Deontic Sensors

Julian Padget, Marina De Vos and Charlie Ann Page
Department of Computer Science, University of Bath, United Kingdom
{j.a.padget,m.d.vos,c.a.page}@bath.ac.uk

Abstract

Normative capabilities in multi-agent systems (MAS) can be represented within agents, separately as institutions, or a blend of the two. This paper addresses how to extend the principles of open MAS to the provision of normative reasoning capabilities, which are currently either embedded in existing MAS platforms – tightly coupled and inaccessible – or not present.

We use a resource-oriented architecture (ROA) pattern, that we call deontic sensors, to make normative reasoning part of an open MAS architecture. The pattern specifies how to loosely couple MAS and normative frameworks, such that each is agnostic of the other, while augmenting the brute facts that an agent perceives with institutional facts, that capture each institution’s interpretation of an agent’s action. In consequence, a MAS without normative capabilities can acquire them, and an embedded normative framework can be de-coupled and opened to other MAS platforms.

More importantly, the deontic sensor pattern allows normative reasoning to be published as services, opening routes to certification and re-use, creation of (formalized) trust and non-specialist access to “on demand” normative reasoning.

1 Introduction

The concepts of norm and institution [North, 1991; Ostrom, 1990] have been imported into artificial intelligence from the humanities as a way to capture formal and informal contextualized expectations of behaviour, and hence to influence the decision-making of autonomous systems, through statements of what ought and ought not to hold and what actions are forbidden, permitted or obliged [von Wright, 1951]. Normative reasoning has been particularly influential in the area of autonomous agents and multiagent systems (MAS), and given its social origins, it is not surprising that knowledge represented as norms is seen too as an appropriate lingua franca to govern actors in socio-technical systems (STS). These together provide the context for the work on engineering for norms presented here.

Some implementations of normative reasoning are stand alone, but embedding in an agent platform is more common (see [Aldewereld et al., 2016] for a detailed comparative evaluation of contemporary examples). The integrated approach has benefits and drawbacks: users must ‘buy the package’ of agents and norms, cannot readily use the normative component alone and cannot readily use an alternative normative component. More importantly, for the development of norm representation and reasoning as a topic and as an AI artefact, normative knowledge represented on one platform cannot typically be shared and re-used. Furthermore, lack of transparency in representation and reasoning, and means for external verification, inhibit establishment of trust.

Reaction to the silo approach of agents + norms is found in several works, including the Environment Interface Standard (EIS) [Behrens et al., 2011], the Alive [Vázquez-Salceda et al., 2010; Carlos Nieves et al., 2011] service-oriented architecture (SOA)1, the THOMAS [del Val Nogueira et al., 2010] platform, and the JaCaMo [Boissier et al., 2016] framework. Each of these tries in different ways to break up aspects of the agent platform in order to allow substitution and re-use of components. They are discussed in more detail in section 6.

It is against the backdrop of the above efforts and the preceding discussion that we introduce the concept of the deontic sensor, and show how to create and deploy such sensors so they can be used freely by any web-client capable platform, through the use of a web framework into which normative reasoning tools can be inserted. We define a deontic sensor as a transducer that observes brute (real world) facts and interprets them to generate institutional (sometimes also called social) facts [Searle, 1995]. The contributions of the paper are: (i) the conceptualisation of norm representation and reasoning as a deontic sensor (section 2); (ii) a ROA design pattern for deontic sensors which makes the concept concrete but has a pattern’s intrinsic scope for variation in implementation (section 3); (iii) an instantiation of the pattern (section 4) and its evaluation using the Jason [Bordini et al., 2007] agent platform and the InstAL [Padget et al., 2016] normative framework (section 5).

By establishing the deontic sensor as a ROA pattern for

1 SOA is a precursor to ROA that uses so-called arbitrary SOAP web services, rather than ROA’s RESTful services [Fielding, 2000].
normative reasoning, we aim to pave the way to more open norm-aware multiagent systems, while published and discoverable norm reasoning services can open routes to certification/validation and re-use of normative reasoning components, build the foundation for trust in such systems, and enable non-specialist access to normative reasoning.

