Engineering Social Reality with Inheritance Relations[*]

Huib Aldewereld[1], Sergio Alvarez-Napagao[2],
Frank Dignum[1], and Javier Vázquez-Salceda[2]

[1] Universiteit Utrecht, The Netherlands
{huib, dignum}@cs.uu.nl
[2] Universitat Politècnica de Catalunya, Barcelona, Spain
{salvarez, 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 the practice. The prevailing solution for this problem is the use of *counts-as* statements. However, current implementations of counts-as view the relations expressed in this notion as static ontological classifications, which presents problems in dynamic environments where the meaning of abstract concepts can change over time. This limitation has already been solved in complex formal theoretical investigations, but the results of that study are far too complex to make a practical implementation. This paper investigates the limitations of current implementations of counts-as, and proposes a more flexible implementation based on the use of inheritance relations.

1 Introduction

A common problem in the design and implementation of complex systems (be it multiagent systems [6, 16] or service-based systems [7]) is the fact that specifications of the organisation of the system generally abstract from actual practice. This creates a distinct gap between the ontology of the organisation (containing abstract concepts such as "means of transport") and the ontology of the implementation (containing concrete concepts such as "trucks"; often domain and/or implementation dependent).

A typical way to solve this is by applying refinement techniques as done in requirement engineering (see, e.g., [17]) to link the abstract model elements to concrete concepts *at design time*. However, this only solves static cases and

[*] This work has been performed in the framework of the FP7 project ALIVE IST-215890, which is funded by the European Community. The author(s) would like to acknowledge the contributions of his (their) colleagues from ALIVE Consortium (http://www.ist-alive.eu)

cannot cope with dynamic domains where the link between the abstract and concrete concepts changes over time (due to unforeseen circumstances). Moreover, using such a method 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 and thus requires the ability to reason about different links between the abstract organisational concepts and concrete concepts used in practice.

It has been proposed already (e.g., see [1, 9, 10]) that an explicit representation of the links between the abstract and concrete concepts can be given by the concept of *counts-as*. The intuition of counts-as as subsumption relation (that is, as a relation pertaining to ontologies) to solve this problem was first stated in [11].

> "There are usually constraints within any institution according to which certain states of affairs of a given type count as, or *are to be classified as*, states of affairs of another type." [11]

The notion of counts-as expressed above is limited to a classificatory view of counts-as. However, the notion of counts-as is far richer than that of an ontological subsumption relation. Early studies of counts-as describe it as being used as a "constitution of social reality" [13], which, in essence, means that counts-as statements *define*, or establish, the appropriate context in which the organisation must act. In this view, a set of counts-as statements defines an institutional frame.

> "[...] 'institutions' are systems of constitutive rules. Every institutional fact is underlain by a (system of) rule(s) of the form "X counts as Y in context C"." [13]

Thus the fact that an *army truck* counts-as a *means of public transport* is not always true, but only in the context of a large scale evacuation being carried out. In fact, it can be one of the constitutive rules that defines what a *large scale evacuation* is (that is, the fact that army trucks are being used means that the evacuation must be a large-scale evacuation). This shows two additional aspects of counts-as. The aspect of something being constituted, that is, something as being the *result of constitution*, and the aspect of something *constituting* a context. All of these notions of counts-as have been formally investigated by [9] and given precise semantical senses.

When designing agent systems for complex and dynamic applications such as crisis management support (or simulation), these counts-as relations play a crucial role in the design and running of the system. While the high level specification of the organisation of the system should stay stable throughout the life cycle of the system (and thus be as abstract as possible), the actual implementation of the MAS should be flexible and adapt to changing environments and contexts. Agents should know that army trucks are not available as means of transport when a burst water pipe floods a street, but are available when the context becomes that of a large scale evacuation after a dam has been breached.

So, this type of systems should incorporate not only the classificatory part of the counts-as relation, but also should implement at least some of the constitutive elements. The constitutive part of counts-as, however, has only been researched from a theoretical (mostly formal) point of view so far; no implementations that can be used in runtime systems exist. In this paper, we explore the possibility of implementing *all* aspects of counts-as in the DROOLS rule engine.

