Towards Runtime Support for Norm-Governed Multi-Agent Systems

Visara Urovi*, Stefano Bromuri*, Kostas Stathis*, Alexander Artikis**
*Department of Computer Science, Royal Holloway, University of London, UK.
**Institute of Informatics and Telecommunications, NCSR Demokritos, Greece.

Abstract

We present a knowledge representation framework with an associated run-time support infrastructure that is able to compute, for the benefit of the members of a norm-governed multi-agent system, physically possible and/or permitted actions current at each time, as well as sanctions that should be applied to violations of prohibitions. Experimental results on a benchmark scenario indicate how by distributing norms we can provide run-time support to large-scale, norm-governed multi-agent systems.

Introduction

Norm-governed multi-agent systems are systems in which actuality does not necessarily coincide with ideality, and thus the agents' interactions are regulated by permissions, obligations, and other normative relations that may exist between them. Despite the proliferation of knowledge representation frameworks for norm-governed systems, these frameworks often focus on the expressive power of the formalism proposed and typically abstract away from the computational aspects and experimentation. If the computational behavior is studied, then this often happens in isolation, at times theoretically only, and in many occasions leaves unexplored any experimental evaluation.

Our work aims at using existing Event Calculi (Kesim and Sergot 1996; Bromuri and Stathis 2009) for computing, at run-time, permissions, prohibitions, and sanctions dealing with the performance of forbidden actions. We assume that agents cannot compute these normative relations on their own because of computational constraints, and incomplete knowledge about the application state. The novelty of our approach is the ability to formulate the distribution of the physical and social environments of a norm-governed application in order to efficiently compute their corresponding physical and social states.

The Open Packet World

To exemplify our approach we use *Packet World* (Weyns, Helleboogh, and Holvoet 2005), an application in which a set of agents situated in a rectangular grid pick colored packets (squares) and deliver them in destinations (circles) matching a packet's color (see Fig. 1(a)(i)). Agents see only

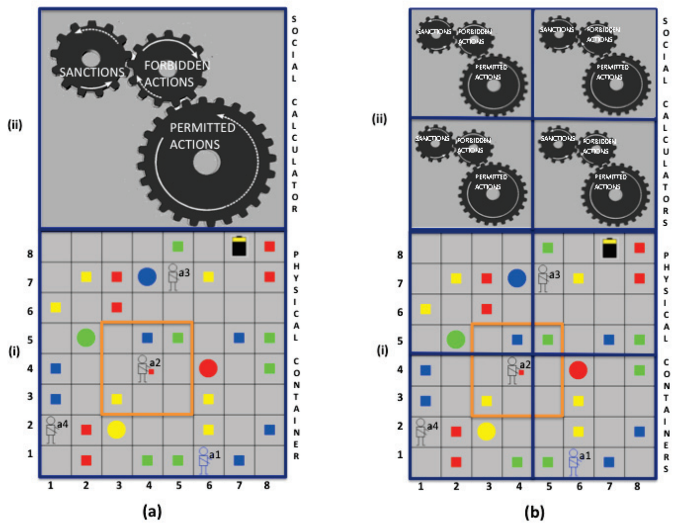

Figure 1: Open Packet-World as a Norm-Governed System

part of the grid (e.g. the square around agent a2 in Fig. 1 is its perception range), and are assumed to be collaborative. Also, agents are powered by a battery that discharges when they move. Recharging the battery requires a charger (located at (7,8) of Fig. 1). The charger emits a perceivable gradient; a small/large value implies that the distance from the charger is close/far respectively.

We introduce a competitive version of the above scenario, the *Open Packet World* (OPW), where agents score points when they deliver packets, and to ensure that they score the most points, they may deceive others e.g. by placing a flag indicating that there no packets left in the surrounding area, although this may not be the case. To deal with (such) unsocial behavior, we introduce norms in OPW — e.g. an agent is not permitted to flag an unexplored area as it will mislead others to think that there are no packets in this part of the grid. Violation of norms results in sanctions, for example, the reduction of points of the violating agent.

We experiment with our scenario using the GOLEM agent platform[1]. GOLEM supports the deployment of *agents* - cognitive entities that can reason about sensory input received from the environment and act upon it, *objects* - re-

[1]http://golem.cs.rhul.ac.uk

sources that lack cognitive ability, and *containers* - virtual spaces containing agents and objects, capturing their ongoing interactions in terms of an *event-based* approach.

The simplest way to model the OPW in GOLEM is shown in Fig. 1(a). Here we deploy a container representing the world (Bromuri and Stathis 2007) extended with an active object which we call *Social Calculator* (Artikis, Sergot, and Pitt 2009). This object contains the norms, encapsulates the state of the physical container in order to check for violations in it, extends it with a social state by storing possible violations, while at the same time serves agents who would want to know what their permissions are at a specific time.