2 Requirements

The following naïve diagram captures the primary intention of providing normative reasoning for an agent platform:

![Diagram](Figure 1: The situated (multi)agent system...)

Figure 2: ...and its (multi)institutional extension

in which agents’ actions (brute facts) pass from the agent platform, to be observed by the deontic sensor, and their interpretation, as normative (social) facts, goes in the other direction to be perceived by agents.

In the remainder of this section, we discuss the requirements that lead to the architecture in section 3. We analyse requirements arising from a consideration of what various kinds of users might want of the deontic sensor architecture. We use the term institution to refer to a (typically) coherent set of norms that it is intended should guide agent behaviour towards the achievement of one or more goals. An institutional model is the computational representation [Fornara et al., 2013] of an institution [Grossi et al., 2006] and a deontic sensor is its instantiation for a given set of literals.

2.1 Institutional Models

We begin by considering the state of an institution, the nature of that state and the operations it might be desirable to support on that state. The deontic sensor creates a resource from the institutional states of all participating institutions within each normative reasoning framework. In abstract terms, the system tracks a sequence of actions performed by agents, each changing the state in the instances of the institutions models that are observing them. The representation and its corresponding computation are invisible, and an instantiation of a particular normative framework with a particular set of norms at a specific time can be regarded as a resource to manipulate through exogenous events, such as agent actions. To realise this institutional repository, it must be possible to populate the deontic sensor platform with (abstract) institution specifications, and instantiate and verify the corresponding models. Hence the initial user stories (US) [Beck, 2000]:

US1: As an institution designer, I want to be able to add new institutions to the deontic sensor;
US2: As an institution designer, I want to be able to create an instance of an institution;
US3: As an institution designer, I want to be able to verify the normative interpretation of an action (and sequences of actions) by an institution;
US4: As an institution designer, I want to be able to investigate the normative interpretations of an action and sequences of actions in the context of one or more (interacting) institutions;
US5: As a system developer, I want to be able to create an instance of an institution;
US6: As a system developer, I want to connect an agent platform and a deontic sensor.

2.2 Agents and Institutions

A common model of a situated agent interacting with its environment in a multi-agent system (Fig. 1) sees the agent receiving information about the environment through percepts and using those percepts to make action selection decisions.

We extend this model to add an explicit, referenceable, normative aspect, represented as an institution (Fig. 2), that observes agents’ actions and interprets them according to the norms of a given institution, giving rise to the institutionally situated agent. Thus, in addition to receiving brute information through conventional environmental percepts, an agent may additionally sense the deontic commentary coming from institutions and acquire normative information that can be incorporated into its reasoning process as beliefs [Dybalova et al., 2013; Lee et al., 2014; van Riemsdijk et al., 2013], assuming for sake of argument a BDI agent. In this way, different institutions, governing concurrently, may deliver compatible or conflicting normative interpretations of an agent’s behaviour, as well as identifying instances of non-compliant behaviour that the subject or other agents in the MAS may decide to act upon. We emphasize that institutions as presented here, are observers, not actors (or enforcers) in a MAS. It is their purpose to provide actors with normative interpretations of actions, while it is the actors who are responsible for using them to choose what to do. Agents’ chosen or hypothetical actions are communicated to the deontic sensors via the platform. This suggests the following additional user stories:

US7: As an agent platform, I want to be able to create and instance an institution;
US8: As an agent platform, I want to communicate that action a has taken place and get a normative interpretation of that action;
US9: As an agent platform, I want to find out the normative interpretation of action a without enacting a (correct subject to the actions of other agents);
US10: As an agent platform, I want to pass normative information back to agents concerned;
US11: As a norm aware agent, I want to be able to perceive normative information.
3 The Deontic Sensor Architecture

Our aim is an architecture for normative reasoning in large scale (open) multi-agent systems that is model independent and distributed computation is an essential aspect of large scale systems. As our naïve diagram at the start Section 2 indicates, we want to decouple the agent platform from the institution platform, so that computation can be separated and distributed. This motivates the proposal for deontic sensors as a Resource Oriented Architecture (ROA) pattern.