To elucidate the need of these additional elements of counts-as we start this paper in section 2 with a brief description of the ALIVE project and a discussion of the kind of dynamic domains the ALIVE project aims to cater to. In ALIVE, the goal is to create systems that organise services in dynamic contexts to serve a specific goal or (organisational) objective. The fact that the context of the system changes at runtime means that the relations between the abstract and concrete concepts cannot be fixed on forehand, as the relations between the abstract and concrete concepts can change during the run of the system. It is also not possible to define all the different relations for all the different contexts on forehand, because not all the necessary information about all the different contexts is available. A more flexible approach is required, and counts-as provides just that. In section 3 we proceed by briefly investigating the different meanings of *counts-as* which are used as a basis for the implementation presented in section 4. Moreover, section 4 contains some considerations about how the constitutive aspects of counts-as are to be used in solving the problem of dynamic domains as encountered in the ALIVE project. We end this paper with conclusions and considerations for future extensions.

2 The ALIVE Project

The research presented in this paper is carried out within the ALIVE project. ALIVE aims to apply organisational theory to the design and implementation of software systems. The main focus of the project is to create complex systems based on the composition of (existing) services, through the addition of levels of abstraction. The advantage of added levels of abstraction to the design process of systems is two-fold: 1) it is often more intuitive to think in organisational structures and interactions while designing complex interactions for services, and the addition of the layers of abstraction allows for a gradual (fluent) transition from the system as foreseen to the actual implementation; 2) when changes happen in the environment (e.g., specific services become unavailable) the added levels of abstraction act as an explicit representation of the conceptual steps made at design, thus giving additional information on why certain interactions are as they are, which enables the system to dynamically cope with the changes. To this extend the project attempts to create a framework for software and service engineering through the combination of the latest in coordination and organisation mechanisms and model driven design. The layers of abstraction introduced by the project are the following (from bottom to top).

– The *Service Layer* augments and extends existing service models with semantic descriptions to make components aware of their social context and of the rules of engagement with other services.
– The *Coordination Layer* provides the means to specify, at a high level, the patterns of interaction between services, using a variety of powerful coordination techniques from recent research in the area.
– The *Organisation Layer* provides context for the other levels – specifying the organisational rules that govern interaction and using recent developments in organisational dynamics to allow the structural adaptation of distributed systems over time.

Adding layers of abstraction to the design process of systems, however, also creates a major problem. The different layers are not necessarily specified on the same level of abstraction which means that each are specified in terms of a different ontology. These different ontologies do not match, and in order to link the specifications of one level to the next, the ontologies have to be related somehow.

The organisational layer uses an OperA-like formalisation for the specification of the organisational structures and interactions [6]. These structures are to be used at the Coordination Layer to be transformed into coordination plans and workflows. A possible implementation of the Coordination Layer is through the use of a multi-agent system. This solution has its advantages in that the connection between multiagent systems and organisational structures has been researched to some extent already (e.g., see [6, 15]); that is to say, agents can be created on the basis of the organisational specification and designed in such a way that they comply to that specification (as proposed in, e.g., [15]). Using agents on the Coordination Layer has another advantage in the fact that agents are autonomous and can be equipped with the means to create elaborate plans to achieve pre-set goals/objectives (e.g., through the use of multiagent planning mechanisms such as TÆMS [5, 12]). The link from the Coordination Layer to the Services and Service Layer can be achieved through service invocations and the design of tools for the assistance of service composition (to allow for more complex service calls and workflow enactment). The specific invocations of services or service workflows are done by the agents playing a role in the organisation.

Although this implementation via an agent system on the coordination layer (as intermediary between the organisational specification and the service practice) seems intuitive and attractive, it does not circumvent the mentioned problem of disparity between the abstract organisational concepts and the concrete (service-based) concepts. While, instead of the need to link three different levels, the problem can be reduced to linking just two levels (the agents can be programmed with either an ontology that closely matches the organisational ontology, or with an ontology close to the service one), assuming that the agents automatically and autonomously come up with the solution to bridge the gap between the abstract and concrete ontologies is unfeasible.