To represent the state of a GOLEM container we use the object-based notation of C-logic, a formalism that describes objects as complex terms (Chen and Warren 1989). The term below, e.g., represents the state of a 2 x 2 packet world showing only one agent, packet, destination and battery:

```
packet_world:c1[
  address ⇒ "container://one@134.219.7.1:13000",
  type ⇒ open,
  grid ⇒ {square:sq1, square:sq2, square:sq3, square:sq4}
  entities ⇒ {picker:ag1, packet:p1, dest:d1, battery:b1}]
```

Object instances belong to classes (e.g. packet_world), are characterized by unique identifiers (e.g. c1), and have attributes with single values (e.g. address) or multiple values (e.g grid). The representation of the 8 x 8 grid of Fig. 1 is similar but larger, i.e. more agents, packets, destinations, and squares.

Complex objects evolve as a result of events happening in the state of a container (Bromuri and Stathis 2009). To query the value Val of an attribute Attr for an entity Id of container C at a specific time T, we will use the definition:

```
solve_at(C, Id, Class, Attr, Val, T) ←
     holds_at(C, container, entity_of, Id, T),
     holds_at(Id, Class, Attr, Val, T).
```

holds_at/5 extends the Event Calculus with an object-based data-model (Kesim and Sergot 1996). The extension describes how the value Val of an attribute Attr for specific Class instance identified by Id holds at a particular time T. For details the reader is referred to (Kesim and Sergot 1996). With this extended Event Calculus we specify physical possibility for the OPW as, e.g.:

```
possible(E, T)←
   do:E [actor ⇒ A, act ⇒ move, location⇒ SqB],
   solve_at(this, A, picker, position, SqA, T),
   adjacent(SqA, SqB),
   not occupied(SqB, T).
```

The rule states that it is possible for an agent to move to an adjacent position as long as it is not occupied. The keyword **this** is used here to refer to the identifier of the current container.

We can now formalize the social state of a system as a C-logic structure that extends the physical state with social attributes to hold information about any current sanctions imposed on any of the agents, and the points agents have collected so far. An example snapshot of a social state for the OPW is shown below:

```
packet_world_social_state: s1 [
    physical_state⇒ packet_world:c1,
    sanctions⇒ {sanction:s1 [agent ⇒ a2, ticket ⇒ 5]},
    records⇒ {record:r1[agent ⇒ a1, points ⇒ 35],
               record:r2[agent ⇒ a2, points ⇒ 25]}]
```

The term above states that agent a2 has been sanctioned with 5 points. We show the records of two agents only to save space. Agent a1 has collected 35 points, while a2 has collected 25 after the sanction is applied. The social state contains rules for what is permitted and what is forbidden :

```
permitted(Event, T)← not forbidden(Event, T).

forbidden(E, T) ←
   do:E[actor ⇒ A, act⇒drop, object⇒flag, location⇒SqA],
   solve_at(this, Id, packet, position, SqB, T),
   adjacent(SqA, SqB).
```

When a forbidden act has taken place, the Social Calculator raises a violation. More complex permissions and sanctions can be formalized similarly.

An alternative way to model the OPW is to split the physical state of a single container into smaller states that we distribute into different containers. Fig. 1(b)(i) shows four 4 x 4 adjacent containers for OPW together with their corresponding Social Calculators (see Fig. 1(b)(ii)). GOLEM supports this feature with the *Ambient Event Calculus* (AEC) (Bromuri and Stathis 2009). Given a container C and a starting Path, we can query a maximum number of neighbors Max, returning a final Path* where an object identifier Id, class Cls, attribute Attr, and value Val hold at time T:

```
neighbouring_at(C, Path, Path*, Max, Id, Cls, Attr, Val, T)←
   Max >= 0,
   locally_at(C, Path, Path*, Id, Cls, Attr, Val, T).
neighbouring_at(C, Path, Path*, Max, Id, Cls, Attr, Val, T)←
   holds_at(C, container, neighbour, N, T),
   not member(N, Path),
   Max* is Max - 1,
   append(Path, [C], New),
   neighbouring_at(N, New, Path*, Max*, Id, Cls, Attr, Val, T).
```

The first clause checks whether the object is in the local state of a container. locally_at/8 checks with holds_at/5 to find the object in the container's state, including sub-containers[2], if any. The second clause looks for neighbors. If a new neighbor N is found, this neighbor is asked the query but in the context of a New path and a new Max*.

We are now in a position to customize our representation for distributing the physical and social state by redefining the solve_at/6. The definition below has the effect of changing all the physical and social rules so that they can work with distributed containers:

```
solve_at(C, Id, Class, Attr, Val, T) ←
     neighbouring_at(C, [], _, 1, Id, Class, Attr, Val, T).
```

The empty list [] above states that the initial path is empty, the underscore '_', that we are not interested in the resulting path, and the number 1 indicates that we should look at all neighbors whose distance is one step from the current container. In this way, we can query all the neighbors of a container in the OPW of Fig. 1(b).

[2]We refer the interested reader to (Bromuri and Stathis 2009) for a definition of locally_at/8.