A pattern is not an environment interface specification or application program interface (API). These both prescribe functionality that must be or is present, at different levels of detail, forcing new and existing clients to adhere to a particular interaction protocol using specified data formats. In contrast a ROA pattern describes a collection of Create, Read, Update and Delete (CRUD) [Fielding, 2000] operations over a set of resources through the (abstract) endpoints AEP1–AEP4 (see Table 1), in this case to realise deontic sensors.

The nature of a ROA decouples the service – in this case an institutional model – from its clients – in this case an agent platform – meaning the service may be implemented by using any institution platform that is able to track the state of its institution(s) and allows queries, as per US1-11.

3.1 Abstract Resource Architecture

Working from the user stories (US1–11) set out in Section 2, we propose the deontic sensor architecture in Figure 3, while Table 1 details the abstract endpoints.

The architecture expands upon the naïve diagram with an agent platform (AP) and a deontic sensor platform (DSP). The AP creates new institutional models on-demand (supplying its own specifications) using AEP1, and creates as many instances of each of those, again on-demand, via AEP2. Agents themselves are unaware of the DSP, but agent actions detected by the AP are passed to the DSP through AEP3, creating fresh institutional states that reflect the observation of the action. Deontic commentary on agent actions is provided by AEP4, which the AP then makes available for its host agents to sense in whatever form is appropriate for the platform. Because action communication and normative interpretation are handled by the AP, its agents can be agnostic about, or even ignorant of the DSP. Norm-aware agents, however, may want to know the normative consequences of an action or sequence of actions beforehand, or want to know how to achieve a particular normative state. Answers to these questions depend on the capabilities of the particular DSP, so although the endpoint to use (AEP4) does not change, and neither does the AP’s role as a mediator (see e.g. A3 in Fig-

Table 1: The three Create and one Read operations of the abstract and concrete deontic sensor ROA
ure 3), such queries establish a dependence between agent
and a DSP through the use of such functionality and through
the language used to express the question (but see the end of
section 3.2). The DSP itself is stateless (in line with RESTful
practice); AEP3 provides the means to read each state that is
created and to create as many states as actions or speculative
queries require, but it is the AP’s responsibility to track which
resource is the current canonical state for each DS it is using.

Outside the AP, we note the role of institution designer,
who can introduce new models (AEP1), instantiate models
and query instances for verification purposes (US1–4).

3.2 Action Monitoring

For an institution to be able to provide its deontic com-
mentary, it needs data about the events that occur in the environ-
ment through the actions of agents. This corresponds to the
observe arc in Fig. 2. At the conceptual level, this is the in-
titutional equivalent of an agent ‘seeing’ a percept. At the
operational level, actions and their representation need to be
delivered in a form that an institution can process and the ac-
tion vocabulary must overlap with that of an institution if it is
to recognise the action and interpret it against the institutional
state. The deontic sensor pattern is neutral on these issues.
We have taken pragmatic decisions in the sample architecture
in Section 4, but interoperability via a standard set of oper-
ators and representation is an open issue. Possible solutions
might be the approach adopted in FIPA ACL of specifying
content language and ontology in the envelope, and building
on the semantic sensor network ontology [W3C, 2011].