Bridging the gap between the abstract and concrete concepts is not the only issue, however. If that was the case, a static solution linking the two levels would suffice. However, ALIVE aims to deal with dynamic runtime elements as well, which require the links between the concepts to change *during the run of the system*. To give a concrete impression of the kind of dynamic organisations and environments that the ALIVE project aims to deal with, we briefly discuss one of the ALIVE use-cases. This use-case scenario, situated in the domain of Crisis Management, will serve as an example throughout the latter sections of the paper.

2.1 Crisis Management Scenario

One of the domains used by the ALIVE project is situated in the field of crisis management, in particular dealing with the handling of disasters in the Netherlands. In a densely populated country like the Netherlands where the threat of flooding is matter-of-fact, regional and nation-wide organisation of the management of crises was required. Crisis management in the Netherlands is organised through a nation-wide agreement on procedures called the GRIP. GRIP stands for "Gecoordineerde Regionale Incidentbestrijding Procedures", which translates to Coordinated Regional Incident handling Procedures. GRIP describes the required organisational and management needs for different levels of incidents. The five different GRIP levels are the following (in increasing severity).
GRIP-0 Routine accidents.
GRIP-1 Incidents.
GRIP-2 Large scale incidents.
GRIP-3 Disasters concerning multiple regions.
GRIP-4 Large scale disasters.

GRIP-0 handles about normal (traffic) accidents where no real coordination is required (coordination between incident handlers is done ad hoc). GRIP-1 to 4 are the real incident levels, ranging from small incidents that only have an effect in the immediate region of the incident (GRIP-1), or that have (apparent) effect regions of a nearby city (GRIP-2), nearby cities (GRIP-3), or even multiple provinces and/or the whole nation (GRIP-4). The scaling from one level to another happens in accordance with the severity of the incident (the range of the apparent affected region is wider because of changes in the incident or because the incident was more/less severe than anticipated) or because the organisational infrastructure of a higher level is deemed required to solve the incident.

Instead of using all the different GRIP levels to illustrate our approach we will focus, due to space limitations, on GRIP levels 2 and 3. The important aspect of the GRIP scenario (and which is covered by our limited scope of just viewing levels 2 and 3) is that the domain and organisation are of a dynamic nature. Therefore, the examples will show that static counts-as relations (as modelled by [1, 11]) do not suffice and a more complex implementation is required.

3 The Intuitions of Counts-as

Due to the dynamic nature of the domains used by the ALIVE project, it is not sufficient to use static references between the abstract concepts in the organisational specification of the system and the concrete concepts used by the services implementing the system. Contexts change during the run of the system, and the relations between the abstract concepts and the concrete concepts need to change with them. While in a large scale incident (GRIP-2) ambulances would be used to evacuate a hospital that is being threatened to be flooded, in disaster situations (GRIP-3) the crisis management can call upon the army to assist, deploying army trucks to evacuate the hospital instead.

As mentioned in section 1, in these domains there is a need to explicitly represent the relations between the abstract and concrete concepts which can be used by agents in their reasoning. The need for an explicit representation is the first argument for using counts-as to bridge the gap instead of using refinement techniques from requirements engineering instead. But there is a second argument for using counts-as relations as well. Let us first look at the nature of counts-as before we further explain this second argument.

The different readings of counts-as can be summarised in the example seen in table 1 presented below (extracted from [9]).

"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 (and can thus be formalised as ontological subsumption as done in, e.g., [1, 11]). The conclusion is a "new", proper contextual classification which is considered to hold with respect to the given system (this requires the extension to a context dependent counts-as as attempted in, e.g., [4, 10]). But what about the first premise? The semantic ingredient of the first premise is not captured by either notions of counts-as; it is neither a contextual nor a proper contextual classification.