Experimentation

We performed two sets of experiments: one where the OPW is deployed in a single container and another where it is deployed in many distributed containers. In both sets of experiments we measured the time needed to compute whether an action is physically possible and/or permitted. We also varied the number of agents participating in OPW, observed the number of events in the system, and observed how these parameters affect the performance of the system in both experimental settings.

In the first set of experiments the environment was represented by a 40x40 grid and 100 packets were collected by the agents and released into one of the 8 destinations in the grid. We run the first test with 10 agents, the second test with 30 agents and the third test with 50 agents. We found that the time needed to compute the social and physical state, in a single container setting, is proportional to the number of events taking place in OPW.

In the second series of experiments we distributed the OPW grid (40x40) first into two containers (20x40) and then into four (20x20) different containers. For the distribution of the containers we used an Intel Centrino Core 2 Duo 2.66GHz with 4GB of RAM and an Intel Centrino Core Duo 1.66Ghz with 1GB of RAM. Agents were deployed in distributed containers and were mobile (Bromuri and Stathis 2009). Figure 2 shows the experimental results. It shows that in a system with a small number of events (0-500), it is better to adopt a single container setting. With an increasing number of events, however, we achieve a considerable performance gain by distributing the grid into two or four containers. In general, we found that when we distribute the agent environment into multiple containers, the time to compute the physical and the social state is inversionally proportional to the number of containers, thus improving the performance. However, there is an additional delay to compute the physical and social state which is due to the interactions between the containers (e.g. when an agent moves from the grid under supervision of one container to the grid supervised by another container). A detailed discussion on our experimental evaluation and the implementation can be found in (Urovi et al. 2010).

Conclusions and Future Work

We presented a knowledge representation framework with an associated run-time infrastructure that is able to compute, for the benefit of the members of a norm-governed multi-agent system, the physically possible and permitted actions current at each time, as well as the sanctions that should be applied to violations of prohibitions. We exemplified the ideas by applying the infrastructure on a benchmark scenario for norm-governed multi-agent systems. Through experimentation we explored how to use the knowledge representation framework to distribute parts of the infrastructure so that we can provide run-time support to larger-scale multi-agent systems regulated by norms.

There are several directions for further work. First, we are examining various caching mechanisms for the Event Calculus, such as those proposed in (Chittaro and Montanari 1996), in order to further improve the efficiency of temporal reasoning. Second, we aim to perform experiments with larger multi-agent systems in order to determine the extent to which our infrastructure can be used for run-time support. Third, we aim to formalise additional normative relations, such as institutional power.

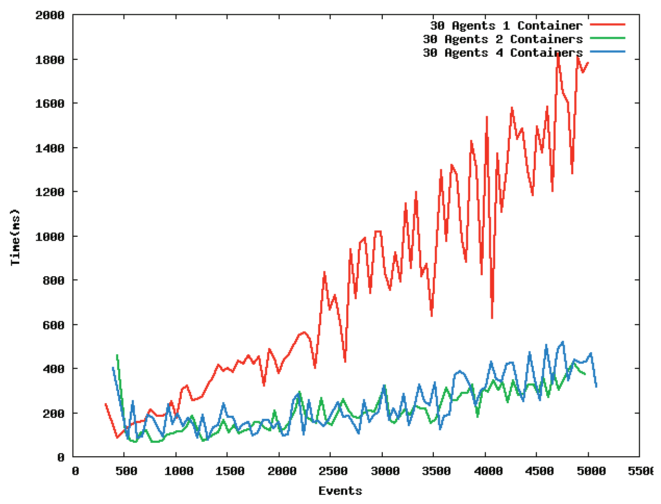

Figure 2: Experimental Results

References

Artikis, A.; Sergot, M.; and Pitt, J. 2009. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic* 10(1).

Bromuri, S., and Stathis, K. 2007. Situating Cognitive Agents in GOLEM. In *Engineering Environment-Mediated Multi-Agent Systems*, LNCS 5049, 115–134. Springer.

Bromuri, S., and Stathis, K. 2009. Distributed Agent Environments in the Ambient Event Calculus. In *Proceedings of conference on Distributed event-based systems*. ACM.

Chen, W., and Warren, D. S. 1989. C-logic of complex objects. In *Proceedings of Symposium on Principles of database systems*, 369–378. ACM.

Chittaro, L., and Montanari, A. 1996. Efficient temporal reasoning in the cached event calculus. *Computational Intelligence* 12:359–382.

Kesim, F. N., and Sergot, M. 1996. A Logic Programming Framework for Modeling Temporal Objects. *IEEE TKDE* 8(5):724–741.

Urovi, V.; Bromuri, S.; Stathis, K.; and Artikis, A. 2010. Run-time support for norm-governed systems. CS Technical Report CSD-TR-10-01, Royal Holloway. `http://golem.cs.rhul.ac.uk/TR/CSD-TR-10-01.pdf`.

Weyns, D.; Helleboogh, A.; and Holvoet, T. 2005. The packet-world: A testbed for investigating situated multiagent systems. In *Software Agent-Based Applications, Platforms, and Development Kits*. Birkhauser Verlag. 383–408.