3.3 Deontic Commentary

The agent senses the normative context as percepts in an
equivalent manner to its sensing other information about its en-
environment. This corresponds to the interpret arc in Fig. 2.
We refer to the ongoing normative information being pro-
vided to agents in the form of percepts as a deontic com-
mentary. Following deontic logic, there are three kinds of
percepts an agent can receive from an institution, regarding
an action A:

\[ \text{pern}(A): \text{Action } A \text{ is permitted, written } PA \text{ in deontic}\]
\[ \text{pro}(A): \text{Action } A \text{ is prohibited (forbidden), written } \neg PA\]
\[ \text{obl}(A, [D, V]): \text{The agent is obliged to perform action } A\]
\[ \text{written } OA.\]

We emphasize that these are the terms used to ground the
architecture, that the percept language can be whatever is fit
for purpose, and reiterate the point made in Section 3.2 above
regarding operators, representation and encoding.

An agent may respond (or not) how it pleases to conven-
tional environmental percepts and the same applies to norma-
tive percepts. Thus, receiving a percept pro(action) does not
mean that an agent is unable to perform action, but – assum-
ing it is a norm-aware agent – that it should not. The norm-
unaware agent may simply disregard it as an unrecognised
percept. An agent’s behaviour is not necessarily regimented
with respect to the institutions it is participating in, although
it would certainly be possible to use the information from the
institution to make it so. Our intention here is to provide nor-
mative information alongside other percepts, as input to an
agent’s decision-making process, on the assumption that the
agent is capable of recognising and the platform is capable of
forwarding normative percepts.

At the architecture level, we intentionally only specify an
abstract syntax for an example set of operators, but it is clear
that the pattern can simultaneously admit different institu-
tional models, which could either deliver normative percepts
in a common language or equally use distinct representations
and semantics, which must be resolved either by agent or plat-
form. The essential feature is that agent normative reasoning
is decoupled from the institutions themselves.

4 A Sample Architecture

We illustrate the deontic sensor pattern using Jason [Bor-
dini et al., 2007] for the agents and the environment, and
InstAL (Institutional Action Language) [Padget et al., 2016]
for the normative framework. InstAL specifications compile
to AnsProlog [Baral, 2003], which following grounding pro-
duces a model of the institution that can be queried through
presenting it with sequences of one or more events, leading
to the generation of one or more answer sets containing nor-
mative interpretations of the action(s)\(^3\).

InstAL-REST\(^2\) provides InstAL as a service with a REST-
ful interface: this supports the deployment of multiple institu-
tional models on a server, and makes each institution queryable
via standard HTTP operations. Queries and results are encoded
using JSON. InstAL-REST is deployed using Docker, a tool for running containerised applications.
InstAL-REST uses four supporting processes, but can be
launched with a single command using docker-compose. The
four additional components are: a work queue (we use Cel-
ery), a messaging system (we use RabbitMQ), web server
(we use Gunicorn) and a database (we use Postgres). InstAL-
REST provides three resources, following Table 1:

- **Model** (CEP1): Contains InstAL files (.ial) and AnsProlog
  files (.lp). The model resource provides an endpoint for
  the creation of an instance of the model.

- **Instance** (CEP2): Contains values for all of the types in
  the model, thus allowing the answer set solver to con-
  struct a variable-free version of the model (this process
  is called grounding). The instance resource provides an
  endpoint for the grounded model to be queried.

- **Query** (CEP3): The execution. It is at this stage that the
  institutional model is actually run, with respect to some
  user-provided parameters. The query resource provides
  endpoints for the client to access the result of the InstAL
  run in a variety of different formats, as discussed in sec-
  tion 3.1 and detailed in Table 1.

The Jason platform provides both the agent and environ-
ment parts of Fig. 1, where an agent senses through percepts
it receives from the environment, then uses those percepts to
reason about how it should act, while the environment re-
ceives and processes the actions of agents and provides per-

\(^2\)We use the clingo ASP solver (https://github.com/potassco/clingo) with a Python wrapper

\(^3\)Code available at https://github.com/instsuite/instal-rest/
cepts to agents based on the environmental state. We provide a Java interface for the Jason environment called Institutional, which provides the blueprint for extending the environment to incorporate the observer-interpret cycle for institutions (Figure 2). The interface specifies the form of how an environment should inform the institution of its state and its actions, and how the environment should translate the output from querying the institution into percepts. That is, the interface determines which endpoints of the architecture are to be used and how the data flows from the AP to the DSP (Fig. 3).