The first premise of table 1 is not classificatory of nature, but is what Searle referred to as the ability to "constitute social reality". 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 the physical reality; it makes no sense to express that "children at the age of 3 *counts-as* writers", since 3-year old children are physically unable to write. Stating that they can 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 capabilities 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. For instance, a normative system that states that "a coordinator has executive command in crisis situations" (a norm of the system) can change the organisational capabilities of a police officer in GRIP-1 because in GRIP-1 "the first police officer at the scene *counts-as* the crisis coordinator" (and again at GRIP-2 where it holds that "the mayor *counts-as* the crisis coordinator").

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.

This constitutive aspect is the second argument on using counts-as rules for the specification of the links between the abstract and concrete concepts in complex and dynamic applications. In next section we will see how applying the constitutive aspect of counts-as would bring important advantages in the crisis management domain.

3.1 Constitutive Counts-As in Crisis Management

In the crisis management domain it is important that agents are able to reason about what different contexts would bring; for instance, reasoning about whether a change from GRIP-2 to GRIP-3 would allow for better solutions to solve the crisis in that the army becomes available as evacuation means. Moreover, the constitutive aspect of counts-as allows agents to change the context to affect the crisis solving. In short, constitutive counts-as allows the agents to *dynamically* change the social reality of the system, thus enabling them to scale from one GRIP-level to another when required.

This was already evident from our earlier example; while army trucks are means of transportation, they are generally not considered to be public transport, except in the case they are used in a large scale evacuation. This change in meaning of the concept public transport (in GRIP-2 it does *not* include army trucks, in GRIP-3 it *does*) has an impact on the planning possibilities of the agents in the system. The change of environment enables new (previously unavailable) means to add to their plans.

Likewise, domain restrictions change their impact on the creation of plans when the environment changes. While the abstract specification of the restrictions in the domain (both expressed in norms and in organisational specifications) remains stable over time, the dynamics of the environment (the changing of contexts) impacts the *interpretation* of those restrictions; that is, the norms and organisational specification remain fixed for all situations, but their application changes due to changes in the meaning of the abstract concepts used to express them. For example, a domain restriction could be that all agents are under the authority of the operational commander (and obliged to obey his commands). The role of operational commander can change from one GRIP to another, which also means that the organisational structure, and the rights and authorities of the agents involved changes between contexts. See [2] for a further elaboration of this kind of organisational and normative dynamics.

The constitutive elements of the counts-as comes into play in the definition of the contexts in an environment. These definitions of contexts play a major role in determining the current context. During the run of a system in a dynamic domain such as the crisis management scenario, context changes can happen. These context changes happen for two different reasons:

1. An agent in the system with the appropriate rights/power decrees that a different context is of effect.
2. The situation at hand does not conform to the constitutional definition of the current context, and a switch to another context is required.

The former kind of change has to do with notions of power and speech acts [13, 11, 3]. The agent with this function has the ability to create or change bridge rules in the normative system, i.e., the utterance "we can now use army trucks for evacuation" made by this agent constitutes that army trucks count as public transport and therefore affects an (indirect) change to GRIP-3.

The latter change can be observed in the environment and constitutes a kind of external trigger for changing the context. E.g., when a flooding extends the city limits it violates the proper classificatory counts-as part of the GRIP-2 definition, and the situation scales automatically to GRIP-3.

3.2 Dealing with Sub-contexts and Overlap

In order to be able to implement the above 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 4, 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-void 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

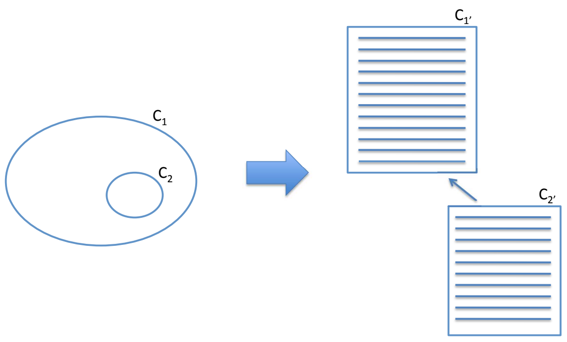

Fig. 1. Context subsumption.

