISWC 2005)*, Jan 2005.

[14] J. Searle. *Speech acts. An essay in the philosphy of language.* Cambridge University Press, 1969.

[15] JBoss Community. JBoss drools business rules, http://www.jboss.org/drools.

[16] J. Vázquez-Salceda and S. Alvarez-Napagao. Using soa provenance to implement norm enforcement in e-institutions. In *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, LNCS 5428, 2009.

[17] M. Zimmerman. OWL2Java, http://www.incunabulum.de/projects/it/owl2java.