Jason, InstAL and JSON are our choices for the sample architecture, but other agent platforms, institutional modelling software and data encodings could equally be used. The pattern describes the relationships between the environment, the agent, and the institution, but not the specifics of the implementation of any of them—they are black-boxes—or the nature of the data that passes between them. Thus, the pattern explicitly decouples the execution of its component parts, and does not address coupling arising from data meaning or representation. Integrating a new tool for the institution and/or the environment would not require modifications to the environment or the institution. It would, however, require modifications to the interface code between the parts. Consequently, the pattern allows substitution of environment, institution, or agent, as long as the percepts use the same conventions and compatible data representations—or transcoding is employed.

We conclude with a small example that demonstrates what the pattern enables. An agent (Alice) wishes to carry out an action action, which is governed by two institutions, a and b. To perform action, the agent must apply for a permit from both institutions. The stages are as follows (see Listing 1):

1. Initially, both institutions prohibit the agent from performing action (line 2).
2. The agent applies for a permit from institution a (line 3).
3. The agent is now permitted to perform action by institution a, but still prohibited by institution b (line 6).
4. The agent applies for a permit from institution b (line 7).
5. The agent is now permitted to perform action by both institutions (line 10).
6. The agent now performs action, and is permitted to do so (line 12).

The full log output appears in Listing 1, with the corresponding agent code in Listing 2. We use Jason’s annotation mech-
Table 2: Batched queries using 12 (Celery) workers

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Average time</th>
<th>Stdev</th>
<th>Average time/query</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 queries</td>
<td>0.546</td>
<td>0.143</td>
<td>0.273</td>
</tr>
<tr>
<td>25 queries</td>
<td>1.21</td>
<td>0.387</td>
<td>0.0485</td>
</tr>
<tr>
<td>50 queries</td>
<td>1.82</td>
<td>0.529</td>
<td>0.0364</td>
</tr>
<tr>
<td>100 queries</td>
<td>2.40</td>
<td>0.592</td>
<td>0.0240</td>
</tr>
<tr>
<td>250 queries</td>
<td>3.49</td>
<td>0.268</td>
<td>0.0140</td>
</tr>
</tbody>
</table>

Table 3: Simulated agent workload

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Average time/agent</th>
<th>Stdev average time/agent</th>
<th>Average total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 agent 2 queries</td>
<td>0.547</td>
<td>0.089</td>
<td>0.628</td>
</tr>
<tr>
<td>5 agents 5 queries</td>
<td>0.890</td>
<td>0.275</td>
<td>1.80</td>
</tr>
<tr>
<td>10 agents 5 queries</td>
<td>0.958</td>
<td>0.153</td>
<td>1.80</td>
</tr>
<tr>
<td>10 agents 10 queries</td>
<td>1.45</td>
<td>0.194</td>
<td>2.44</td>
</tr>
<tr>
<td>5 agents 10 queries</td>
<td>2.48</td>
<td>0.135</td>
<td>4.38</td>
</tr>
</tbody>
</table>

6 Discussion

To place deontic sensors in the wider context of normative multi-agent systems we examine four themes: interoperability, agent platforms (APs) that are norm agnostic, APs with embedded norms and APs that work with services. We aim to provide an overview, rather than be exhaustive, and contrast our approach with a representative selection of alternatives.