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 1 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

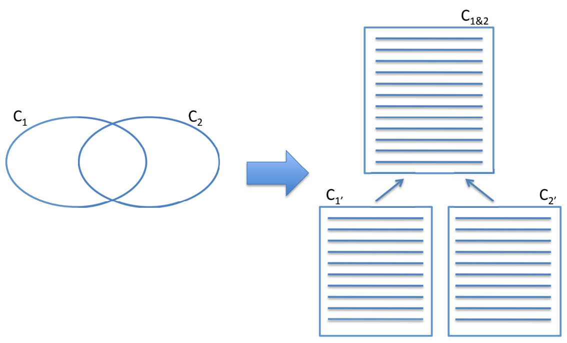

Fig. 2. Context overlap.

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 2 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 $C_{1\&2}$.

In the next section we show how this can be implemented in DROOLS.

4 Implementing Counts-as

A prototype of the counts-as has been implemented as a DROOLS rule file. DROOLS is an open-source Java-based rule engine for declarative reasoning (supporting reasoning based on the standardised Description Logic OWL-DL) [14]. Its rule engine is an implementation of the forward chaining inference Rete algorithm [8].

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 *obj*3 of the domain) is in fact a mayor.

```
Object obj3 = new Object();
ksession.insert(obj3);
ksession.insert(new Mayor(obj3));
```

The class *Predicate* is designed specifically for the prototype and is the superclass of every predicate in the system. We also defined the sub-class *Context* which means that contexts can be asserted to the knowledge base in a similar way:

```
ksession.insert(Context.GRIP2);
ksession.insert(Context.GRIP3);
```

Defining contexts as concepts in the knowledge base allows us to also refer to them explicitly and reason with 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 ClassificatoryCountsAs
          (ChiefFire.class, OperationalCommander.class, Context.GRIP2));
ksession.insert(new ClassificatoryCountsAs
          (Mayor.class, OperationalCommander.class, Context.GRIP3));
```

The expressions above show two examples of the classificatory counts-as, where the first statement expresses that the commander of the fire brigade counts as the operational commander in GRIP-2, while the second statement expresses that in GRIP-3 the mayor counts-as operational commander, instead.

To implement the uniqueness criterium specified in subsection 3.2, which allows for more efficient runtime use of the counts-as rules, we implemented the DROOLS rule file to create internal parallel sets of contexts (based on the intuitions expressed in figures 1 and 2 in section 3.2). The first rule of figure 3 shows how this splitting is done, while the second rule of figure 3 gives an example of how one can identify in which (original) context a counts-as rule was formulated.

The following 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));

[...]

[GRIP2GRIP3, GRIP3]
```

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

```
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
```

Fig. 3. Context splitting.

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 3.2, this split allows for an easy and efficient means to check the similarities and differences between the contexts GRIP-2 and GRIP-3[3].

The internal effect of a context activation is the activation of all its shared contexts (see Figure 4). 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 rules file by automatically detecting which context should be active based on the active counts-as rules. Figure 5 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

[3] 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 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
```

Fig. 4. Activation of counts-as rules.

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 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 flooding. 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