Behrens et al. [2011] aims to address interoperability by defining an environment interface standard (EIS), which mediates and decouples the act-sense cycle between agent platform and environment. It is described as “…a Java-based interface standard for connecting agents to controllable entities in an environment such as a game. The interface provides support for managing the connection…” [Behrens et al., 2011]. In this way EIS is a classical interface pattern, aiming to replace an $n \times n$ connections by $n \times 1$ and $1 \times n$ connections. Currently, EIS is actively supported by GOAL [Hindriks and Dix, 2014], code is provided in the Jason [Bordini et al., 2007] distribution, and the sourceforge version of 2APL does not support EIS, but through basing the new implementation [Dastani and Testerink, 2016] on design patterns, it is claimed to obviate the need for adherence to EIS. The point of contrast with the deontic sensors pattern is that EIS also advocates for the decoupling of of MAS components, in this case, agents and environment, but it addresses the problem through the definition of an API, with specified signatures, rather than the more abstract technique of a pattern.

The three platforms above, and other APs (e.g. JADEX [Pokahr et al., 2013]), are norm-agnostic: they do not contain a normative framework. Like Jason, these are suitable APs for instantiation of the deontic sensor pattern described here. The point for discussion is these significant, maintained and widely-used agent platforms could be enhanced through access to explicit norm frameworks, although the agent’s internal reasoning would need extension (e.g. [Dybalova et al., 2013; Lee et al., 2014]) to account for normative percepts.

The third topic is APs with embedded normative frame-works. We highlight three approaches: (i) the Electronic Institutions Development Environment (EIDE) and the associated Ameli simulation platform [Esteva et al., 2004], where norms are explicit, and agents act through governors, that block non-compliant actions, so agents are effectively norm-regimented; (ii) ROMAS-Magenta2 [García et al., 2016] provides an (explicit) approach in which norms are defined for roles, affecting agents who play a role, and for organizations, affecting all members of an organization, covering the range of constitutive, regulative and procedural norms; (iii) the JaCaMo [Boissier et al., 2016] framework combines agents (Jason), environment (CArtAgO) and organization (MOISE), where MOISE provides role-based organizational norms and CArtAgO provides services, including normative artifacts (sic) whose norms (expressed in Normative Organization Programming Language (NOPL) [Hübner et al., 2011]) constrain the behaviour of an agent wishing to manipulate an artifact. These three platforms illustrate alternative approaches to embedding normative frameworks (via governors, roles, organizations and artifacts), demonstrating a richness that appears to be inaccessible except to agents written for the particular platform. The norm part of Magentix2 is described as a norm reasoning service and CArtAgO is described as using a service-oriented style, but they appear not to be addressable as web services. What these platforms highlight is the existence of APs with embedded normative frameworks that could be refractored into deontic sensors and hence made accessible to norm-agnostic platforms as well as opening these APs up to other normative frameworks.

Fourthly, we consider two approaches that apply service-oriented concepts to MAS: (i) Alive [Vázquez-Salceda et al., 2010] puts forward a three level (organization, coordination, and services) architecture in which semantic web services are used for discovery and delivery of services to a coordination layer of software agents, that participate in organizational structures whose roles encapsulate norms; (ii) THOMAS [del Val Noguera et al., 2010] uses semantic web services to implement a virtual organization (VO), agent actions are invoked through service requests, and norm processing is handled by the THOMAS platform, using a rule engine. The VO aspect has been subsumed into Magentix2 without the semantic web services and Alive is no longer maintained. These platforms show efforts to address the MAS silo issue using services, but through service-oriented architecture, which does not decouple as fully as the pattern approach adopted here, while THOMAS has apparently abandoned (semantic) web services and Alive is no longer maintained, which may indicate lessons in the use of arbitrary web services.

In conclusion, we see the deontic sensor pattern as a valuable contribution to decoupling agents and institutions, in design and implementation, allowing independent development, by refactoring a key MAS component into a service. We hope this work encourages further deconstruction, and on-demand delivery of component AI as a service, for other MAS aspects such as environmental models and agent reasoning. Next steps are: (i) distribution of deontic sensors across compute resources, (ii) application in agent-based simulation, and (iii) investigation of semantic technologies (e.g. [W3C, 2011]) to address the semiotics of deontic sensors.
References