```
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
```

Fig. 5. Automatic activation of contexts.

GRIP-3 the mayor counts-as commander. This means that the commander has to transfer his command to the mayor.
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 Conclusions

Counts-as statements play a crucial role in the design of agent systems for complex and dynamic applications. They are needed to create the links between the abstract system specification and the concrete practice. These links created between the abstract and concrete ontologies are context dependent.

This paper presents an implementation of the notion of constitutive counts-as and proper classificatory counts-as as a means to solve this gap in dynamic domains. We considered all intricate theoretical aspects of counts-as in the implementation, and the implementation is made in such a way that it allows for reasoning about the differences between contexts and the effects of (possible) context changes. The implementation in DROOLS is expressive enough to create the necessary tools for the definition of contexts and the specification of context-dependent relations which are needed in complex and dynamic domains.

An advantage of the Drools implementation is that it allows reflection on the counts-as statements, thus allowing for agents (with sufficient capabilities to do so) to reason about what changes between contexts, and even to reason about whether changing a context is a valid option to achieve their (organisational) objective(s). That means, the implementation of counts-as presented in this paper is such that it not only represents the links in different contexts, but also allows reasoning about the effects of context change.

Drools, in its last version, is an integrated platform supporting *rules*, *workflows*, and *events*, fully adapted to object-orientation and working under Java and .NET. Its rule engine is an implementation of the forward chaining inference Rete algorithm. The knowledge base is dynamic, supporting the addition and removal of facts and rules at runtime, and it works as a truth maintenance system with logical assertions, supporting Predicate and First-Order Logic.

Drools is nowadays the most powerful and efficient open-source rule engine, with a strong support from the community. One of the advantages of using Drools for our implementation is the possibility of contributing our results back to be included in further releases of the platform.

We are aware of only a few similar approaches towards the use of context-dependent ontological subsumption rules (i.e., [10] and [4]) that could also be used to solve the dynamic problem ontologically. While these approaches achieve similar results in the specification of ontological relations that can change depending on the current context, neither of these take any note of the definition of contexts themselves. That is, the constitutive aspect of the relation is overlooked and is not implemented. To our knowledge, this is the first attempt in implementing the constitutive aspect of counts-as.

For our current application our implementation has all properties that were needed to define contexts and be able to reason when and why to change between contexts. Although the Drools implementation stays quite close to the theoretical work on the counts-as some "short-cuts" had to be taken. Most notably is that contexts are defined as concepts, on the same level as "domain" concepts. Of course, the idea is that they will not be used in the same way as other concepts, because this might lead to circular definitions of contexts. As future work we will look more thoroughly at the logical properties that still can be derived using this implementation.

References

1. H. Aldewereld. *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols.* PhD thesis, Universiteit Utrecht, June 2007.
2. H. Aldewereld, F. Dignum, L. Penserini, and V. Dignum. Norm dynamics in adaptive organisations. In G. Boella et al., editor, *Proc. of the 3rd Int. Workshop on Normative Multiagent Systems (NorMAS 2008)*, 2008.
3. J. L. Austin. *How to Do Things With Words.* Harvard University Press, 1962.
4. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. Contextualizing ontologies. *Journal of Web Semantics*, 1(4):325–343, 2004.

5. K. S. Decker. TÆMS: A framework for environment centered analysis & design of coordination mechanisms. In *In Foundations of Distributed Artificial Intelligence, Chapter 16*, pages 429–448. Wiley, 1996.
6. V. Dignum. *A Model for Organizational Interaction: based on Agents, founded in Logic.* PhD thesis, Universiteit Utrecht, 2004.
7. European FP-7 Project. Coordination, organisation and model driven approaches for dynamic, flexible, robust software and services engineering (ALIVE), http://www.ist-alive.eu/.
8. C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.
9. D. Grossi. *Designing Invisible Handcuffs: Formal Investigations in Institutions and Organizations for Multi-agent Systems.* PhD thesis, Universiteit Utrecht, 2007.
10. D. Grossi, H. Aldewereld, J. Vázquez-Salceda, and F. Dignum. Ontological aspects of the implementation of norms in agent-based electronic institutions. In *Computational and Mathematical Organization Theory*, pages 104–116. AISB, 2005.
11. A. Jones and M. Sergot. A formal characterization of institutionalised power. *Journal of the IGPL*, 3:427–443, 1996.
12. V. Lesser, K. Decker, and T. Wagner. Evolution of the GPGP/TÆMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, 2004.
13. J. Searle. *Speech acts. An essay in the philosphy of language.* Cambridge University Press, 1969.
14. JBoss Community. JBoss drools business rules, http://www.jboss.org/drools.
15. W. W. Vasconcelos, J. Sabater, C. Sierra, and J. Querol. Skeleton-based agent development for electronic institutions. In *In Proc. AAMAS'02*, pages 696–703. ACM press, 2002.
16. J. Vázquez-Salceda, V. Dignum, and F. Dignum. Organising multiagent systems. *JAAMAS*, 11(3):307–360, November 2005.
17. R. Wieringa. *Requirements Engineering: Frameworks for Understanding.* John Wiley & Sons, Inc., 1996.